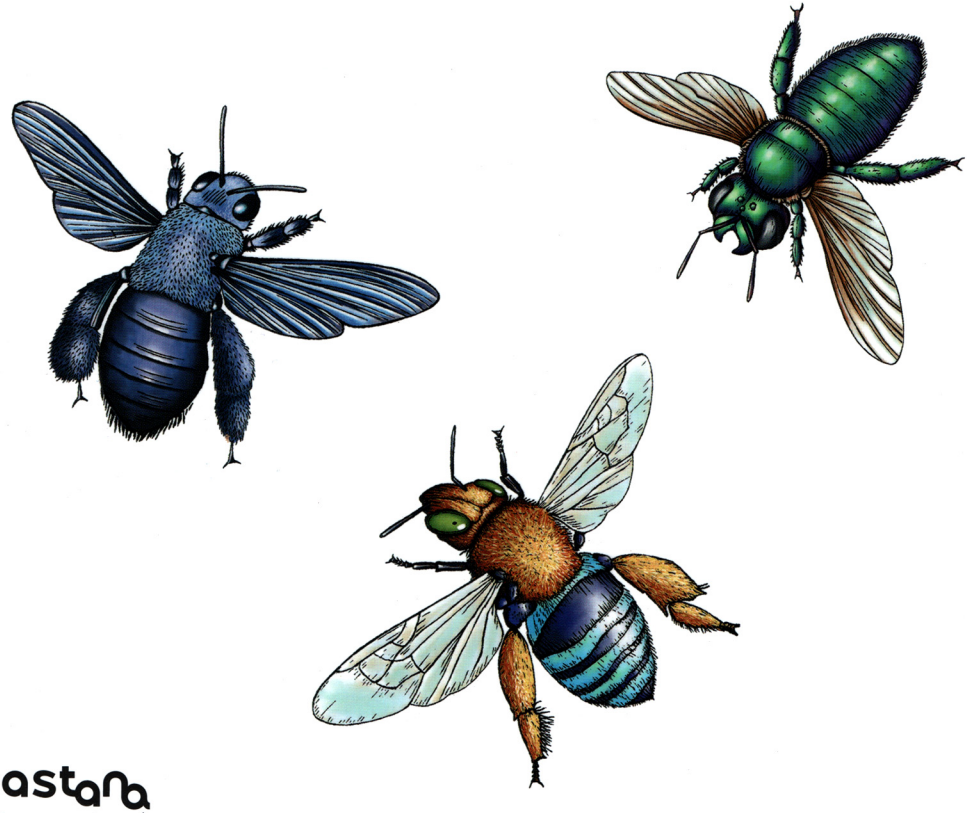
O'REILLY®

2-е издание

Визуализация данных с помощью Python и JavaScript

Анализ и преобразование данных



astana
international
publishing

Киран Дейл

O'REILLY®

Книги по программированию

Data Visualization with Python and JavaScript

Scrape, Clean, Explore, and Transform Your Data

Kyran Dale

Визуализация данных с помощью Python и JavaScript

Анализ и преобразование данных

Киран Дейл

УДК 004.4
ББК 32.973.26-018.2
Д27

Kyran Dale

Data Visualization with Python and JavaScript:
Scrape, Clean, Explore, and Transform Your Data 2nd Edition

© 2026 "Astana International Publishing" LLP Authorized Russian translation of the English edition of Data Visualization with Python and JavaScript, 2E ISBN 9781098111878
© 2023 Kyran Dale Limited This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

Дейл, Киран.

Д27 Визуализация данных с помощью Python и JavaScript. Анализ и преобразование данных / Киран Дейл : [перевод с английского Ю. Смирновой]. — Алматы : Астана иностранная пресса, 2026. — 624 с. — (O'Reilly. Книги по программированию).

ISBN 978-601-12-4680-4

Хотите научиться эффективно представлять данные? Эта книга покажет полный путь преобразования сырых данных в яркие и информативные визуализации. Вы освоите инструменты Python и JavaScript, используя популярные и доступные библиотеки. Киран Дейл делится проверенными методами сбора, очистки и анализа данных, демонстрируя создание динамических веб-интерфейсов. Вы сможете уверенно создавать привлекательные и понятные представления данных как локально, так и прямо в браузере.

Будет полезно для всех, кто хочет прокачать навыки обработки и отображения данных в современных веб-приложениях.

УДК 004.4
ББК 32.973.26-018.2

© Смирнова Ю. Н., перевод на русский язык, 2026
© Издание на русском языке, оформление.
ТОО «Издательство «Астана иностранная пресса», 2026

ISBN 978-601-12-4680-4

Оглавление

Предисловие 10
Второе издание 14
Принятые в книге обозначения 15
Использование примеров кода 16
Благодарности 17

Введение 18
Для кого эта книга? 19
Почему именно Python и JavaScript? 22
Чему вы научитесь 25
Предварительные сведения 26
Тулчейн для визуализации данных 27
Как пользоваться этой книгой 30
Немного контекста 31
Резюме 33
Рекомендуемые книги 34

Раздел I. Базовый пакет инструментов

Глава 1. Подготовка окружения 36
Сопутствующий код 36
Python 36
Установка дополнительных библиотек 37
JavaScript 39
Базы данных 41
Интегрированная среда разработки 43
Резюме 44

Глава 2. Обучающий мостик между Python и JavaScript 45
Сходство и различия 45
Взаимодействие с кодом 46
Строим мост 49
Примеры различий 77
Шпаргалка 90
Резюме 92

Глава 3. Чтение и запись данных с помощью Python 94
Просто ли это? 94
Передача данных 95
Работа с системными файлами 96
CSV, TSV и табличные форматы данных 97
JSON 100
SQL 105

MongoDB ... 116
Работа с датами, временем и сложными типами данных ... 122
Резюме ... 123

Глава 4. Основы веб-разработки ... 125
Общая картина ... 125
Одностраничные приложения ... 126
Настройка инструментов ... 126
Создание веб-страницы ... 130
Chrome DevTools ... 140
Базовая страница с плейсхолдерами ... 142
Позиционирование и изменение размера контейнеров с помощью Flex ... 146
Масштабируемая векторная графика ... 155
Резюме ... 169

Раздел II. Получение данных

Глава 5. Получение данных из интернета с помощью Python ... 173
Получение данных из интернета с помощью библиотеки Requests ... 173
Получение файлов данных с помощью Requests ... 174
Использование Python для получения данных через web API ... 177
Доступ к web API с помощью библиотек ... 183
Скрейпинг данных ... 189
Получение объекта BeautifulSoup ... 191
Выбор тегов ... 192
Резюме ... 202

Глава 6. Эффективный скрейпинг с помощью Scrapy ... 203
Установка Scrapy ... 204
Постановка целей ... 205
Работа с XPath в Scrapy ... 207
Первый паук Scrapy ... 214
Скрейпинг биографических страниц лауреатов ... 221
Цепочка запросов и извлечение данных ... 224
Конвейеры Scrapy ... 229
Скрейпинг текста и изображений с помощью конвейера ... 231
Резюме ... 239

Раздел III. Очистка и исследование данных с помощью pandas

Глава 7. Введение в NumPy ... 243
Массив NumPy ... 244
Создание функций для работы с массивами ... 251
Резюме ... 253

Глава 8. Знакомство с библиотекой pandas ... 254
Почему pandas оптимальна для визуализации данных ... 254
Зачем разработали pandas ... 254
Классификация данных и измерения ... 255
DataFrame ... 256

Создание и сохранение структур DataFrame ... 261
Создание DataFrame из Series ... 272
Резюме ... 275

Глава 9. Очистка данных с помощью pandas ... 276
Чистая правда о грязных данных ... 276
Проверка качества данных ... 278
Индексы и отбор данных с помощью pandas ... 282
Очистка данных ... 286
Полная функция для очистки данных ... 306
Добавление столбца born_in ... 307
Сохранение очищенных наборов данных ... 312
Резюме ... 313

Глава 10. Визуализация данных с помощью Matplotlib ... 314
Pyplot и объектно-ориентированная библиотека Matplotlib ... 314
Запуск интерактивной сессии ... 315
Создание интерактивных графиков с помощью глобального состояния pyplot ... 316
Фигуры и объектно-ориентированная Matplotlib ... 322
Типы графиков ... 327
Seaborn ... 336
Резюме ... 345

Глава 11. Анализ данных с помощью pandas ... 347
Начало исследования ... 348
Построение графиков с помощью pandas ... 350
Гендерные диспропорции ... 352
Национальные тренды ... 360
Возраст и ожидаемая продолжительность жизни лауреатов ... 373
Нобелевская «диаспора» ... 380
Резюме ... 382

Раздел IV. Передача данных

Глава 12. Передача данных ... 385
Передача данных ... 386
Доставка файлов данных ... 391
Динамическое обновление данных с помощью Flask API ... 396
Использование динамической или статической доставки ... 398
Резюме ... 399

Глава 13. RESTful Data с помощью Flask ... 400
Инструменты для работы с RESTful ... 400
Создание базы данных ... 401
Flask RESTful для работы с данными ... 402
Добавление маршрутов RESTful API ... 405
Расширение API с помощью MethodView ... 411
Пагинация возвращаемых данных ... 414
Удаленное развертывание API на Heroku ... 418
Резюме ... 422

Раздел V. Визуализация данных с помощью D3 и Plotly

Глава 14. Перенос диаграмм в интернет с помощью Matplotlib и Plotly 425
Создание статических диаграмм с помощью Matplotlib 425
Построение диаграмм с помощью Plotly 430
Из Notebook в веб-формат с помощью Plotly 444
Создание нативных JavaScript-диаграмм с помощью Plotly 448
Интерактивная визуализация Plotly с помощью JavaScript и HTML 454
Резюме 459

Глава 15. Разработка концепции визуализации Нобелевской премии 460
Для кого эта визуализация? 460
Выбор визуальных элементов 461
Строка меню 462
Распределение премии по годам 463
Карта, показывающая выборку стран нобелевских лауреатов 464
Столбчатая диаграмма, показывающая количество лауреатов по странам 465
Список выбранных лауреатов 466
Визуализация целиком 468
Резюме 469

Глава 16. Создание визуализации 470
Предварительные сведения 471
HTML-каркас 473
Стили CSS 477
Движок JavaScript 481
Запуск приложения для визуализации данных о нобелевских лауреатах 496
Резюме 497

Глава 17. Введение в D3 на примере столбчатой диаграммы 498
Формулирование задачи 499
Работа с выборкой 499
Добавление элементов DOM 503
Использование D3 510
Шкалы в D3: от данных к их визуальному представлению 510
Привязка данных к элементам DOM — главное преимущество D3 516
Обновление DOM при изменении данных 516
Сборка столбчатой диаграммы 520
Оси и метки 523
Переходы 529
Резюме 534

Глава 18. Визуализация отдельных премий 535
Создание структуры 535
Шкалы 536
Оси 537
Метки номинаций 538
Вложенные данные 540
Добавление лауреатов с помощью вложенных объединений данных 543
Добавим немного блеска! 547
Резюме 549

Глава 19. Картографирование с помощью D3 550
Доступные карты 550
Форматы данных для картографирования в D3 551
Библиотека D3-geo, проекции и пути 556
Соединение элементов воедино 562
Обновление карты 565
Добавление индикаторов показателей 569
Готовая карта 572
Создание простой всплывающей подсказки 573
Резюме 578

Глава 20. Визуализация данных отдельных лауреатов 579
Создание списка лауреатов 580
Создание биографического блока 584
Резюме 587

Глава 21. Строка меню 589
Создание HTML-элементов с помощью D3 590
Создание строки меню 590
Резюме 601

Глава 22. Заключение 602
Подведение итогов 602
Дальнейшее развитие 605
Заключительные замечания 607

Приложение А. Паттерн enter/exit библиотеки D3 608
Метод enter 609
Доступ к привязанным данным 613

Об авторе 616

Послесловие 617

Алфавитный указатель 618

Предисловие

Главная цель этой книги — представить тулчейн (англ. toolchain, «цепочка инструментов») для визуализации данных (далее также — визуализация), который дает большие преимущества в эпоху интернета. Цель создания этого тулчейна — извлекать из полученных данных каждую крупицу ценной информации и передавать в браузер. После передачи вы можете делиться своими визуализациями со всем миром или с ограниченным кругом лиц (например: внутри локальной сети или с использованием аутентификации). Интернет открывает огромные возможности для визуализации, и будущее этой области тесно связано с JavaScript, лучшим языком для веб-разработки. Однако JavaScript не располагает средствами для предварительной обработки сырых данных, поэтому к процессу визуализации требуется привлекать другие языки программирования. Я надеюсь, что прочитав эту книгу, вы согласитесь со мной: Python — наиболее подходящий язык для совместной работы с JavaScript для визуализации данных в браузере.

Хотя книга получилась довольно объемной (что автор чувствует особенно остро), в ней не удалось охватить все замечательные инструменты Python и JavaScript для визуализации. Пришлось сосредоточиться на тех инструментах, которые формируют основу наиболее эффективных решений. Большое число полезных библиотек, оставшихся за рамками книги, подчеркивает жизнеспособность экосистемы Data Science на базе Python и JavaScript. Пока писалась книга, появились новые отличные библиотеки на обоих языках, и этот процесс продолжается.

Любая визуализация данных подразумевает их трансформацию. Чтобы продемонстрировать основные инструменты визуализации, рассмотрим превращение одного способа отображения набора данных (с помощью списков и HTML-таблиц) в другой: более современный, интерактивный и наглядный, основанный на браузере. Наша задача — преобразовать базовый список лауреатов Нобелевской премии, взятый из Википедии, в современную интерактивную визуализацию в браузере. Таким образом, тот же самый набор данных будет представлен в более доступной и привлекательной форме.

Чтобы создать на основе сырых данных интерактивную визуализацию с широкими возможностями, нам понадобятся лучшие в своем классе инструменты. Для начала необходимо получить набор данных. Иногда мы получаем его

от коллег или друзей, но, чтобы немного усложнить задачу и отработать важные навыки, научимся использовать *скрейпинг* наборов данных из интернета, в данном случае со страниц Википедии о Нобелевской премии. Для этого воспользуемся мощной Python-библиотекой Scrapy. Затем полученный набор сырых данных потребуется очистить и проанализировать, и для этого нет равных библиотеке pandas из экосистемы Python. Pandas в связке с Matplotlib и Jupyter Notebook — золотой стандарт для такого рода аналитики. Из очищенных и проанализированных данных, сохраненных в SQL-формате с помощью SQLAlchemy или SQLite, выберем интересные для визуализации аспекты. Я расскажу, как использовать Matplotlib и Plotly для встраивания статических и динамических диаграмм из Python в веб-страницы. Однако лучшей библиотекой для масштабной веб-визуализации остается D3 на основе JavaScript. Мы познакомимся с основами D3, создавая визуализацию данных о лауреатах Нобелевской премии.

В книге представлен набор инструментов, формирующий цепочку, а связующей нитью выступает визуализация данных о лауреатах Нобелевской премии. Структура книги позволяет легко находить главы по интересующему вас вопросу. Каждый раздел является самостоятельным, что помогает быстро отыскать и вспомнить пройденный материал.

Книга содержит пять разделов. Первый является введением в базовый набор инструментов Python и JavaScript для визуализации. В остальных четырех показано, как собирать и очищать сырые данные, анализировать их и превращать в современную веб-визуализацию. Давайте кратко сформулируем, какие основные уроки можно извлечь из каждого раздела.

Раздел I. Базовый пакет инструментов

О чем этот раздел:

- Обучающий мостик между Python и JavaScript, создан, чтобы сгладить переход между языками, подчеркнуть их сходные элементы и подготовить окружение для создания современной визуализации с помощью обоих языков. В последней версии JavaScript[1] появилось еще больше общего с Python, поэтому переключаться с одного на другой стало проще.
- Одна из сильных сторон Python — чтение/запись основных форматов обмена данными (например, JSON и CSV), а также поддержка баз данных (как SQL, так и NoSQL). Python легко передает данные, преобразуя их

[1] Есть много версий JavaScript на основе спецификации ECMAScript, но больше всего новых функциональных возможностей у ES6.

из одного формата в другой и меняя базы данных по мере необходимости. Такая гибкость в управлении данными — ключевой элемент, обеспечивающий плавную работу любого тулчейна визуализации.

— Мы также рассмотрим базовые навыки веб-разработки, которые необходимы для создания современной интерактивной визуализации в браузере. Чтобы минимизировать рутинное веб-программирование и сосредоточиться на разработке ваших визуальных проектов, мы не станем делать сложный сайт, ограничимся одностраничным веб-приложением (Single-Page Aapplication, SPA) на JavaScript. Введение в SVG (Scalable Vector Graphics, «язык разметки векторной графики»), на котором в основном строятся D3-визуализации, — подготовка к созданию визуализации данных по Нобелевской премии в разделе IV.

Раздел II. Подготовка данных

В этой части книги мы рассмотрим, как самому получить данные из интернета с помощью Python, если вам не предоставили готовый файл с чистыми данными:

— Если вам повезло, и в открытом доступе есть такой файл в подходящем формате, например JSON или CSV, то достаточно отправить простой HTTP-запрос. Кроме того, для вашего набора данных может отыскаться специальный web API, хорошо, если это будет RESTful API. В качестве примера мы рассмотрим применение Python-библиотеки Tweepy для доступа к Twitter API. Мы также увидим, как использовать Google Таблицы (Google Spreadsheets), популярный инструмент для обмена данными в визуализации.

— Если же данные представлены в интернете в формате, ориентированном на людей, например: в виде HTML-таблицы, списков или структурированного контента, то задача усложняется. Тогда для извлечения сырого HTML-контента придется использовать *скрейпинг*, а затем с помощью парсера извлечь из полученных данных нужную информацию. Мы рассмотрим, как использовать для скрейпинга легковесную библиотеку Beautiful Soup и куда более многофункциональную и тяжеловесную Scrapy, самую крупную звезду веб-скрейпинга на небосклоне Python.

Раздел III. Очистка и исследование данных с помощью pandas

В этом разделе для очистки и исследования наборов данных мы задействуем «тяжелую артиллерию» — Python-библиотеку pandas. Сначала мы рассмотрим

pandas как часть экосистемы NumPy, которая предоставляет доступ к мощным низкоуровневым библиотекам для быстрой обработки массивов данных. Особое внимание уделим использованию pandas для очистки и анализа набора данных о лауреатах Нобелевской премии:

— Даже те данные, которые получены через официальные web API, в основном грязные. Чтобы очистить их и подготовить для визуализации, потребуется гораздо больше времени, чем вы, вероятно, ожидаете. Мы возьмем наш тренировочный набор данных о лауреатах Нобелевской премии и постепенно очистим его. Найдем и удалим неточные даты, аномальные типы данных, пропуски и прочую «грязь», прежде чем приступить к исследованию данных и их последующей визуализации.
— Очистив (насколько сумеем) набор данных о Нобелевской премии, мы увидим, как просто с помощью pandas и Matplotlib интерактивно исследовать данные, создавать диаграммы со всевозможными срезами данных, а также получить общее представление о них и отыскать ценную информацию, которую вы хотите донести до пользователя с помощью визуализации.

Раздел IV. Доставка данных

Здесь мы разберемся, как с помощью Flask создать минимальный API, чтобы передавать в браузер как статический, так и динамический контент.

Сначала посмотрим, как использовать Flask для работы со статическими файлами, а затем, как запустить собственный базовый API для данных из локальной БД. Минимализм Flask позволяет создать очень тонкий сервисный слой между результатами обработки данных с помощью Python и их конечной визуализацией в браузере.

Прелесть открытого ПО в том, что всегда можно найти надежную и простую в использовании библиотеку, которая решит вашу задачу лучше, чем если бы вы делали все вручную. Во второй главе раздела рассмотрим, насколько просто использовать лучшие в своем классе Python-библиотеки (на примере Flask) при создании надежного и гибкого RESTful API для обслуживания данных онлайн. Мы также рассмотрим простое развертывание сервера данных на облачной платформе Heroku, популярной среди Python-разработчиков.

Раздел V. Визуализация данных с помощью D3 и Plotly

В первой главе мы рассмотрим, как из данных, отобранных после анализа с помощью pandas, создать диаграммы или карты и опубликовать их в интернете. Статические диаграммы полиграфического качества мы сделаем с помощью Matplotlib, а интерактивные элементы и динамические диаграммы — с помощью Plotly. Мы рассмотрим, как вызвать формирование диаграммы Plotly из Jupyter Notebook и отобразить полученную диаграмму на веб-странице.

Часть, посвященная D3 — одна из самых сложных в книге, но D3 незаменима, если нужно создавать многоэлементные визуализации. Одним из плюсов библиотеки D3 является возможность найти в интернете множество примеров ее применения, хотя большинство из них демонстрируют только какую-то одну технику. Примеров, которые показывают, как организовать взаимодействие нескольких визуальных элементов, очень мало. В главах о D3 мы разберем, как синхронизировать обновление временной диаграммы (отображающей все вручения Нобелевской премии), карты, столбчатой диаграммы и списка лауреатов, когда пользователь применяет фильтры или меняет показатель присуждения премии (абсолютный или на душу населения).

Эти главы позволят вам дать волю воображению и учиться на практике. Я бы порекомендовал выбрать интересные для вас данные и на их основе разработать что-нибудь с помощью D3.

Второе издание

Я с некоторым сомнением воспринял предложение издательства O'Reilly поработать над вторым изданием этой книги. Первое издание получилось объемнее, чем ожидалось, и на его доработку могло потребоваться немало труда. Однако, когда я проверил актуальность описанных в книге библиотек и изменений, касающихся визуализации, в экосистемах Python и JavaScript, выяснилось, что большинство библиотек (например: Scrapy, NumPy, pandas) остаются отличными вариантами, и существенных изменений текста не требуется.

Больше всего изменилась библиотека D3, но при этом ее стало проще использовать и легче изучать. Модульность стала стандартом разработки на JavaScript, что сделало JS-код чище и привычнее для питонистов.

Выбор нескольких Python-библиотек теперь выглядит менее удачным, а пара из них попросту устарела. В первом издании довольно подробно рассматривалась MongoDB — база данных NoSQL, но теперь я считаю, что старый добрый SQL лучше подходит для работы с визуализацией, а легкая

однофайловая бессерверная SQLite — идеальное решение, если для визуализации требуется БД.

Вместо замены устаревшего RESTful-сервера на другою Python-библиотеку, я решил показать, как создать простой сервер с нуля, используя такие замечательные библиотеки на Python, как marshmallow, которые полезны во многих сценариях визуализации.

С учетом времени, отведенного на обновление книги, я решил показать исследования и анализ с помощью Matplotlib и pandas на примере набора данных из первого издания, сосредоточившись на обновлении всех библиотек до версий, актуальных на середину 2022 года. Это позволило мне выделить время на изложение нового материала и, самое главное, — написать главу о Plotly. Эта библиотека на Python упрощает перенос наработок из Jupyter Notebook в интерактивное веб-представление. Особое преимущество этого подхода — доступ к картам богатой картографической экосистемы Mapbox.

Основной упор во втором издании я делал на следующем:

— обновить все библиотеки;
— удалить и/или заменить библиотеки, которые не выдержали испытания временем;
— добавить новый материал, связанный с изменениями в быстроразвивающемся мире визуализации с помощью Python и JavaScript.

Я считаю, что концепция тулчейна для визуализации осталась в силе, и конвейер преобразований — от сырых, необработанных веб-данных через исследовательский анализ до безупречной веб-визуализации — остается прекрасным способом изучения ключевых инструментов.

Принятые в книге обозначения

Типографские соглашения:

Курсивом
: выделяются новые термины, URL-адреса, адреса электронной почты, имена и расширения файлов.

`Моноширинным шрифтом`
: выделен код программы, а также встречающиеся в тексте программные элементы: переменные, имена функций и баз данных, типы данных, ключевые слова, операторы и переменные окружения.

Полужирным моноширинным шрифтом
: выделены команды и другой текст, который пользователь должен ввести без изменений.

Моноширинным курсивом
: выделен текст, который надо заменить на заданные пользователем значения, или на значения, определяемые контекстом.



Этот элемент означает подсказку или предложение.



Этот элемент означает общее примечание.



Этот элемент означает предупреждение или предостережение.

Использование примеров кода

Дополнительные материалы (примеры кода, упражнения и т. д.) доступны для скачивания по адресу https://github.com/Kyrand/dataviz-with-python-and-js-ed-2.

Предназначение этой книги — помочь вам решить свои задачи. Как правило, вы можете использовать примеры кода из этой книги в своих программах и документации. Вам не нужно обращаться к нам за разрешением, кроме тех случаев, когда вы собираетесь воспроизвести значительную часть кода. Например, вам не требуется разрешение, если вы пишете программу, в которой используются несколько фрагментов кода из данной книги. Однако для продажи или распространения компакт-диска с примерами из книг O'Reilly разрешение требуется. Цитировать эту книгу с примерами из кода вы можете свободно, но если вы собираетесь включить значительное число примеров кода из книги в документацию по своему продукту, вам следует обратиться к нам за разрешением.

Благодарности

Прежде всего хочу поблагодарить Меган Бланшет (Meghan Blanchette), которая положила начало этой книге и помогла мне с самыми трудными главами. Затем Дон Шанафелт (Dawn Schanafelt) взяла бразды правления в свои руки и проделала большую часть необходимой редакторской работы. Кристен Браун (Kristen Brown) блестяще подготовила книгу к печати, в чем ей помогла стальная хватка литературного редактора Джилиана МакГарви (Gillian McGarvey). Работа с этими талантливыми, преданными делу профессионалами была для меня не только честью и привилегией, но и обучением: если бы я с самого начала знал все, чему научился от них, мне было бы гораздо легче писать книгу. Но ведь так всегда и бывает?

Огромное спасибо Эми Зелински (Amy Zielinski), благодаря которой автор выглядел лучше, чем он того заслуживает.

Книга существенно улучшилась благодаря ряду ценных замечаний Кристофа Вио (Christophe Viau), Тома Парслоу (Tom Parslow), Питера Кука (Peter Cook), Иэна Макиннеса (Ian Macinnes) и Иэна Освальда (Ian Ozsvald).

Я также хотел бы поблагодарить отважных охотников за ошибками, допущенными в черновике книги. Это Дуглас Келли (Douglas Kelley), Павел Сук (Pavel Suk), Брайам Хаусман (Brigham Hausman), Марко Хемкен (Marco Hemken), Нобль Кеннамер (Noble Kennamer), Манфреди Бьясутти (Manfredi Biasutti), Мэтью Мальдонадо (Matthew Maldonado) и Хирт Боувенс (Geert Bauwens).

Второе издание

Прежде всего, я должен поблагодарить Ширу Эванс (Shira Evans) за сопровождение книги от замысла до реализации, и Грегори Хаймана (Gregory Hyman), который держал меня в курсе дел по черновикам и предоставлял обратную связь. Мне вновь посчастливилось работать с Кристен Браун (Kristen Brown), именно она подготовила к печати второе издание книги.

Также благодарю технических рецензентов Джордана Голдмайера (Jordan Goldmeier), Дрю Уинстела (Drew Winstel) и Джесс Мэйлс (Jess Males) за отличные советы.

Введение

Цель этой книги — познакомить вас с самым мощным, по моему мнению, стеком для визуализации данных: Python и JavaScript. Изучив такие крупные библиотеки, как pandas и D3, вы можете сразу приступить к созданию веб-визуализации данных и сформировать собственный тулчейн. Книга поможет вам получить базовые навыки, а мастерство будет возрастать с дальнейшей практикой.



Я рад буду получить от вас отзыв о книге. Пишите по адресу pyjsdataviz@kyrandale.com. Заранее благодарю.
Также на моем веб-сайте (htpps://www.kyrandale.com/viz/nobel_viz_v2/) вы найдете рабочую копию визуализации данных о Нобелевской премии (Visualising the Nobel Prize), на основе которой, в прямом и переносном смысле, выстроена эта книга.

Центральная тема книги — история одной визуализации данных, которую я тщательно отбирал из многих, чтобы продемонстрировать потенциал мощных библиотек и инструментов — тулчейна для Python и JavaScript. С помощью этого тулчейна мы начнем со сбора сырых, необработанных данных и в конце превратим их в интересную веб-визуализацию. Любая визуализация данных — это процесс их трансформации. В данном случае мы превратим список лауреатов Нобелевской премии в интерактивную визуализацию, чтобы знакомство с историей Премии стало простым и увлекательным.

Визуализация любых данных ориентирована на интернет, возможности которого по доставке данных многократно превосходят все, что было раньше. Эта книга призвана сгладить переход от обработки и анализа данных на ПК и локальных серверах к интернету.

Python и JavaScript — мощные языки для визуализации данных, к тому же писать на них программы интересно и увлекательно.

Многие программисты полагают, что между *веб-разработкой* и программированием на Python и JavaScript лежит пропасть. По их мнению, для веб-разработки придется постигать мудреную науку о языках разметки, администрировании и скриптах для описания стиля и обязательно применять инструменты со странными названиями *Webpack* или *Gulp*. В наши дни можно с минимальными

SKLADCHIK.ORG
усилиями преодолеть эту пропасть и сосредоточиться на хорошо знакомом программировании (см. рисунок I-1), делегировав доставку данных веб-серверам.

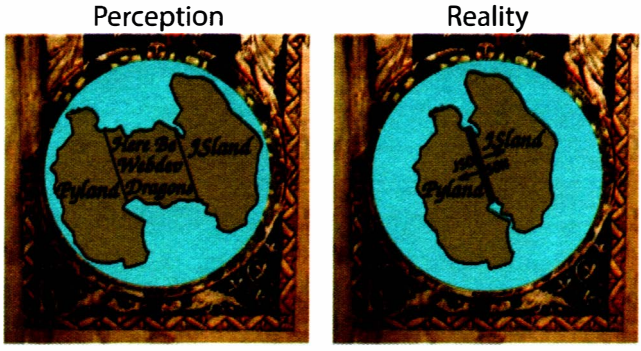


Рис. I-1. *Here be webdev dragons («Тут живут драконы веб-разработки»)*

Для кого эта книга?

В первую очередь она предназначена для тех, кто хорошо знает Python или JavaScript (JS) и хочет изучить одну из самых интересных и перспективных областей в сфере обработки данных: визуализацию данных в браузере. В книге также можно найти решение некоторых конкретных и, по-моему, довольно распространенных проблем.

Если вам предстоит написать техническую книгу, стоит подумать о *«болевых точках»*, которые она сможет устранить. Две ключевые «болевые точки» этой книги проиллюстрированы парой историй, одна — из моей практики, а другую в разных вариантах мне рассказали знакомые JavaScript-программисты.

Я влюбился в Python давно, еще когда был ученым-исследователем. В тот момент я писал несколько сложных имитационных моделей на C++, и Python показался мне глотком чистого воздуха, без boilerplate-кода типа Makefile, деклараций и объявлений. Программировать стало увлекательно. Python оказался идеальным связующим звеном, прекрасно сочетаясь с библиотеками на C++ (сам Python не может похвастаться быстротой). Он с непревзойденной легкостью выполнял все, что так сложно для языков низкого уровня, например: ввод/вывод файлов, доступ к БД и сериализацию. Я начал писать все свои графические интерфейсы (GUI) и визуализации на Python, используя wxPython, PyQt и массу других удивительно простых инструментов. К сожалению, некоторые из этих инструментов плохо совместимы с современными библиотеками.

В то время уже существовали браузеры, которые, казалось, идеально подходили для распространения ПО, которое я так увлеченно создавал. Браузеры, со встроенным интерпретируемым языком программирования позволяли создавать приложения, работающие на любом компьютере на Земле — напиши один раз и запускай везде. Но у Python было мало точек соприкосновения с браузерами, а их возможности ограничивались статическими изображениями и не очень понятными преобразованиями JS-объектов с помощью jQuery. JavaScript был упрощенным языком, привязанным к очень медленному интерпретатору, и годился только для небольших манипуляций с DOM, но даже не приближался к тому, что я мог сделать на Python. Так что этот путь я отверг. Я хотел поместить свои визуализации в браузер, но не имел возможности.

Примерно через 10 лет JavaScript благодаря движку Google V8 Engine стал в десятки раз быстрее и по быстродействию намного обогнал Python[1]. С HTML 5 тоже стало намного приятнее работать, чем с предыдущими версиями HTML, поскольку уменьшилось количество нетворческого boilerplate-кода. Такая графика, как SVG (Scalable Vector Graphics), стала надежнее благодаря мощным библиотекам визуализации, в частности D3. Все современные браузеры могут хорошо работать с SVG, а многие и с 3D-графикой благодаря WebGL и таким библиотекам, как THREE.js. Визуализации, созданные на Python, теперь можно просматривать в локальном браузере. С минимальными усилиями их можно сделать доступными для каждого устройства: ПК, ноутбука, смартфона и планшета в любой точке мира.

Так почему же питонисты не хотят добывать данные этим путем? Отмечу, что большинство моих знакомых специалистов по работе с данными предпочитают самостоятельно вести разработку, иначе результат их не устроит. Тут есть две проблемы. Во-первых, для *веб-разработки* надо освоить сложную гипертекстовую разметку, непрозрачные таблицы стилей, а также новые инструменты и IDE. Во-вторых, надо освоить незнакомый язык JavaScript, который до недавнего времени считался не подходящим для серьезных проектов: так, ни рыба, ни мясо. Моя цель — показать, что можно создавать современные веб-визуализации (обычно одностраничные веб-приложения) с минимумом HTML и CSS boilerplate-кода, что делает процесс программирования творческим и облегчает переход питонистов к JavaScript. Не пропускайте главу 2 — мостик, который призван помочь специалистам по Python и JS преодолеть разрыв между языками, акцентируя внимание на их сходных элементах и предлагая простые способы перевода с языка с на язык.

[1] Для сравнения по скорости см. сайт Benchmarks Game (https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/python.html).

Другая сторона истории про визуализацию данных — это взгляд со стороны JavaScript-разработчиков. Обработка данных с помощью JavaScript далека от идеала. Хотя есть несколько мощных библиотек, и благодаря последним улучшениям языка с данными стало приятнее работать, настоящая экосистема обработки данных пока не появилась. Таким образом, в JS есть явная асимметрия между чрезвычайно мощными библиотеками визуализации (D3, несомненно, является важнейшей) и низкой способностью очищать и обрабатывать любые данные для передачи в браузер. Поэтому для очистки, обработки и изучения данных приходится использовать или другой язык, или инструментарий, такой как Tableau. Часто все сводится к разрозненному применению смутно знакомого Matlab, довольно сложного для изучения языка R, или одной-двух библиотек на Java.

Инструменты вроде Tableau выглядят внушительно, но часто в конечном итоге разочаровывают разработчиков, поскольку в своем графическом интерфейсе не могут повторить того, на что способен хороший язык программирования общего назначения. Кроме того возникает вопрос: что делать, если нужно создать небольшой веб-сервер для доставки обработанных данных? Придется изучать хотя бы один новый язык, подходящий для веб-разработки.

В результате разработчики JavaScripts, которые хотят улучшить свои визуализации данных, ищут дополнительный стек, который можно освоить быстрее и проще.

Как получить максимальную пользу от книги

Я всегда неохотно налагаю ограничения, особенно если это касается программирования и интернета, где масса самоучек, не скованных формальностями, учатся стремительно и энергично. А как еще можно учиться, когда академические программы отстают от текущих реалий на несколько световых лет? Как Python, так и JavaScript — идеальные кандидаты на роль первого языка программирования. Они оба максимально просты для изучения. Для понимания их кода не требуется чрезмерных умственных усилий.

Опытные программисты, даже без опыта работы с Python и JavaScript, смогут с помощью этой книги уже через неделю писать кастомные библиотеки.

Для начинающих программистов, незнакомых с Python или JavaScript, эта книга может показаться сложной. В таком случае я рекомендую начать с учебных пособий, веб-ресурсов, скринкастов и т. п., чтобы освоить азы. Сосредоточьтесь на конкретной задаче, которую хотите решить. Учитесь на практике — это единственный способ стать программистом.

Входной порог для тех, кто немного программировал на Python или JavaScript — вы использовали пару библиотек на том и другом языке, усвоили основные идиомы «своего» языка и, увидев фрагменты кода на другом, в целом понимаете, что этот код делает. Иными словами, книга будет полезна как питонистам, которые могут использовать пару модулей из стандартной JS-библиотеки, так и JavaScript-разработчикам, которые использовали библиотеку на Python и могут разобраться в ее коде.

Почему именно Python и JavaScript?

С JavaScript все просто. Сейчас (и в обозримом будущем) есть только один первоклассный язык программирования для браузера. Несмотря на различные попытки потеснить его, старый добрый простой JS — вне конкуренции. Если ваша цель — создавать и мгновенно делиться с миром современными динамичными интерактивными визуализациями, вам необходим JavaScript. Даже если вы не планируете стать гуру, базовые навыки программирования на JS распахнут перед вами двери в одну из самых захватывающих областей современной Data Science. С помощью этой книги вы их получите.

Почему Python не подходит для браузера?

Недавно появились попытки запуска ограниченной версии Python в браузере. Например, Pyodide — это переход от CPython к WebAssembly. Эти попытки впечатляют, но на данный момент основным способом создания веб-диаграмм на Python остается их автоматическое преобразование через промежуточную библиотеку.

В нескольких впечатляющих проектах разработчики направили усилия на то, чтобы в браузере можно было запускать визуализации, обработанные на Python, и часто собранные с помощью Matplotlib. Для этого код Python конвертируется в JavaScript для создания растровой (`canvas`) или векторной (`svg`) графики. Самые популярные и зрелые библиотеки визуализации — Plotly и Bokeh. В главе 14 вы узнаете, как с помощью Plotly генерировать в Jupyter Notebook диаграммы и передавать их на веб-страницу. Это прекрасный инструмент визуализации, который стоит включить в свой инструментарий.

Однако у этих замечательных конвертеров в формат JavaScript имеется ряд ограничений:

— Автоматическое конвертирование работает неплохо, но обычно результирующий код для человека практически нечитаем.

— Адаптация и настройка итоговой графики с помощью мощной браузерной среды разработки JavaScript может оказаться непростой задачей. В главе 14 мы рассмотрим, как облегчить эту задачу, используя Plotly JS API.
— В этих библиотеках ограниченный набор типов диаграмм, а интерактивность доступна лишь на базовом уровне. Настройку пользовательских элементов управления лучше всего делать на JavaScript, используя инструменты разработчика браузера.

Учтите, что эти библиотеки создают эксперты по JavaScript. Чтобы понять, что они делают, и воспользоваться ими, придется освоить JavaScript на достаточном уровне.

Почему для обработки данных выбран Python?

На этот вопрос ответить сложнее. В области обработки данных для Python имеются достойные альтернативные варианты. Рассмотрим некоторых конкурентов, начиная с гиганта корпоративного класса — Java.

Java

Среди основных языков программирования общего назначения богатую экосистему библиотек, сравнимую с экосистемой Python, предлагает только Java. К тому же, он работает существенно быстрее. На нем проще программировать, чем на C++, но все же, на мой взгляд, Java излишне многословный язык, перегруженный boilerplate-кодом. Со временем подобные вещи начинают угнетать и превращают программирование в тяжкий труд. Конечно, интерпретатор Python по умолчанию работает медленнее, но Python отлично связывает компоненты, написанные на разных языках. Эту способность демонстрируют крупные Python-библиотеки для обработки данных: NumPy (а также pandas из ее экосистемы), SciPy и им подобные, которые для выполнения тяжелой работы применяют библиотеки на C++ и Fortran, сохраняя при этом легкость использования простого скриптового языка.

R

До недавнего времени многие специалисты по работе с данными предпочитали язык R, который в этой области является основным конкурентом Python. Как и у Python, у R есть активное сообщество и прекрасные инструменты, например пакет для визуализации данных ggplot2. Его синтаксис специально разработан для статистики и Data Science. Но его специализация — палка о двух концах. Язык R был разработан для конкретной цели и не очень подходит для других.

Например, если вы хотите создать веб-сервер для данных, обработанных с помощью R, вам придется перейти на другой язык со всеми издержками на его изучение или попытаться как-то связать компоненты, которые подходят друг другу, как корове черкесское седло. Универсальность Python и его богатая экосистема позволяют выполнять практически все, что требуется для конвейера обработки данных (кроме графики JS), не выходя из зоны комфорта. Я считаю, что это достойная компенсация за некоторую синтаксическую неуклюжесть.

Прочие

Ни одна из оставшихся альтернатив и близко не обладает гибкостью и мощью универсального, простого в использовании языка программирования с богатой экосистемой библиотек. Например, среды математического программирования, такие как Matlab и Mathematica, имеют активные сообщества и множество отличных библиотек, но они не универсальные, поскольку предназначены для узкоспециализированных задач. К тому же они проприетарные, то есть требуют оплаты за использование и не позволяют изменять исходный код, в отличие от среды Python с открытым кодом.

Инструменты для визуализации с акцентом на графический интерфейс, такие как Tableau — прекрасны, но они быстро разочаруют того, кто ценит свободу в программировании. Они работают отлично, пока вы, так сказать, пляшете под их дудку. Любое отклонение от заданного пути приводит к проблемам.

Python постоянно улучшается

Я считаю, что Python уже сейчас является оптимальным языком для начинающих специалистов по работе с данными. И развитие продолжается: возможности Python в этой области растут невероятно быстро. Я программирую на Python уже более 20 лет и обычно нахожу модуль Python, подходящий для решения любой задачи, при этом меня поражает скорость появления новых библиотек для обработки данных — они выходят каждую неделю. Например, Python долго уступал R по библиотекам статистического анализа, но теперь разрыв сокращается благодаря таким мощным модулям, как statsmodels.

Еженедельно универсальная экосистема Python для обработки данных развивается и улучшается. Неудивительно, что многие члены сообщества переживают душевный подъем — такой темп развития вдохновляет.

Что касается визуализации в браузере, то JavaScript по праву занимает исключительное место в веб-экосистеме. Благодаря стремительному росту производительности интерпретатора и мощным библиотекам визуализации, таким как D3.js, которые украсили бы любой язык, JavaScript приобрела серьезные преимущества.

Итак, Python и JavaScript прекрасно дополняют друг друга при создании веб-визуализаций данных.

Чему вы научитесь

В наш тулчейн для визуализации входят несколько крупных библиотек на Python и JavaScript. Для их полного описания потребовалась бы целая серия книг. Однако я думаю, что основы работы большинства библиотек, в том числе рассматриваемых, можно усвоить довольно быстро. Для наработки опыта необходимы время и практика, но чтобы получить необходимые базовые знания, больших усилий не требуется.

Книга даст вам прочный фундамент практических навыков, которые впоследствии вы сможете развивать самостоятельно. Моя цель — максимально упростить процесс обучения, помочь вам преодолеть начальный барьер и овладеть практическими навыками, необходимыми для профессионального роста.

Эта книга призвана приносить практическую пользу и знакомить с лучшими практиками. Нам нужно охватить большой практический материал, поэтому для теоретических рассуждений места недостаточно. Я сосредоточусь на наиболее востребованных возможностях библиотек из тулчейна и поделюсь ресурсами для решения других задач. Большинство библиотек содержит основное ядро функций, методов, классов и прочего, которое составляет необходимый фундамент. Используя этот функционал, вы можете решить большинство задач. Но когда возникнет потребность в чем-то большем, вы найдете подсказки в других хороших книгах, документации и на онлайн-форумах.

Выбор библиотек

Я отбирал библиотеки по следующим трем критериям:

- открытый исходный код и бесплатность, чтобы вы избежали дополнительных расходов после покупки книги;
- популярность, долговечность и поддержка сообщества;
- первенство в своем классе (при наличии хорошей поддержки и активного сообщества), «золотая середина» между популярностью и полезностью.

Полученные навыки должны оставаться актуальными еще долгое время. В целом я выбрал библиотеки, которые говорят сами за себя. При необходимости я рассмотрю альтернативные варианты и обосную свой выбор.

Предварительные сведения

Несколько вводных глав предваряют главную тему книги — трансформация набора данных о Нобелевской премии с помощью тулчейна. Эти главы дадут вам базовые сведения, которые облегчат освоение материала о тулчейне. Из первых глав вы узнаете о следующем:

Глава 2. Обучающий мостик между Python и JavaScript

Как построить мост между этими двумя языками программирования.

Глава 3. Чтение и запись данных с помощью Python

Работа с файлами и базами данных различных форматов в Python.

Глава 4. Основы веб-разработки

Знакомство с основами веб-программирования, необходимыми для этой книги.

Эти главы обучающие и справочные одновременно. Вы можете их пропустить и сразу перейти к тулчейну, возвращаясь к главам 2-4 мере необходимости.

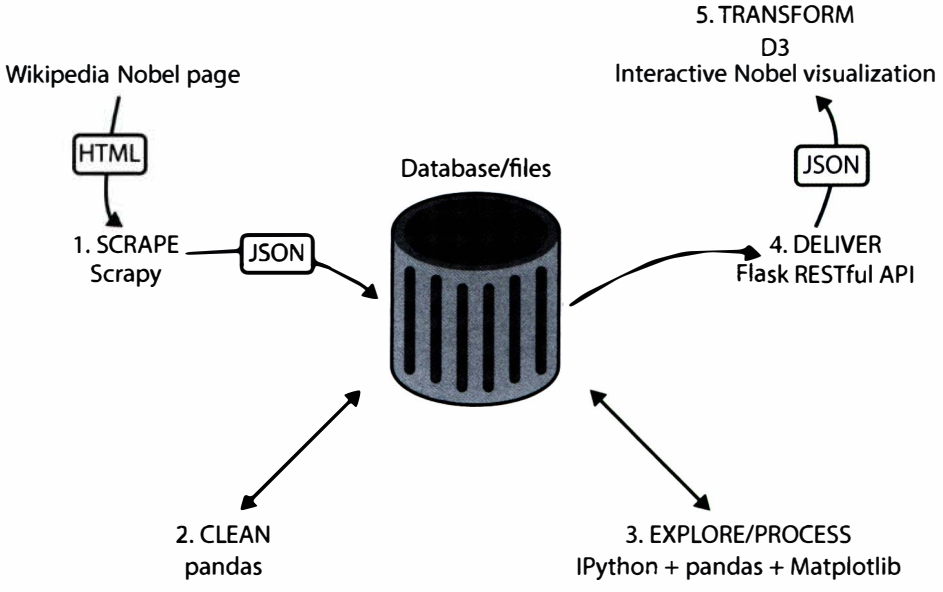


Рис. I-2. *Тулчейн для визуализации*

Тулчейн для визуализации данных

В основной части книги мы рассмотрим тулчейн визуализации данных, пройдя путь от скрейпинга сырых данных о лауреатах Нобелевской премии до создания вовлекающей во взаимодействие JavaScript-визуализации. В процессе сбора вы познакомитесь с очисткой и трансформацией данных с помощью ряда крупных библиотек (см. рисунок I-2). Эти библиотеки с богатыми возможностями — звенья нашего тулчейна, демонстрирующие могущество стека визуализации на основе Python и JavaScript. Затем кратко ознакомимся с пятью этапами применения нашего тулчейна и с соответствующими библиотеками.

1. Скрейпинг данных с помощью Scrapy

Первая задача в проекте по визуализации — собрать нужные данные. Иногда их удается получить готовыми, но чаще всего приходится добывать в интернете самостоятельно. Я расскажу, как использовать Python для получения данных из интернета различными способами (например, через web API или Google Таблицы). Нужный нам набор данных о Нобелевской премии я собрал со страниц Википедии с помощью Scrapy[1].

Scrapy — Python-фреймворк промышленного класса для скрейпинга, способный управлять потоками данных и обеспечивать конвейерную обработку медиафайлов, что необходимо при обработке значительных объемов данных. Скрейпинг — часто единственный способ получить интересующие вас данные. Освоив Scrapy, вы сможете собирать ранее недоступные наборы данных, просто запустив пауков[2].

2. Очистка данных с помощью pandas

Практически все собранные данные требуют очистки, и этот процесс может занять добрую половину проектного времени. Желание упростить и ускорить решение этой задачи — веская причина для выбора нужных инструментов.

Библиотека для обработки и анализа данных *pandas* играет большую роль в экосистеме Python. Основной структурой данных в этой библиотеке является DataFrame («датафрейм»), по сути, программируемая электронная таблица. Pandas расширяет возможности NumPy — мощной библиотеки для

[1] Веб-скрейпинг — это программная технология извлечения информации с веб-сайтов, обычно включающая в себя получение и парсинг веб-страниц.

[2] Пауками (spiders) называют контроллеры Scrapy.

математических вычислений — в области наборов гетерогенных данных, таких как категориальные, темпоральные и порядковые.

Библиотека pandas хорошо справляется с трудоемкой работой по очистке данных: позволяет легко выявлять дубликаты, исправлять сомнительные даты, находить пропуски и так далее. Кроме того, она отлично подходит для интерактивного исследования данных в связке с библиотекой Matplotlib и функцией для построения графиков plot.

3. Исследование данных с помощью pandas и Matplotlib

Перед тем, как приступить к трансформации данных для визуализации, с ними надо разобраться. Закономерности, тренды и аномалии, скрытые в данных, послужат основой для вашей истории, будь то виджет анализа роста продаж за год или демонстрация глобальных изменений климата.

Matplotlib (с такими дополнениями, как seaborn) и pandas при использовании совместно с *IPython* (интерпретатором Python «на стероидах») предоставляют отличные возможности для интерактивного анализа данных, создания разнообразных графиков с помощью командной строки и выделения срезов данных для выявления интересных закономерностей. Результаты исследований затем можно сохранить в файле или БД, чтобы передать в визуализацию на JavaScript.

4. Доставка данных с помощью Flask

Данные после очистки и анализа нужно отправить в браузер, где их трансформирует JavaScript-библиотека, например D3. Одно из важнейших преимуществ Python как языка общего назначения — это возможность так же легко создать веб-сервер всего несколькими строками кода, как и работать с большими наборами данных с помощью специализированных библиотек, таких как NumPy и SciPy[1]. *Flask* — самый популярный легковесный фреймворк для Python, идеальный для создания небольших RESTful API[2], через которые данные из файлов или БД передаются с сервера в браузер для дальнейшей обработки средствами JavaScript. Я покажу, как всего несколькими строками кода реализовать REST API, способный доставлять данные из SQL или NoSQL баз данных.

[1] Библиотека на Python, предназначенная для научных расчетов, часть экосистемы NumPy.

[2] REST (Representational State Transfer) — хорошо себя зарекомендовавший и самый распространенный архитектурный стиль создания API на основе HTTP.

5. Превращение данных в интерактивные визуализации с помощью Plotly и D3

После очистки и уточнения данных наступает этап визуализации, на котором мы показываем выбранные представления набора данных, чтобы пользователи могли изучить их даже в интерактивном режиме. В зависимости от данных вы можете создавать классические диаграммы, карты или принципиально новые виды визуализаций.

Plotly — великолепная библиотека, которая позволяет создавать графики на Python и переносить их в интернет. Как мы увидим в главе 14, у Plotly также есть JavaScript API, который имитирует Python API, предоставляя доступ к бесплатной нативной JS-библиотеке для построения графиков.

D3 — библиотека визуализации на JavaScript — является одним из самых мощных инструментов визуализации среди всех существующих библиотек. Мы будем использовать D3, чтобы создать современную визуализацию данных о Нобелевской премии с множеством интерактивных элементов, с помощью которых пользователь может получить интересующую его информацию. Освоить D3 может быть непросто, но я быстро введу вас в курс дела и подготовлю так, чтобы вы смогли самостоятельно оттачивать навыки на практике.

Небольшие библиотеки

Помимо крупных библиотек, о которых шла речь выше, существуют и библиотеки более скромного размера, но для тулчейна также необходимые. В частности, в богатой экосистеме Python можно отыскать небольшие специализированные библиотеки для выполнения практически любой работы. Среди них выделяются следующие:

Requests
: Популярная HTTP-библиотека на Python, которая полностью оправдывает свой девиз «HTTP для людей». Requests намного превосходит urllib2 — один из модулей Python.

SQLAlchemy
: Лучшая Python-библиотека для работы с SQL с применением технологии ORM (Object Relational Mapping). Обладает богатым функционалом и существенно упрощает работу с различными реляционными БД.

Seaborn

Этот программный пакет — отличное дополнение к Matplotlib, мощной Python-библиотеке для работы с графикой. Он предоставляет несколько дополнительных типов графиков, в том числе статистических, очень полезных для визуализации данных. Также можно утверждать, что seaborn улучшает внешний вид графиков, переопределяя стандартные настройки Matplotlib.

Crossfilter

В основном JavaScript-библиотеки для обработки данных требуют доработки, но недавно среди них появилось несколько действительно полезных. Особо стоит отметить Crossfilter.js, которая чрезвычайно быстро обрабатывает наборы многомерных данных. Она идеально подходит для визуализации, что неудивительно — в число разработчиков входит создатель D3 Майк Бостоk (Mike Bostock).

Marshmallow

Очень удобная библиотека, которая преобразует сложные типы данных, такие как объекты, в типы данных Python и обратно.

Как пользоваться этой книгой

Хотя вся книга посвящена трансформации данных, ее необязательно читать последовательно, от корки до корки. Первый раздел содержит вводные сведения, вполне вероятно, что многие читатели хорошо знакомы с азами визуализации в браузере с использованием Python и JavaScript-инструментов. Читайте то, о чем не знаете. При необходимости можно вернуться для уточнения деталей по ссылкам, приведенным в тексте. Читатели, знакомые с обоими языками, могут пропустить главу «Обучающий мостик между Python и JavaScript», хотя и они, возможно, найдут там что-то ценное.

Остальные части книги, рассказывающие о тулчейне для преобразования обычного списка в полноценную интерактивную D3-визуализацию, в основном не зависят друг от друга. Если вы хотите сразу погрузиться в раздел III, посвященный очистке и исследованию данных с помощью pandas, смело приступайте, но учтите, что изначально предполагается наличие набора грязных данных о Нобелевской премии. Если вам удобнее читать в таком порядке, вы можете позже вернуться и посмотреть, как получить набор данных с помощью Scrapy. Аналогично, если вы хотите сразу же приступить к созданию приложения для визуализации, переходите к разделу IV и разделу V, но помните, что для нее требуется набор очищенных данных.

Вы можете читать главы в любом порядке, но если планируете заниматься визуализацией профессионально, в конечном итоге вам придется освоить все базовые навыки, описанные в книге.

Немного контекста

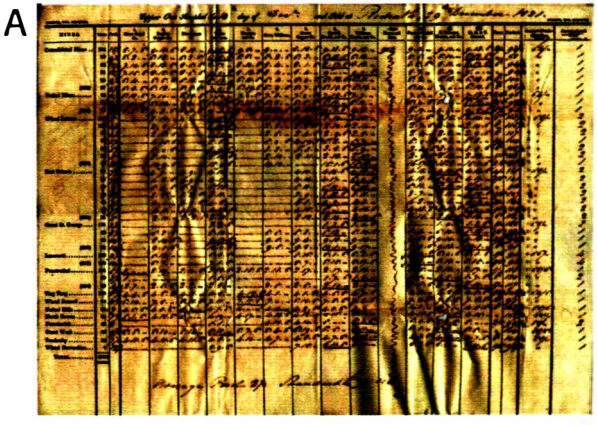
Это практическая книга. Предполагается, что читатель знает, что именно хочет визуализовать, представляет себе, как должна выглядеть и работать визуализация, и стремится реализовать идею, не перегружая себя теорией. Тем не менее, небольшой экскурс в историю визуализации данных не только пояснит центральную тему книги, но и добавит ценный контекст. История также помогает объяснить, почему сейчас самое подходящее время заняться областью визуализации — технологические инновации стимулируют появление новых форм визуализации данных, а мы сталкиваемся с проблемой отображения растущего объема многомерных данных, которые генерирует интернет.

Есть несколько замечательных книг по теории визуализации данных, которые я советую прочесть (см. небольшую подборку «Рекомендуемые книги» на стр. 34). Полезно понимать, как лучше подавать визуальную информацию. Например, круговая диаграмма — почти всегда менее удачный способ сравнения данных, чем простая гистограмма. Психометрические эксперименты показали, как можно обмануть зрение и усложнить восприятие связей в данных. И наоборот, некоторые визуальные формы могут четко проявить эти связи. Литература, как минимум, предлагает некоторые практические рекомендации, как достичь от визуализации данных желаемого эффекта.

По сути, хорошая визуализация выявляет или подчеркивает существующие закономерности или тенденции в данных: эмпирических или же полученных в результате абстрактных математических исследований (например, прекрасные фрактальные узоры множества Мандельброта). Закономерности могут быть простыми (например, средний вес по стране) или результатом сложного статистического анализа (например, кластеризация данных в многомерном пространстве).

Можно представить эти данные в их исходном состоянии, как беспорядочный набор чисел или категорий, плавающих в тумане. Никакие закономерности или корреляции не просматриваются. Легко упустить из виду, что скромная сводная таблица (Рис. I-3a) — это тоже форма визуализации. Организация данных в строки и столбцы — попытка упорядочить их, упростить обработку и выявить неточности (например, в финансовой отчетности). Конечно, далеко не каждый может выявлять закономерности в рядах чисел, поэтому были

разработаны более доступные для восприятия визуальные формы, которые воздействуют на зрительную кору головного мозга — наш основной канал получения информации о мире. Все мы знаем гистограмму, круговую диаграмму[1] и линейную диаграмму. Чтобы представить статистические данные в более доступной форме, использовали и более творческий подход. Один из самых известных примеров — визуализация данных по злополучному вторжению Наполеона в Россию в 1812 году (рисунок I-3b). Ее создал в XIX веке Шарль Жозеф Минар (Charles Joseph Minard).



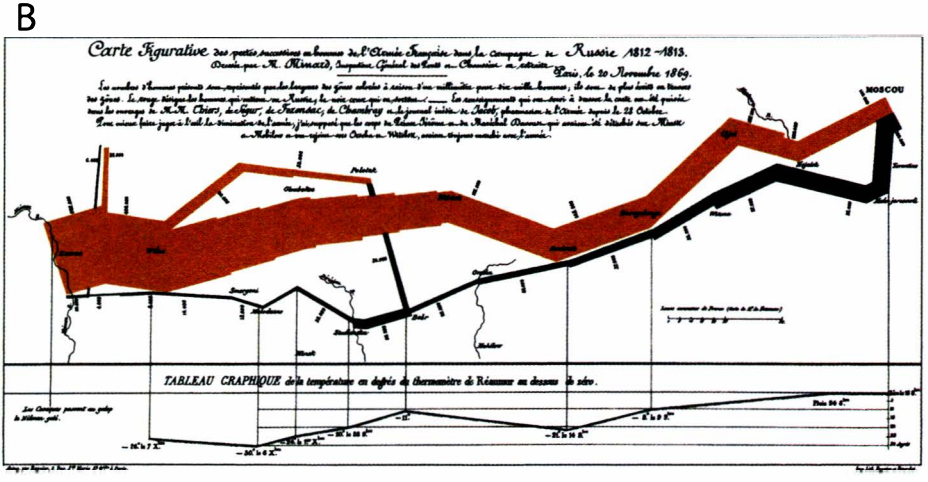


Рис. I-3. *(a) Сводная таблица и (b) визуализация «русской кампании» Наполеона 1812 г., выполненная Шарлем Жозефом Минаром*

[1] Круговая диаграмма впервые появилась в «*Статистическом бревиарии*», вышедшей в 1801 году книге Уильяма Плейфэра (William Playfair).

Светло-коричневая полоса на рисунке I-3b показывает наступление армии Наполеона на Москву, а черная линия — отступление. Толщина полосы соответствует численности армии Наполеона, которая таяла по мере потерь. График внизу показывает температуру по пути следования. Обратите внимание, как изящно Минар скомбинировал многомерные данные (статистику потерь, географические координаты и температуру), чтобы создать полное впечатление о сокрушительном разгроме, чего трудно было добиться другим путем. Только представьте себе попытку выявить необходимые связи, перескакивая от таблицы потерь к списку локаций. Я считаю, что задача, которая стоит перед современной интерактивной визуализацией данных, схожа в задачей Минара: как выйти за рамки традиционных одномерных гистограмм (подходящих во многих случаях) и найти новые эффективные способы выражать многоуровневые закономерности.

Еще не так давно визуализации мало чем отличалась от графики времен Минара. Статичные и предварительно отрендеренные модели отображали данные в пусть важном, но единственном представлении, с которым пользователи не могли взаимодействовать. В этом смысле изображение на мониторе отличалось от чертежа на бумаге всего лишь масштабом распространения.

Переход к интернету просто заменил газетную бумагу пикселями, но визуализация по-прежнему осталась некликабельной и статичной. Теперь, благодаря мощным библиотекам визуализации, среди которых лидирует D3, и возросшей производительности JavaScript появился новый тип визуализации — доступный и динамичный, стимулирующий исследования и открытия. Граница между анализом данных и их отображением стирается. Этот новый тип визуализации данных, которому посвящена моя книга, делает сегодня сферу веб-визуализации такой увлекательной. Разработчики пытаются создать новые способы визуализации данных и сделать их более доступными и полезными для пользователей. Это настоящая революция!

Резюме

Сейчас веб-визуализация данных превратилась в увлекательное занятие. Инновационные интерактивные визуализации создаются одна за другой, и многие из них, если не большинство, разрабатываются с помощью D3. JavaScript — единственный язык, который исполняется в браузере, поэтому код крутых визуальных эффектов или пишется на нем, или преобразуется в него. Однако в JavaScript отсутствуют инструменты или среда, необходимые для менее заметных, но не менее важных аспектов современной визуализации: агрегации,

хранения и обработки данных. В работе с данными командует парадом Python — универсальный, лаконичный, в высшей степени удобочитаемый язык программирования с доступом к постоянно растущему числу первоклассных инструментов. Многие из них могут использовать высокоэффективные низкоуровневые библиотеки, что упрощает и ускоряет обработку данных в Python.

В книге рассматриваются как тяжеловесные инструменты, так и множество более легких, но не менее важных. Книга также показывает, что комбинация Python и JavaScript являются лучшим из существующих стеков технологий для создания веб-визуализаций.

Далее следует первый раздел книги, который дает предварительные знания, необходимые для использования тулчейна. Вы можете как проработать этот раздел, так и перейти сразу к разделу II и началу описания тулчейна, обращаясь к первому разделу по мере необходимости.

Рекомендуемые книги

Здесь перечислены несколько ключевых книг по визуализации данных, которые могут вас заинтересовать. Они охватывают весь спектр: от интерактивных информационных панелей до красивой, интуитивно понятной инфографики.

- Bertin, Jacques. *Semiology of Graphics: Diagrams, Networks, Maps*. Esri Press, 2010.
- Cairo, Alberto. *The Functional Art*. New Riders, 2012.
- Few, Stephen. *Information Dashboard Design: Displaying Data for At-a-Glance Monitoring*, 2nd Ed. Analytics Press, 2013.
- Rosenberg, Daniel and Anthony Grafton. *Cartographies of Time: A History of the Timeline*. Princeton Architectural Press, 2012.
- Tufte, Edward. *The Visual Display of Quantitative Information*, 2nd Ed. Graphics Press, 2001.
- Wexler, Steve. *The Big Book of Dashboards*. Wiley, 2017.
- Wilke, Claus. *Fundamentals of Data Visualization*. O'Reilly, 2019. (Бесплатная онлайн-версия)

РАЗДЕЛ I

Базовый пакет инструментов

Этот раздел о базовом пакете инструментов для будущего тулчейна отчасти является справочником, отчасти — учебным пособием. Предполагаю, что у целевой аудитории книги уже имеется довольно широкий круг знаний, и некоторые темы этого раздела могут оказаться ей знакомыми. Мой совет — выбирайте материал, который восполнит пробелы в ваших знаниях или поможет что-то освежить в памяти.

Если вы уверены, что уже владеете базовым пакетом инструментов, переходите к Разделу II, чтобы начать знакомиться с нашим тулчейном.



Исходный код для этого раздела доступен в репозитории книги на GitHub: https://github.com/Kyrand/dataviz-with-python-and-js-ed-2.

Подготовка окружения

Из этой главы вы узнаете о загрузке и установке необходимых программных компонентов, а также получите краткие рекомендации по среде разработки. Вы увидите, что настройка окружения теперь стала проще. Я по отдельности расскажу о зависимостях Python и JavaScript и сделаю краткий обзор интегрированных сред разработки (IDE) для разных языков.

Сопутствующий код

В репозитории GitHub размещена основная часть кода, приведенного в этой книге, включая полную визуализацию данных о лауреатах Нобелевской премии. Чтобы скачать его, выполните следующую команду в подходящем каталоге:

```
$ git clone https://github.com/Kyrand/dataviz-with-python-and-js-ed-2.git
```

После выполнения будет создан локальный каталог *dataviz-with-python-and-js-v2* с основным исходным кодом, описанным в книге.

Python

Поскольку мы будем использовать в основном Python-библиотеки, их установка для разных операционных систем значительно упрощается благодаря Anaconda — платформе Python, которая объединяет в удобный пакет большинство популярных библиотек для анализа данных. Без нее было бы сложно предоставить исчерпывающие инструкции по установке каждой библиотеки отдельно, учитывая особенности разных ОС. Предполагается, что вы используете Python 3, первая версия которого вышла в 2008 году.

Anaconda

Установка некоторых крупных библиотек Python, таких как NumPy, ранее была сложной, в частности потому, что она включает в себя низкоуровневый код на C и на Fortran. Теперь процесс упростился, большинство библиотек легко установить при помощи модуля `Python easy_install` или команды `pip`:

```
$ pip install NumPy
```

Однако некоторые библиотеки, обрабатывающие большие массивы чисел, все еще непросто устанавливать. Проблемы с управлением зависимостями и версиями Python (если на одной машине установлено несколько его версий) могли бы усугубить ситуацию, но Anaconda решает эти трудности. Она устраняет необходимость проверки зависимостей и установки бинарных файлов. Anaconda — очень удобный инструмент для этой книги.

Чтобы скачать бесплатный дистрибутив, перейдите на сайт Anaconda, выберите подходящую для вашей ОС версию (желательно с Python не ниже 3.5). Далее следуйте инструкциям. Для Windows и macOS доступен графический установщик (просто загрузите и дважды щелкните), а для Linux нужно запустить небольшой bash-скрипт:

```
$ bash Anaconda3-2021.11-Linux-x86_64.sh
```

Инструкции по установке (https://www.anaconda.com/docs/getting-started/anaconda/install):

— Для Windows
— Для macOS
— Для Linux

Рекомендую при установке Anaconda придерживаться инструкций по умолчанию.

Пошаговое официальное руководство см. на сайте Anaconda. Пользователи Windows и macOS могут использовать графический интерфейс Anaconda Navigator или, как и пользователи Linux, работать через командную строку conda.

Установка дополнительных библиотек

Anaconda включает в себя большинство Python-библиотек, которые используются в книге (список пакетов см. в документации Anaconda). Если нужна

библиотека, не входящая в Anaconda, мы можем использовать pip (Pip Installs Python), который де-факто является стандартным инструментом для установки Python-библиотек. Применять `pip` для установки элементарно — наберите в командной строке команду `pip install имя_пакета`. Пакет будет установлен, либо вы получите понятное сообщение об ошибке:

```
$ pip install dataset
```

Виртуальные окружения

Виртуальные окружения создают изолированную среду разработки с конкретной версией Python и/или набором сторонних библиотек. Таким образом, вы не загрязните глобальную среду Python ненужными пакетами и сможете свободно экспериментировать с разными их версиями или при необходимости менять версию Python). Использование виртуальных окружений — лучшая практика разработки на Python, настоятельно рекомендую следовать ей.

Anaconda поставляется с системной командой `conda`, которая облегчает создание и использование виртуальных окружений. Создадим виртуальное окружение для работы над книгой, используя полный набор пакетов Anaconda:

```
$ conda create --name pyjsviz anaconda
...
#
# Чтобы активировать это окружение, используйте:
# $ source activate pyjsviz
#
# Чтобы выйти из окружения, используйте:
# $ source deactivate #
```

Как сказано в комментариях, чтобы использовать виртуальное окружение, вам нужно только активировать его — `source activate` (в Windows `source` можно опустить):

```
$ source activate pyjsviz
discarding /home/kyran/anaconda/bin from PATH prepending /home/
kyran/.conda/envs/pyjsviz/bin to PATH (pyjsviz) $
```

Обратите внимание, что в командной строке дается полезная подсказка, какое виртуальное окружение сейчас активно.

Команды `conda` способны на большее, чем просто облегчать работу с виртуальными окружениями. Conda, помимо прочего, сочетает в себе функциональность утилиты `virtualenv` и установщика пакетов Python `pip`. Полный перечень команд см. в документации Anaconda.

Создавать виртуальные окружения Python стало намного проще, когда в стандартную библиотеку Python включили модуль venv. Чтобы создать виртуальное окружение, наберите в командной строке:

```
$ python -m venv python-js-viz
```

Эта команда создает каталог `python-js-viz`, содержащий различные элементы виртуального окружения, в том числе некоторые скрипты активации. Чтобы активировать виртуальное окружение на macOS или Linux, запустите следующий скрипт активации:

```
$ source python-js-viz/bin/activate
```

На Windows запустите файл *activate.bat:*

```
$ python-js-viz/Scripts/activate.bat
```

Затем с помощью `pip` вы можете устанавливать Python-библиотеки в виртуальное окружение, избегая конфликтов с глобальным пространством имен Python:

```
$ (python-js-viz) pip install NumPy
```

Чтобы установить все необходимые библиотеки, используйте файл *requirements.txt* из репозитория книги на GitHub:

```
$ (python-js-viz) pip install -r requirements.txt
```

Информацию о виртуальном окружении см. в документации Python.

JavaScript

Спешу обрадовать: нам понадобится совсем немного инструментов JavaScript. Единственное, что должно быть обязательно, — браузер Chrome/Chromium,

который используется в этой книге. Этот кросс-платформенный браузер предлагает самый мощный набор инструментов разработчика.

Чтобы скачать Chrome, просто перейдите на главную страницу и загрузите версию для вашей операционной системы, которая определится автоматически.

Все JS-библиотеки, которые используются в книге, есть в репозитории GitHub. Как правило, используются два способа их доставки в браузер. Можно пользоваться CDN (Content Delivery Network, «сеть доставки контента»), которая эффективно кэширует копию библиотеки из сети доставки. Второй вариант — поместить библиотеку в локальное хранилище браузера. Оба метода используют тег `script` в HTML-документе.

Сети CDN

Вместо JS-библиотеки, установленной на вашей машине, браузер может получать JavaScript из интернета, с ближайшего кэширующего сервера CDN. С географически распределенных серверов контент доставляется гораздо быстрее, чем вы можете это сделать самостоятельно.

Чтобы использовать библиотеку из CDN, примените тег `<script>`, разместив его в нижней части HTML-страницы. Например, так вы вызовете текущую версию D3:

```
<script
 src="https://cdnjs.cloudflare.com/ajax/libs/d3/7.1.1/d3.min.js"
 charset="utf-8">
</script>
```

Локальная установка библиотек

Если у вас проблемы с интернетом или необходимо работать автономно, есть несколько простых способов установить JS-библиотеки локально.

Можно загрузить отдельные библиотеки в каталог static вашего локального сервера. Ниже показана типичная структура каталогов. Сторонние библиотеки размещаются в каталоге *static/libs*, внутри корневого каталога проекта. Например:

```
nobel_viz/
└── static
    ├── css
    ├── data
    ├── libs
    │   └── d3.min.js
    └── js
```

Теперь, чтобы использовать D3, внутри тега `<script>` укажите следующий путь:

```
<script src="/static/libs/d3.min.js"></script>
```

Базы данных

Для небольших и средних проектов визуализации рекомендуется использовать замечательную однофайловую бессерверную СУБД SQLite. Нам будет достаточно ее возможностей для всего нашего тулчейна.

Мы также рассмотрим основы взаимодействия Python с MongoDB, самой популярной нереляционной базой данных (NoSQL DB):

SQLite
: SQLite предустанавливается на устройства вместе с macOS и Linux. Для Windows следуйте руководству.

MongoDB
: Инструкции по установке MongoDB для различных операционных систем смотрите в ее официальной документации.

Обратите внимание, что мы будем использовать SQL-библиотеку Python SQLAlchemy либо напрямую, либо через другие Python-библиотеки. Так что что мы можем адаптировать любые примеры SQLite для другой SQL-базы данных (например, MySQL или PostgreSQL), изменив одну-две строки конфигурации.

Настройка и запуск MongoDB

Установка MongoDB может потребовать некоторых усилий. Как я уже говорил, устанавливать MongoDB нам не понадобится, но если вы хотите попробовать работать с ней, вот несколько рекомендаций.

Инструкции по установке MongoDB для macOS ищите в официальной документации.

Руководство для Windows из официальной документации поможет настроить и запустить MongoDB на сервере. Для создания необходимых каталогов данных и выполнения операций вам потребуются права администратора.

Сейчас MongoDB чаще всего устанавливают на сервер Linux (обычно Ubuntu), который использует для установочных пакетов формат deb. В официальной документации MongoDB подробно описывается установка для Ubuntu.

В зависимости от установки, вам может потребоваться вручную создать каталог *data*, в котором MongoDB хранит базы данных. На устройстве macOS или Linux командой `mkdir` как суперпользователь (`sudo`) создайте корневой каталог *data*:

```
$ sudo mkdir /data
$ sudo mkdir /data/db
```

Затем назначьте себя владельцем:

```
$ sudo chown 'whoami' /data/db
```

Для Windows, установив MongoDB Community Edition, создайте каталог *data* следующей командой:

```
$ cd C:\
$ md "\data\db"
```

При установке для Linux сервер MongoDB часто запускается по умолчанию. Также экземпляры сервера Linux и macOS запускает команда:

```
$ mongod
```

Чтобы запустить экземпляр сервера для Windows Community Edition, введите в командной строке следующее:

```
C:\mongodb\bin\mongod.exe
```

Как упростить работу MongoDB с Docker

Установка MongoDB иногда может преподносить сюрпризы. Например, в новых версиях Ubuntu (начиная с 22.04) есть проблемы с совместимостью SSL-библиотек. Если у вас установлен Docker, вы можете легко развернуть локальную базу данных для разработки на порту по умолчанию всего одной командой:

```
$ sudo docker run -dp 27017:27017 -v local-mongo:/data/db
             -- name local-mongo --restart=always mongo
```

Это позволяет избежать проблем с несовместимыми библиотеками.

Интегрированная среда разработки

В подразделе «Мифы об IDE, инструментах и фреймворках» на стр. 128 я объясняю, что не обязательно использовать IDE, чтобы программировать на Python или JavaScript. Благодаря инструментам, которые предоставляют современные браузеры, в частности Chrome, вам будет достаточно хорошего редактора кода.

Тем не менее, сейчас для JavaScript-проектов среднего и высокого уровня сложности часто используют фреймворки типа React, Vue и Svelte, которым хорошая IDE дает преимущества, особенно при обработке мультиформатных файлов, содержащих одновременно HTML, CSS и JS. Но в целом стандартом для современной веб-разработки стал бесплатный редактор кода Visual Studio Code (VSCode). У этого редактора есть плагины практически для всего, а также большое и активное сообщество, поэтому на свои вопросы вы, как правило, получите ответы, а ошибки будут быстро обнаружены.

Я не отдаю предпочтения ни одной из специализированных IDE для Python, которые опробовал. В первую очередь я пытался найти приличную систему отладки. Устанавливать точки останова для Python с помощью редактора не слишком удобно, а запускать отладчик `pdb` из командной строки — как-то старомодно. Тем не менее, у Python есть неплохая система логирования, позволяющая избежать довольно неуклюжей отладки по умолчанию. VSCode довольно хорош для программирования на Python, но есть специализированные IDE для Python, в которых процесс разработки идет более гладко.

Вот три IDE, которые я протестировал и счел приемлемыми:

PyCharm
: IDE с серьезным инструментарием, помогающим писать и отлаживать программный код. Вероятно, стала бы фаворитом по результатам опроса опытных питонистов.

PyDev
: Если вам нравится Eclipse и вас не пугает заметное сходство с ним, то это вариант для вас.

Wing Python IDE
: Надежная IDE с отличным отладчиком. Постепенно совершенствовалась в течение примерно 15 лет.

Резюме

Благодаря пакетному менеджеру для Python-библиотек Anaconda и встроенным в браузер инструментам разработки JavaScript все необходимое для разработки на Python и JavaScript можно получить парой кликов мыши. Добавьте редактор кода и базу данных по вашему выбору[1], и все готово. Дополнительные библиотеки, такие как *node.js*, могут быть полезными, но использовать их необязательно. Итак, мы определились с программным окружением и в следующих главах подготовимся к старту преобразования данных с помощью тулчейна. Начнем с языкового мостика между Python и JavaScript.

[1] SQLite обходится без сервера и прекрасно подходит для разработки.

Обучающий мостик между Python и JavaScript

Пожалуй, самая смелая идея книги — работа с двумя языками программирования сразу, причем читателю достаточно владеть только одним из них. Это осуществимо потому, что Python и JavaScript довольно простые для понимания языки и имеют много общего. Цель данной главы — выявить общие черты и построить учебный мостик между языками, чтобы основные навыки работы с одним вы могли легко применить к другому.

Сначала я покажу сходство и различие между Python и JS, а затем объясню, как создать среду для их изучения. Основная часть главы посвящена ключевым синтаксическим и концептуальным различиям, после чего рассматриваются шаблоны и идиомы, которые я часто использую при визуализации данных.

Сходство и различия

Если не касаться синтаксиса, у Python и JavaScript действительно много общего. Со временем вы будете переходить с одного на другой естественно, безо всякого напряжения[1]. Сравним оба языка с точки зрения визуализации данных.

Основное сходство:

— Оба языка интерпретируемые, а не компилируемые.
— Оба поддерживают интерактивный интерпретатор, что позволяет вводить строки кода и сразу видеть результат.
— В обоих языках есть garbage collection («сборка мусора») — механизм автоматического управления памятью.
— Ни в одном, ни в другом нет заголовочных файлов, boilerplate-кода и прочего, свойственного таким языкам, как C++, Java и др.

[1] Различия между методами обработки списков/массивов в этих языках могут немного сбивать с толку. В Python метод pop удаляет элемент из списка, а append добавляет элемент в конец списка. В JavaScript элементы в конец массива добавляет push, а append используется для объединения массивов.

— Для разработки на Python и JavaScript достаточно текстового редактора или легковесной IDE.
— В обоих языках функции являются объектами первого класса и могут передаваться в качестве аргументов.

Основные различия:

— Одно из главных различий: язык JavaScript — однопоточный, с неблокирующими асинхронными операциями ввода/вывода, то есть действия вроде доступа к файлу предполагают использование коллбэк-функций, которые передаются другим функциям в качестве аргумента и вызываются после выполнения некоего кода, обычно асинхронно.
— JS до недавнего времени работал только в браузерах[1], и используется в основном для веб-разработки, в то время как Python может решать широкий круг задач.
— JS — единственный первоклассный язык для браузеров, про Python такого не скажешь.
— Python предлагает всеобъемлющую стандартную библиотеку, в то время как у JS есть только ограниченный набор объектов, например: JSON, Math.
— В Python используются объектно-ориентированные классы, а в JS — прототипы объектов.
— Среди JS-библиотек практически отсутствуют мощные универсальные библиотеки для обработки данных[2].

Эти различия и привели к необходимости использовать в книге оба языка. При создании визуализаций в браузере JavaScript требуется поддержка со стороны традиционных инструментов обработки данных, и Python тут вне конкуренции.

Взаимодействие с кодом

Одно из основных преимуществ Python и JavaScript заключается в том, что с их кодом можно взаимодействовать на лету — он интерпретируется. Интерпретатор Python можно запустить из командной строки. Для JavaScript используется

[1] Появление Node.js позволило выполнять JavaScript-код на сервере.

[2] Хотя ситуация и меняется с развитием библиотек типа TensorFlow.js и построенной на ее основе Danfo.js (JS-фреймворка, подражающего pandas), но в этой области JS по-прежнему уступает Python, R и др.

консоль браузеров, поддерживающих встроенные инструменты разработчика. В этом подразделе мы рассмотрим, как запустить сеанс интерпретатора и начать тестирование кода.

Python

Несомненно, лучший интерпретатор командной строки для Python — это IPython, который выпускается в трех вариантах: базовый для терминала/командной строки, расширенная графическая версия и блокнот (notebook) для браузера. Начиная с версии IPython 4.0, два последних модуля перенесли в отдельный проект Jupyter. Jupyter Notebook представляет собой мощную интерактивную вычислительную среду, работающую в браузере. Большое преимущество блокнотов Jupyter — доступ к интернету и возможность сохранять сеанс[1]. Блокноты отлично подходят для обучения и восстановления программного контекста благодаря простоте обмена сессиями программирования и встроенной визуализации данных. Поэтому я и включил блокноты Jupyter в главы, посвященные Python.

Чтобы запустить панель Jupyter, введите `jupyter notebook` в командную строку:

```
$ jupyter notebook
[I 15:27:44.553 NotebookApp] Serving notebooks from local
directory:
...
[I 15:27:44.553 NotebookApp] http://localhost:8888/?token=5e09...
```

Затем откройте вкладку браузера по заданному URL-адресу (в данном случае *http://localhost:8888)* и приступайте к чтению или записи в блокноты.

JavaScript

Поскольку интерпретатор JavaScript встроен во все современные браузеры, вы можете протестировать JS-код без запуска сервера. Проверить, как работают фрагменты JavaScript, HTML и CSS кода можно с помощью специальных онлайн-сервисов. Один из лучших — CodePen. На таких сайтах можно обмениваться кодом и тестировать сниппеты, а также обычно можно добавлять библиотеки, такие как *D3.js*, всего несколькими кликами мыши.

Посмотреть значения переменных при исполнении кода или проверить однострочный скрипт лучше всего в консоли браузера. Для доступа к консоли

[1] Благодаря выполнению интерпретатора Python на сервере.

браузера Chrome нажмите сочетание клавиш Ctrl+Shift+J (для Mac: Command + Option + J). Кроме тестирования JS-сниппетов, консоль позволяет проанализировать любые объекты в области видимости, их методы и свойства. Это прекрасный способ проверить состояние объекта и обнаружить ошибки.

Один из недостатков использования онлайн-редакторов JavaScript — вы лишаетесь возможностей привычной среды редактирования: линтинга, знакомых «горячих клавиш» и тому подобного (см. главу 4). Обычно онлайн-редакторы довольно примитивны. Если предстоит серьезный сеанс JavaScript, и вы хотите пользоваться излюбленным редактором, запустите локальный сервер.

Сначала создайте каталог проекта, например с именем *sandpit*, и добавьте в него минимальный HTML-файл:

```
sandpit
├── index.html
└── script.js
```

В файле *index.html* будет всего несколько строк обязательного кода, тег <script>, со ссылкой на внешний файл с JS-кодом, и необязательный `div` с плейсхолдером. Это основа, с которой вы можете начать создавать визуализации или экспериментировать с DOM:

```
<!-- index.html -->
<!DOCTYPE html>
<meta charset="utf-8">

<div id='viz'></div>

<script type="text/javascript" src="script.js" async></script>
```

Добавьте немного кода JavaScript в файл *script.js:*

```
// script.js
let data = [3, 7, 2, 9, 1, 11];
let sum = 0;
data.forEach(function(d){
   sum += d;
});

console.log('Sum = ' + sum);
// outputs 'Sum = 33'
```

SKLADCHIK.ORG

С помощью модуля Python `http` запустите в каталоге проекта dev-сервер:

```
$ python -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 ...
```

Откройте браузер по адресу *http://localhost:8000*, нажмите Ctrl+Shift+J (Cmd+Opt+J для macOS), чтобы перейти в консоль, где вы должны увидеть результат выполнения скрипта, как на рисунке 2.1 (подробнее в главе 4).



Рис. 2.1. *Вывод на консоль Chrome*

Мы выяснили, как запускать демонстрацию кода, и теперь начнем наводить мосты между Python и JavaScript. Сначала остановимся на основных различиях в синтаксисе. Вы увидите, что различия не очень значительные и их легко запомнить.

Строим мост

В этом подразделе я сравню ключевые особенности программирования на обоих языках.

Руководства по стилю кода, PEP 8 и директива 'use strict'

В то время как руководства по стилю для JavaScript не так уж строги (разработчики часто по умолчанию придерживаются стиля, принятого в крупных библиотеках, таких как React), для Python есть стандарт PEP (Python Enhancement Proposal, «предложения по развитию Python»). Я рекомендую ознакомиться с PEP-8, но не воспринимать его как истину в последней инстанции. В большинстве случаев его рекомендации правильны, но оставляйте некоторый простор для личного выбора. Для выявления любых отклонений от PEP-8 есть удобная онлайн-проверка PEP8 Online. Многие питонисты пользуются Black, инструментом для форматирования кода Python в соответствии с PEP-8.

Для обозначения отступов блока кода Python принято использовать четыре пробела. В JavaScript строгого требования в отступам нет, но чаще всего для отступа используются два пробела.

Одно из недавних дополнений к JavaScript (ES5) — директива `'use strict'`, которая активирует строгий режим (strict mode). Я настоятельно рекомендую его использовать. Этот режим вынуждает соблюдать лучшие практики JavaScript, например, препятствует случайному объявлению глобальных переменных. Чтобы включить строгий режим, поместите `'use strict'` в верхнюю часть модуля или функции:

```
(function(foo){
   'use strict';
   // ...
}(window.foo = window.foo || {}));
```

Нотации CamelCase и другие

В JS и Python по-разному именуют переменные. В JS применяют camelCase (см. пример 2-4: `processStudentData`), а в переменных Python — в соответствии PEP-8 — snake_case, когда слова в нижнем регистре объединяются подчеркиваниями (пример 2-3: `process_student_data`). По соглашению об именовании в Python имена констант пишутся буквами в верхнем регистре с подчеркиваниями, имена классов — в стиле Upper CamelCase, остальные имена — в нижнем регистре с подчеркиваниями (см. пример ниже). Учтите, что соглашения в экосистеме Python соблюдать важнее, чем в JS.

```
FOO_CONST = 10
class FooBar(object): # ...
def foo_bar():
   baz_bar = 'some string'
```

Импорт модулей и скриптов

Использование собственных и сторонних библиотек — фундамент современного программирования. В Python есть простая и эффективная система импорта. Удивительно, но в JavaScript долго отсутствовал стандартный способ подключения библиотек[1].

Проблема была решена с появлением ECMAScript 6, добавившим в JavaScript выражения `import` и `export` для работы с инкапсулированными модулями. Теперь в JavaScript есть модули (обычно с расширением `.mjs`), которые позволяют

[1] Основное ограничение было связано с необходимостью доставки JS-скриптов через интернет по протоколу HTTP.

импортировать и экспортировать инкапсулированные функции и объекты. Это большой шаг вперед. В разделе V мы увидим, насколько легко с ними работать.

Вы можете импортировать сторонние библиотеки с помощью тега `script`, который добавляет их в глобальное пространство имен в качестве объекта. Например, чтобы добавить D3, напишите во входном HTML-файле (обычно *index.html*) следующий тег `script`:

```
<!DOCTYPE html>
<meta charset="utf-8">
   <script src="http://d3js.org/d3.v7.min.js"></script>
```

Затем вы можете использовать библиотеку D3, например так:

```
let r = d3.range(0, 10, 2)
console.log(r)
// out: [0, 2, 4, 6, 8]
```

Python поддерживает философию «все включено», то есть его стандартная установка включает всесторонний набор библиотек с широким охватом: от контейнерных типов данных (collections) до работы с CSV-файлами. Подключить любую из этих библиотек можно с помощью ключевого слова `import`:

```
In [1]: import sys

In [2]: sys.platform
Out[2]: 'linux'
```

Для импорта отдельных компонентов библиотеки служит ключевое слово `from`, для использования псевдонима вместо имени — `as`:

```
import pandas as pd
from csv import DictWriter, DictReader
from numpy import * ❶

df = pd.read_json('data.json')
reader = DictReader('data.csv')
md = median([12, 56, 44, 33])
```

❶ Символ * означает, что из модуля в текущее пространство имен импортируются все переменные, и это, как правило, плохая практика. Есть вероятность, что импортированная переменная перекроет существующую, что противоречит правилу Дзена Python (PEP20) — «Явное лучше неявного». Исключение из правила — использование интерпретатора Python в интерактивном режиме. Только в этом случае имеет смысл импортировать все функции библиотеки, чтобы сократить длину строки кода. Например, если вы занимаетесь математическими расчетами, можно импортировать все математические функции (`from math import *`).

При импорте сторонней библиотеки Python пытается найти путь к ней с помощью `sys.path`, который содержит:

— каталог импортируемого модуля (текущий каталог);
— переменную `PYTHONPATH` — список путей к каталогам;
— дефолтный каталог, куда устанавливаются библиотеки с помощью `pip` или `easy_install` (в зависимости от настроек установщика).

Большие библиотеки часто являются пакетами, которые делятся на подмодули. Для доступа к подмодулям используется точечная нотация (dot notation):

```
import matplotlib.pyplot as plt
```

Пакет — каталог с организованными в систему файлами и обычно пустым файлом *__init__.py*, сообщающим Python — каталог является пакетом (см. пример 2-1). Благодаря init-файлам каталог становится видимым для системы импорта Python.

Пример 2-1. *Создание пакета Python*

```
mypackage
├── __init__.py
...
├── core
│   └── __init__.py
│   ...
...
└── io
    ├── __init__.py
    └── api.py
    ...
    └── tests
        └── __init__.py
```

```
        └── test_data.py
        └── test_excel.py ❶
        ...
...
```

❶ Этот модуль также можно импортировать с помощью следующей конструкции `from mypackage.io.tests import test_excel`.

Используя точечную нотацию, вы можете, начиная от корневого каталога (в примере 2-1 это `mypackage`), получить доступ к пакетам из списка `sys.path`. Особый случай импорта — intrapackage references. Подмодуль `test_excel.py` из примера 2-1 может импортировать подмодули из пакета `mypackage`. Импорт может быть, как абсолютным, так и относительным:

```
from mypackage.io.tests import test_data ❶
from . import test_data ❷
import test_data ❷
from ..io import api ❸
```

❶ Абсолютный импорт модуля `test_data.py` по полному пути от корневой папки пакета.
❷ Явный (`. import`) и неявный относительный импорт.
❸ Относительный импорт из пакета, родственного по отношению к пакету `tests`.

Модули JavaScript

В JavaScript есть модули для импорта и экспорта инкапсулированных переменных. Питонистам многое покажется знакомым в синтаксисе импорта JS, но на мой взгляд, в JS он организован лучше. Предлагаю вам краткий обзор.

Допустим, у нас есть точка входа — модуль JS `index.js`, который должен использовать функции или объекты библиотечного модуля `libFoo.js`, расположенного в каталоге `lib`. Структура файлов будет следующей:

```
.
├── index.mjs
└── lib
    └── libFoo.mjs
```

В `libFoo.mjs` мы экспортируем функцию-заглушку, используя конструкцию `export default`, которая применяется для экспорта одного объекта из модуля, — обычно API, например, со служебными методами:

```
// lib/libFoo.mjs
export let findOdds = (a) => {
   return a.filter(x => x%2)
}
let api = {findOdds} ❶

export default api
```

❶ Пример создания объекта с сокращенной формой имени свойства, которая эквивалентна записи `{findOdds: findOdds}`

Чтобы импортировать экспортированные функции и объекты из нашего модуля, используем выражение `import`, которое позволит импортировать как API по умолчанию, так и выбранные переменные в фигурных скобках:

```
// index.mjs
import api from './lib/libFoo.mjs'
import { findOdds } from './lib/libFoo.mjs'

let odds = findOdds([2, 4, 24, 33, 5, 66, 24])
console.log('Odd numbers: ', odds)

odds = api.findOdds([12, 43, 22, 39, 52, 21])
console.log('Odd numbers: ', odds)
```

Импорт и экспорт JS поддерживает псевдонимы, что способствует чистоте кода.

```
// index.mjs
import api as foo from './lib/libFoo.mjs'
import { findOdds as odds } from './lib/libFoo.mjs'
// ...
```

Как видите, в импорте и экспорте Python и JavaScript много общего, хотя, по-моему, в JS они реализованы удобнее. Подробнее об инструкциях export и import см. в официальной документации Mozilla.

Поддержка чистоты пространства имен

Переменные, определенные в модуле Python, находятся вне области видимости других модулей. Если переменные не импортированы явно (например `from foo import baa`), то нужно обращаться к ним из пространства имен импортированного модуля с помощью точечной нотации (например `foo.baa`). Тот факт, что каждый модуль имеет свое глобальное пространство имен, справедливо считается преимуществом и соответствует одному из ключевых принципов Python: «Явное лучше неявного». При анализе кода Python вы всегда можете точно определить, откуда взялся класс, функция или переменная. Пространства имен помогают избежать конфликта переменных или их перекрытия, что особенно важно при разрастании кодовой базы.

В прошлом JavaScript серьезно, и полностью справедливо, критиковали за нарушение соглашений о пространстве имен. Наиболее яркий пример — переменные, объявленные вне функций или не имеющие ключевого слова `var`[1], становились глобальными, а не ограничивались скриптом, в котором они объявлены. В современном модульном JavaScript появилась схожая с Python инкапсуляция, а также импорт и экспорт переменных.

Модули JS изменили правила игры относительно недавно. Раньше, чтобы изолировать локальные переменные от глобального пространства имен, широко использовались самовызывающиеся функции, но и сейчас вы их можете встретить в старом коде. При использовании таких функций переменные, объявленные через `var`, являются локальными переменными функции, не загрязняя глобальное пространство имен. Сейчас в JavaScript вместо `var` рекомендуется использовать ключевое слово let, поскольку оно обеспечивает блочную видимость переменных. Чтобы сделать доступными для других скриптов любые объекты, функции и переменные, их можно привязать к объекту, являющемуся частью глобального пространства имен.

В примере 2-2 используется паттерн «модуль». Обязательные элементы — объявление функции и ее немедленный вызов (с метками ❶ и ❷ в примере кода) — фактически создают инкапсулированный модуль. По сравнению с модульным JavaScript этот паттерн был далеко не идеальным, но наилучшим решением вплоть до момента, когда специализированную систему импорта из ES6 внедрили во всех основных браузерах.

[1] При использовании директивы ES5 `'use strict'` присвоение значения переменной без ее объявления приводит к сообщению об ошибке, а не к созданию глобальной переменной.

Пример 2-2. *Паттерн «модуль» для JavaScript*

```
(function(nbviz) { ❶
  'use strict';
  // ...
  nbviz.updateTimeChart = function(data) {..} ❷
  // ...
}(window.nbviz = window.nbviz || {})); ❸
```

❶ Получает глобальный объект `nbviz`.

❷ Привязывает метод `updateTimeChart` к глобальному объекту `nbviz`, фактически *экспортируя* его.

❸ Если объект `nbviz` существует в глобальном пространстве имен (пространстве имен объекта window), передает его в функцию модуля, в противном случае добавляет в глобальное пространство имен.

Вывод "Hello World!"

Самая популярная первая программа при изучении любого языка программирования — вывод сообщения «Hello World!» в той или иной форме. Напишем этот код на Python и JavaScript.

Вывод на языке Python такой, что проще не придумаешь:

```
print('Hello World!')
```

В JavaScript нет функции обычного вывода на экран, но можно вывести сообщение в консоль браузера:

```
console.log('Hello World!')
```

Простая обработка данных

Хороший способ получить представление о различиях — написать одну и ту же функцию на обоих языках. Примеры 2-3 и 2-4 показывают, как изменение данных реализуется на Python и JavaScript. Используем эти примеры для сравнения синтаксиса двух языков. Для удобства обозначим блоки кода заглавными буквами (A, B…).

Пример 2-3. *Простая трансформация данных с помощью Python*

```
# A
student_data = [
   {'name': 'Bob', 'id':0, 'scores':[68, 75, 56, 81]},
   {'name': 'Alice', 'id':1, 'scores':[75, 90, 64, 88]},
   {'name': 'Carol', 'id':2, 'scores':[59, 74, 71, 68]},
   {'name': 'Dan', 'id':3, 'scores':[64, 58, 53, 62]},
]

# B
def process_student_data(data, pass_threshold=60,
                         merit_threshold=75):
   """ Выполняет базовые статистические операции над данными
   студентов """

   # C
   for sdata in data:
      av = sum(sdata['scores'])/float(len(sdata['scores']))
      if av > merit_threshold:
         sdata['assessment'] = 'passed with merit'
      elif av >= pass_threshold:
         sdata['assessment'] = 'passed'
      else:
         sdata['assessment'] = 'failed'

      # D
      print(f"{sdata['name']}'s (id: {sdata['id']}) \
      final assessment is: {sdata['assessment'].upper()}")
      # Для версий младше Python 3.7 форматирование строк в старом
      # стиле эквивалентно # print("%s's (id: %d) final \
      # assessment is: %s"%(sdata['name'],sdata['id'], \
      # sdata['assessment'].upper())) sdata['name'], sdata['id'],\
      # sdata['assessment'].upper()))
      print(f"{sdata['name']}'s (id: {sdata['id']}) \
      final assessment is: {sdata['assessment'].upper()}")
   sdata['average'] = av

# E
if __name__ == '__main__':
   process_student_data(student_data)
```

Пример 2-4. *Простая трансформация данных с помощью JavaScript*

```
let studentData = [ ❶
   {name: 'Bob', id:0, 'scores':[68, 75, 76, 81]},
   {name: 'Alice', id:1, 'scores':[75, 90, 64, 88]},
   {'name': 'Carol', id:2, 'scores':[59, 74, 71, 68]},
   {'name': 'Dan', id:3, 'scores':[64, 58, 53, 62]},
];

// B
function processStudentData(data, passThreshold, meritThreshold){
   passThreshold = typeof passThreshold !== 'undefined'?\
   passThreshold: 60;
   meritThreshold = typeof meritThreshold !== 'undefined'?\
   meritThreshold: 75;

   // C
   data.forEach(function(sdata){
      let av = sdata.scores.reduce(function(prev, current){
         return prev+current;
      },0) / sdata.scores.length;
      if(av > meritThreshold){
         sdata.assessment = 'passed with merit';
      }
      else if(av >= passThreshold){
         sdata.assessment = 'passed';
      }
      else{
         sdata.assessment = 'failed';
      }
      // D
      console.log(sdata.name + "'s (id: " + sdata.id +
         ") final assessment is: " +
            sdata.assessment.toUpperCase());
      sdata.average = av;
   });
}
// E
processStudentData(studentData);
```

❶ Обратите внимание на преднамеренную и обоснованную непоследовательность: одни ключи объектов заключены в кавычки, а другие — нет.

Создание строк

В блоках D примеров 2-3 и 2-4 показан стандартный способ вывода результата в консоль или на экран. В JavaScript нет выражения `print`, но метод объекта `console` выведет данные в консоль браузера:

```
console.log(sdata.name + "'s (id: " + sdata.id +
   ") final assessment is: " + sdata.assessment.toUpperCase());
```

Обратите внимание: переменная целочисленного типа `id` преобразуется в строку, что позволяет применять конкатенацию. Python не поддерживает неявное приведение типов, поэтому попытка таким же образом добавить строку к целому числу выбросит ошибку. Вместо этого выполняется явное приведение числового типа к строке с помощью функции `str` или `repr`.

В сегменте A примера 2-3 строка результата формируется с использованием форматирования, как в языке C. Строковые (`%s`) и целочисленные (`%d`) плейсхолдеры предоставляются конечным кортежем (`%(...)`):

```
print("%s's (id: %d) final assessment is: %s"
   %(sdata['name'], sdata['id'], sdata['assessment'].upper()))
```

Сейчас я редко использую функцию `print()`, потому что предпочитаю более мощный и гибкий Python-модуль `logging`, задействованный в следующем блоке кода. Хотя использовать его не так просто, как print, но он предлагает гораздо больше возможностей. Модуль позволяет записывать результаты в файл с расширением.log и/или выводить на экран, задавать уровень логирования и многое другое. Подробнее см. в документации Python.

```
import logging
logger = logging.getLogger(__name__) ❶
//...
logger.debug('Some useful debugging output')
logger.info('Some general information')

// IN INITIAL MODULE
logging.basicConfig(level=logging.DEBUG) ❷
```

❶ Создает логгер с именем данного модуля.
❷ Установка уровня «debug», предназначенного для вывода наиболее детализированной информации (подробнее см. в документации Python).

Значение отступов и фигурных скобок

Значимые пробельные символы (whitespace) — одна из ключевых особенностей синтаксиса Python. В таких языках, как C и JavaScript пробельные символы служат только для удобства чтения, их можно удалить, чтобы сжать весь код в одну строку[1]. Однако в Python отступы (четыре пробела) определяют структуру кода, и удаление пробелов меняет смысл программы. Дополнительные усилия по правильному выравниванию кода Python с лихвой компенсируются улучшенной читабельностью: часто на чтение и понимание кода требуется больше времени, чем на его написание, и, вероятно, прекрасное состояние библиотечной экосистемы Python во многом зависит от простоты чтения кода. Четыре пробела фактически стали стандартом (см. PEP-8), лично я предпочитаю так называемую *мягкую табуляцию* (soft tabs), когда мой редактор автоматически вставляет при нажатии клавиши Tab нужное количество пробелов[2].

По соглашению в коде, приведенном ниже, отступ оператора `return` должен состоять из четырех пробелов[3]:

```
def doubler(x):
    return x * 2
# |<-этот отступ очень важен
```

В JavaScript число пробелов между операторами и переменными не играет роли, для разграничения блоков кода используются фигурные скобки. Обе приведенные ниже функции doubler работают одинаково:

```
let doubler = function(x){
    return x * 2;
}
let doubler=function(x){return x*2;}
```

[1] Выполняется JavaScript-компрессорами для уменьшения размера файлов загружаемых веб-страниц.

[2] Ведутся жаркие, но мало полезные споры о том, что лучше — пробелы или жесткая табуляция? PEP-8 рекомендует использовать пробелы, и меня это устраивает.

[3] Можно выбрать два или три пробела, главное, чтобы их количество было постоянным.

Пробельные символы очень важны в Python. Большинство моих знакомых хороших программистов настраивают свои редакторы так, чтобы обеспечивать единообразие отступов и стиля. Python буквально вынуждает придерживаться этих рекомендаций. Повторюсь, я уверен, что наряду с простым синтаксисом значительный вклад в развитие экосистемы Python вносит исключительная читабельность кода.

Комментарии и документация

Для добавления комментария в код Python используется # (октоторп, иначе хеш-символ):

```
# ex.py, пример однострочного комментария

data = {} # Наш основной набор данных
```

В JavaScript для однострочных комментариев используется двойной обратный слеш (//), а для многострочных — /* … */:

```
// script.js, пример однострочного комментария
/* Блок комментариев для описаний функций,
заголовков скриптов библиотек и др. */
let data = {}; // Наш основной набор данных
```

Кроме комментариев в Python имеются строки документации (docstring) — строковые литералы, которые повышают читабельность и прозрачность кода в соответствии с философией языка. Сразу под объявлением функции `process_student_data` из примера 2-3 есть строка текста в тройных кавычках, значение которой автоматически присваивается атрибуту `__doc__` данной функции. Кроме однострочных docstring применяются также многострочные:

```
def doubler(x):
    """Функция возвращает удвоенный аргумент."""
    return 2 * x
def sanitize_string(s):
    """Функция удаляет любые пробельные символы из начала и конца
    строки, а затем заменяет каждый оставшийся пробел на '-'
    """
    return s.strip().replace(' ', '-')
```

Написание docstrings — это хорошая практика, особенно при командной разработке. Они распознаются большинством приличных редакторов Python, и используются автоматизированными инструментами для генерации документации, такими как Sphinx. Доступ к строке документации функции или класса можно получить с помощью их атрибута `doc`.

Объявление переменной с помощью let или var

Для объявления переменных в JavaScript используются ключевые слова `let` и `var`. Обычно рекомендуется выбирать `let`.

В то время как в Python просто переходят на новую строку кода, выражения JS должны, строго говоря, заканчиваться точкой с запятой. Тем не менее, современные браузеры обычно корректно интерпретируют код даже без точек с запятой. Есть пара пограничных случаев (edge case), когда использовать точки с запятой необходимо: например, их отсутствие может привести к сбоям в работе минификаторов и компрессоров, которые удаляют пробельные символы из кода. Поэтому я рекомендую всегда ставить точки с запятой — с ними код выглядит аккуратнее и улучшается читабельность.



Особенность JavaScript — *поднятие (hoisting) переменной*, то есть переменные, объявленные с помощью `var`, обрабатываются прежде остального кода. Где бы ни была объявлена переменная, ее объявление (но не присваивание) «поднимается» в начало области видимости. Эта особенность может привести к странным ошибкам и путанице. Явное размещение `var` в начале кода поможет избежать этой проблемы, но лучше использовать современное `let`, чтобы объявлять переменные с блочной областью видимости.

Строки и числа

Строковые значения для переменной *name* в данных о студентах (см. блок A из примеров 2-3 и 2-4) в JavaScript будут интерпретироваться как UCS-2 (предшественник формата UTF-16)[1], а в Python 3 в Unicode (по умолчанию UTF-8)[2].

[1] Предположение, что JavaScript использует UTF-16 (это не так), становится причиной ошибок и вызванных ими серьезных проблем. Интересный анализ на эту тему есть в блоге Матиаса Байненса (Mathias Bynens).

[2] Кодирование и декодирование строк в Unicode — большая проблема Python 3. Рекомендую почитать об этом. В Python 2 использовались байтовые строки.

Оба языка позволяют заключать строки как в одинарные, так и в двойные кавычки. Если внутри строки используются одинарные кавычки, то ее нужно заключить в двойные, и наоборот, как в примере ниже:

```
pub_name = "The Brewer's Tap"
```

JavaScript поддерживает единый числовой тип number, представляющий целые и дробные числа. Данными такого типа являются `scores` из блока A примера 2-4 — числа с плавающей точкой двойной точности (IEEE-754), занимающие в памяти 64 бита. Функция `parseInt`[1] в JavaScript, как и функция `floor`, округляет числа с плавающей точкой до ближайшего меньшего целого, при этом тип данных остатся `number`:

```
let x = parseInt(3.45); // округляем x до 3
typeof(x); // "number"
```

В Python есть два основных числовых типа: 32-битные целые числа `int` (к этому типу преобразуются оценки студентов из примера) и числа с плавающей точкой `float` — эквивалент `number` в JS. Итак, в Python могут быть представлены любые целые числа, JavaScript в этом смысле имеет ограничение[2]. Числовой тип Python можно изменить следующим образом:

```
foo = 3.4 # type(foo) -> float
bar = int(3.4) # type(bar) -> int
```

Работа с числами в Python и JavaScript проста и интуитивна. Для более сложных вычисления в Python используйте библиотеку NumPy, которая обеспечивает точное управление числовыми типами (подробнее о NumPy в главе 7). В JavaScript, за исключением некоторых ультрасовременных проектов, обычно достаточно 64-битных чисел с плавающей точкой.

[1] Функция `parseInt` также применяется для преобразования строк в числа. Например, `parseInt(12.5px)` сначала удаляет px, затем преобразует строку в число и возвращает 12. В `radix`, необязательном втором аргументе, указывается основание системы счисления.

[2] Поскольку в JavaScript все числа с плавающей точкой, поддержка целых чисел ограничена 53 разрядами. Операция над большими целыми числами может привести к целочисленному переполнению.

Логические операторы

В Python логические (boolean) операторы обозначаются иначе, чем в JavaScript и C-подобных языках, но работают примерно так же. В таблице ниже они сравниваются:

```
Python     bool    True  False  not and or
JavaScript boolean true  false  !   &&  ||
```

Принятое в Python написание `True` и `False` JavaScript-разработчик может попытаться заменить привычным, и наоборот, но любая приличная подсветка синтаксиса, как и линтер кода, не пропустят такой ошибки.

Результатом логической операции как в Python, так и JavaScript не всегда являются логические true (истина) или false (ложь). Выражения `and`/`or` возвращают один из аргументов, который может быть как логического, так и другого типа. В Таблице 2-1 на примере Python показано, как работают операторы:

Таблица 2-1. *Логические операторы Python*

Операция	Результат
x or y	если x ложно, то y, иначе x
x and y	если x ложно, то x, иначе y
not x	если x ложно, то True, иначе False

Этим свойством можно иногда воспользоваться, чтобы присвоить значение переменной:

```
rocket_launch = True
(rocket_launch == True and 'All OK') or 'We have a problem!'
Out:
'All OK'

rocket_launch = False
(rocket_launch == True and 'All OK') or 'We have a problem!'
Out:
'We have a problem!'
```

Контейнеры данных: словари, объекты, списки и массивы

Объекты JavaScript можно использовать аналогично словарям (`dict`) Python, а массивы (array) JS — спискам (`list`) Python. В Python также есть кортежи

(tuple), которые представляют собой неизменяемую последовательность. Рассмотрим два примера:

```
# Python
d = {'name': 'Groucho', 'occupation': 'Ruler of Freedonia'}
l = ['Harpo', 'Groucho', 99]
t = ('an', 'immutable', 'container')
// JavaScript
d = {'name': 'Groucho', 'occupation': 'Ruler of Freedonia'}
l = ['Harpo', 'Groucho', 99]
```

Как вы видели в примерах 2-3 и 2-4, в Python ключи `dict` — это строки, обязательно заключенные в кавычки (хешируемый тип), а JavaScrip позволяет опускать кавычки, если имя свойства объекта является допустимым идентификатором (не содержит специальных символов: пробелов, дефисов и др.). В объектах `studentData` JS автоматически преобразует имя свойства `'name'` в строку.

Объявления данных о студентах выглядят примерно одинаково в обоих примерах и используются похожим образом. Главное отличие: несмотря на внешнее сходство со словарями `dict` в Python, JS-контейнеры в фигурных скобках (например `studentData`) являются объектами JS.

При визуализации данных в JS основным контейнером данных, как правило, выступают массивы объектов, и в таком случае объекты JS ведут себя подобно словарям Python. Доступ к элементам возможен по ключу или через точечную нотацию (удобно, если в ключе есть пробелы/дефисы, и он должен быть в кавычках):

```
let foo = {bar:3, baz:5};
foo.bar; // 3
foo['baz']; // 5, same as Python
```

Объекты Java-Script — это не только контейнеры для хранения данных, объекты — одна из основ языка (кроме таких примитивов, как строки и числа, практически все в JavaScript является объектами)[1]. Но в большинстве примеров визуализации в этой книге они используются как аналог словарей Python.

В таблице 2-2 сопоставляются основные операции для работы с массивами и списками.

[1] Это несколько усложняет работу со свойствами объектов. Подробнее см. в Stack Overflow. https://stackoverflow.com/questions/8312459/iterate-through-object-properties

Таблица 2-2. Списки (list) и массивы (array)

JavaScript array (a)	Python list (l)
`a.length`	`len(l)`
`a.push(item)`	`l.append(item)`
`a.pop()`	`l.pop()`
`a.shift()`	`l.pop(0)`
`a.unshift(item)`	`l.insert(0, item)`
`a.slice(start, end)`	`l[start: end]`
`a.splice(start, howMany, il, ...)`	`l[start: end] = [il, ...]`

Функции

В блоке B примеров 2-3 и 2-4 показано объявление функции. В Python для этого служит ключевое слово `def`:

```
def process_student_data(data, pass_threshold=60,
                         merit_threshold=75):
    """ Выполняет базовые статистические операции над данными
    студентов. """
    ...
```

JavaScript использует ключевое слово function:

```
function processStudentData(data, passThreshold=60,
                            meritThreshold=75){
...
}
```

Функции в обоих языках могут принимать произвольное число параметров. Блок кода функции в JS заключается в фигурные скобки {...}, а в Python обозначается двоеточием и отступами.

В примере ниже показан альтернативный способ определения функции в JS — *функциональное выражение* (function expression):

```
let processStudentData = function( ...){...}
```

Сейчас набирает популярность сокращенная форма записи:

```
let processStudentData = ( ...) => {...}
```

Различия настолько незначительны, что на данный момент не будем на них останавливаться[1].

Обработка параметров функций в Python более гибкая, чем в JS. Как вы могли заметить в `process_student_data` (см. блок B из примера 2-3), в Python можно устанавливать аргументы параметров по умолчанию, а в JavaScript параметру, которому при вызове функции не передается аргумент, по умолчанию присваивается значение *undefined*.

Итерирование: циклы for и их функциональные альтернативы

В блоках C примеров 2-3 и 2-4 мы видим первое ключевое различие между Python и JavaScript — циклы `for`.

Циклы `for` в Python просты, интуитивно понятны и эффективно перебирают элементы коллекций, таких как массивы и словари (`dict`). Подводный камень словаря `dict` — стандартная итерация выполняется по ключам, а не по элементам. Например:

```
foo = {'a':3, 'b':2}
for x in foo:
   print(x)
# outputs 'a' 'b'
```

Для перебора пар «ключ-значение» используется метод словарей `items`, например так:

```
for x in foo.items():
   print(x)
# возвращает кортежи с парами ключей и значений ('a', 3) ('b' 2)
```

Для удобства можно можно определить ключи-значения в выражении `for`. Например:

```
for key, value in foo.items():
```

Python-цикл `for` работает с любыми объектами, у которых реализован итератор, например вы можете проходить циклом по строкам файла:

[1] Любознательные могут познакомиться с неплохим разбором объявления функций и функциональных выражений в блоге Ангуса Кролла (Angus Croll).

```
for line in open('data.txt'):
   print(line)
```

По сравнению с Python цикл `for` в JS сложней для понимания, например:

```
for(let i in ['a', 'b', 'c']){
   console.log(i)
}
// outputs 1, 2, 3
```

Здесь `for .. in` возвращает не элементы массива, а их индексы. Кроме того, порядок возврата индексов не гарантирован, что может сбивать с толку разработчиков Python.

Переключение между циклами `for` Python и JS требует сосредоточенности. Однако в JS можно легко обойтись без циклов `for`. Я ими практически никогда не пользуюсь, поскольку в JS появилось несколько мощных функций, которые делают код более выразительным и меньше сбивают с толку питонистов. Немного освоившись с этими методами, вы поймете, что они вам необходимы[1].

В блоке C из примера 2-4 показан `forEach()` — один из современных *функциональных* методов массивов JavaScript[2]. Метод `forEach()` перебирает элементы массива, применяя к ним по очереди колбэк-функцию, определенную в качестве первого аргумента. Выразительные возможности функциональных методов map, filter и др. полностью реализуются при их объединении, но мы уже получили более чистую и изящную итерацию, лишенную прежней неуклюжести.

Колбэк-функция принимает несколько аргументов — текущий элемент массива, а также индекс и исходный массив (опционально):

```
data.forEach(function(currentValue, index){...})
```

До появления этих методов даже итерация по парам ключ-значение объекта была сложной. В отличие от `dict` в Python объекты JS могли наследовать свойства через цепочку прототипов, чтобы отфильтровать эти свойства, приходилось применять метод `hasOwnProperty`. Вам вполне может встретиться код наподобие следующего:

[1] Это одна из областей, где JS превосходит Python, и многие хотели бы иметь аналогичную функциональность в Python.

[2] Добавлены в ES5 и доступны во всех современных браузерах.

```
let obj = {a:3, b:2, c:4};
for (let prop in obj) {
   if( obj.hasOwnProperty( prop ) ) {
      console.log("o." + prop + " = " + obj[prop]);
   }
}
// out: o.a = 3, o.b = 2, o.c = 4
```

В то время, как JS имеет набор функциональных методов для итерации по массивам (`map`, `reduce`, `filter`, `every`, `sum`, `reduceRight`), для `object`, играющих роль «псевдословарей» их нет. К счастью, появилось несколько полезных дополнительных методов класса `Object`, которые заполняют этот пробел. Теперь можно выполнить итерацию по парам «ключ-значение» с помощью метода `Object.entries`:

```
let obj = {a:3, b:2, c:4};
for (const [key, value] of Object.entries(object1)) {
   console.log(`${key}: ${value}`); ❶
}
// возвращает: a: 3
//             b: 2 ...
```

❶ Обратите внимание на шаблон строки вывода переменных значений `${foo}`.

Условные операторы if, else, elif, switch

В секции C из примеров 2-3 и 2-4 показаны условные операторы Python и JavaScript в действии. Не считая скобок, которые используются JavaScript, конструкции этих операторов в обоих языках очень похожи. Единственное реальное различие — дополнительное ключевое слово `elif` в Python используется вместо `else if` в JS.

В отличие от большинства языков высокого уровня в Python нет конструкции `switch`, хотя в ней есть потребность. В JS этот оператор имеется, и представляет собой следующую конструкцию:

```
switch(expression){
   case value1:
      // выполняет, если expression === value1
      break; // необязательный оператор выхода из switch
```

```
    case value2:
      //...
    default:
      // выполняется, если каждый case ложь
}
```

Однако на радость питонистам в Python 3.10 появился мощный механизм сопоставления с шаблоном, который действует не только как аналог `switch`, но и предлагает множество других возможностей https://oreil.ly/5x76a. Переключение между различными case выглядит следующим образом:

```
for value in [value1, value2, value3]:
  match value:
    case value1:
      # do foo
    case value2:
      # do baa
    case value3:
      # do baz
```

Файловый ввод-вывод

Python предоставляет максимально простые и удобные средства для работы с файлами, в то время как в JavaScript, ориентированном на браузеры, полноценных аналогов нет:

```
# ЧТЕНИЕ ФАЙЛА
f = open("data.txt") # открывает файл в режиме чтения

for line in f: # итерирование по строкам файла
  print(line)

lines = f.readlines() # собирает все строки файла в один список
data = f.read() # возвращает  содержимое файла в виде единой строки

# ЗАПИСЬ В ФАЙЛ
f = open("data.txt", 'w')
# 'w' — запись, 'a' — добавление в конец файла
f.write("this will be written as a line to the file")
f.close() # явно закрывает файл
```

При открытии файла настоятельно рекомендуется использовать контекстный менеджер Python — конструкцию `with ... as`. Она гарантирует, что файл будет закрыт автоматически, и по сути является синтаксическим сахаром для блока `try ... except ... finally` Ниже показано, как открывать файл с использованием `with... as`:

```
with open("data.txt") as f:
    lines = f.readlines()
    ...
```

В JavaScript имеется метод `fetch` для извлечения данных из сети по URL, который позволяет работать с файлами. Чтобы получить набор данных с веб-сервера, в каталоге *static/data* выполните следующее:

```
fetch('/static/data/nobel_winners.json')
    .then(function(response) {
    console.log(response.json())
})
Out:
[{name: 'Albert Einstein', category: 'Physics'...},]
```

Подробная документация Fetch API доступна на сайте Mozilla.

Классы и прототипы

В этой теме есть некоторая путаница, поскольку основой для реализации объектно-ориентированного программирования (ООП) в JavaScript выбрали прототипы, а не традиционные классы. Питонистам, привыкшим к классам, потребуется немного времени на адаптацию, но, по моему опыту, вы освоитесь легко и быстро.

Когда я только начал изучать более современные языки типа C++, я увлекся ООП, в частности концепцией наследования на основе классов. Тогда полиморфизм был в тренде, и классы геометрических фигур включали подклассы прямоугольников и эллипсов, а они в свою очередь более узкие подклассы квадратов и кругов.

Я быстро понял, что четкое разделение классов, как в учебниках, редко встречается в практике программирования, и что попытки сбалансировать универсальные и специальные API чреваты трудностями. Я считаю, что композиция и примеси — более полезные концепции ООП, чем расширения через подклассы,

и предпочитаю им всем техники функционального программирования, особенно в JavaScript. Тем не менее, важно понимать различия между классами Python и прототипами JS, ведь чем лучше вы разберетесь в нюансах, тем качественнее сможете писать код[1].

Создавать и использовать классы Python довольно просто и интуитивно понятно, что характерно для большинства компонентов языка. Сейчас я рассматриваю классы как простой способ инкапсуляции данных с помощью удобного API и редко расширяю их дальше одного уровня наследования. Приведу простой пример:

```
class Citizen(object):

    def __init__(self, name, country): ❶
        self.name = name
        self.country = country
    def __str__(self): ❷
        return f'Citizen {self.name} from {self.country}'
    def print_details(self):
        print(f'Citizen {self.name} from {self.country}')

groucho = Citizen('Groucho M.', 'Freedonia') ❸
print(groucho) # or groucho.print_details()
# Вывод:
# Citizen Groucho M. from Freedonia
```

❶ Для работы с классами в Python есть ряд методов, окруженных двойными подчеркиваниями. Самый распространенный из них `__init__`, который вызывается при создании нового экземпляра класса. Каждый метод в классе начинается с явного аргумента `self` (не советую его переименовывать), который ссылается на текущий экземпляр класса. В данном случае мы устанавливаем с его помощью свойства имени и страны.

❷ Переопределим метод класса str, который используется при вызове функции `print`.

❸ Создаем новый экземпляр класса `Citizen` с начальными значениями имени и страны.

[1] Я рассказал другу, талантливому программисту, что мне приходится объяснять прототипы Python-разработчикам, а он ответил, что большинству JavaScript-программистов не помешали бы подсказки по классам. В этом есть немалая доля истины: многие JS-разработчики продуктивно используют прототипы «*классовым*» способом, избегая таким образом пограничных случаев.

Наследование классов в Python в основном соответствует классическому паттерну ООП. Реализуется оно просто и, видимо, поэтому широко используется. Создадим на основе класса `Citizen` класс для лауреата Нобелевской премии `Winner` с несколькими дополнительными свойствами:

```
class Winner(Citizen):

    def __init__(self, name, country, category, year):
        super().__init__(name, country) ❶
        self.category = category
        self.year = year

    def __str__(self):
        return 'Nobel winner %s from %s, category %s, year %s'\
        %(self.name, self.country, self.category,\
        str(self.year))

w = Winner('Albert E.', 'Switzerland', 'Physics', 1921)
w.print_details()
# Out:
# Nobel prizewinner Albert E. from Switzerland, category Physics,
# year 1921
```

❶ Мы повторно вызовем метод `__init__` родительского класса (он же суперкласс) `Citizen`, используя экземпляр класса `Winner` как `self`. Метод `super` расширяет дерево наследования, добавляя ветвь из первого аргумента, а второй передавая в качестве экземпляра методу экземпляра класса.

Лучшее из того, что я читал о различиях между прототипами JavaScript и классами, — это статья Реджинальда Брейтуэйта (Reginald Braithwaite) *OOP, JavaScript, and so-called Classes*. По-моему, следующая цитата лучше всего отражает различие между классами и прототипами:

> Разница между прототипом и классом примерно такая же, как и между моделью дома и его чертежом.

При создании экземпляра класса C++ или Python выполняется определенный шаблон: создается объект и при обходе дерева наследования вызывается цепочка конструкторов. Вы создаете совершенно новый экземпляр класса с нуля.

С прототипами JavaScript все иначе: у вас уже есть готовая модель дома (объекта) со всем комнатами (методами). Хотите обновить гостиную? Перекрасьте стены. Хотите новую оранжерею? Сделайте пристройку. Но вы не строите заново, а модифицируете существующий объект.

Теперь, когда мы теоретически подкованы и понимаем, что о наследовании объектов знать полезно, но мало применимо для визуализации, рассмотрим простой прототип объекта JavaScript на примере 2-5.

Пример 2-5. *Простой объект JavaScript*

```
let Citizen = function(name, country){ ❶
   this.name = name; ❷
   this.country = country;
};

Citizen.prototype = { ❸
   logDetails: function(){
      console.log(`Citizen ${this.name} from ${this.country}`);
   }
};

let c = new Citizen('Groucho M.', 'Freedonia'); ❹

c.logDetails();
Out:
Citizen Groucho M. from Freedonia

typeof(c) # object
```

❶ Эта функция становится инициализатором, когда ее вызывают через оператор `new`.

❷ Ключевое слово `this` — это неявная ссылка на *контекст вызова* функции. Пока оно ведет себя ожидаемо, напоминая `self` Python, но далее мы увидим существенное различие.

❸ Определенные здесь методы переопределят любые методы прототипов выше по цепочке наследования и будут унаследованы любыми объектами, производными от `Citizen`.

❹ Оператор `new` создает новый объект, устанавливает его прототип равным свойству prototype функции-конструктора `Citizen`, а затем вызывает конструктор `Citizen` для нового объекта.

В ES6 был добавлен синтаксический сахар для объявления классов JavaScript. По сути, это обертка, которая позволяет прототипному наследованию (см. пример 2-5) выглядеть более узнаваемо для программистов, работающих с объектно-ориентированными языками типа Java и C#. Думаю, что классы не нашли особого применения в JS-программировании для фронтенда, поскольку их потеснили переиспользуемые компоненты современных фреймворков (например: React, Vue, Svelte). Можно изменить `Citizen Object`, показанный в примере 2-5:

```
class Citizen {
   constructor(name, country) {
      this.name = name
      this.country = country
   }

   logDetails() {
      console.log(`Citizen ${this.name} from ${this.country}`)
   }
}
const c = new Citizen('Groucho M.', 'Freedonia')
```

Сравнение self и this

Может показаться, что `self` в Python и `this` в JavaScript по сути одно и то же, разве что `this` в отличие от `self`, не передается явно в качестве параметра методам класса. На самом деле `this` и `self` существенно различаются. Давайте рассмотрим наш класс `Citizen` на обоих языках.

В Python `self` — это переменная, представляющая экземпляр класса, которая передается в методы класса (переименовать ее можно, но не рекомендуется). А `this` — это ключевое слово в JavaScript, которое ссылается на объект, вызывающий метод. При этом вызывающий объект может отличаться от экземпляра объекта данного метода, и JavaScript предоставляет методы объекта Function для вызова, привязки и применения функции: `call`, `bind` и `apply` https://oreil.ly/ONAkj, чтобы управлять этим поведением.

Используем метод `call`, чтобы изменить вызывающий объект для метода `print_details`, и, следовательно, ссылку для `this`, которая используется в методе для получения имени гражданина:

```
let groucho = new Citizen('Groucho M.', 'Freedonia');
let harpo = new Citizen('Harpo M.', 'Freedonia');
```

```
groucho.logDetails.call(harpo);
// Out:
// "Citizen Harpo M. from Freedonia"
```

Таким образом, `this` в JavaScript — более гибкий посредник, чем `self` в Python, предоставляющий больше свободы, но требующий большего контроля за вызывающим контекстом. При создании объектов обязательно использовать `new`[1].

Я не случайно привел пример 2-5, демонстрирующий использование оператора `new` при создании экземпляров объектов JavaScript. Вам придется довольно часто его использовать, несмотря на некоторую синтаксическую громоздкость, особенно при реализации наследования. ES5 предоставил улучшенный способ создания объектов и работы с наследованием — метод `Object.create`. Я рекомендую использовать его в своем коде, но не удивлюсь, если `new` появится в каких-нибудь сторонних библиотеках.

Создадим в примере ниже `Citizen` и наследующий ему `Winner` с помощью `Object.create`. Подчеркну, что в JavaScript есть много способов это сделать, но в примере 2-6 я привел самый лучший из найденных мною, я всегда его использую.

Пример 2-6. *Прототипное наследование с помощью Object.create*

```
let Citizen = { ❶
   setCitizen: function(name, country){
      this.name = name;
      this.country = country;
      return this;
   },
   printDetails: function(){
      console.log('Citizen ' + this.name + ' from ',\
      + this.country);
   }
};
```

[1] Еще одна причина использовать директиву ES5 'use strict', которая не пропустит подобные ошибки.

```
let Winner = Object.create(Citizen);

Winner.setWinner = function(name, country, category, year){
   this.setCitizen(name, country);
   this.category = category;
   this.year = year;
   return this;
};

Winner.logDetails = function(){
   console.log('Nobel winner ' + this.name + ' from ' +
   this.country + ', category ' + this.category + ', year ' +
   this.year);
};

let albert = Object.create(Winner)
   .setWinner('Albert Einstein', 'Switzerland', 'Physics', 1921);

albert.logDetails();
// Out:
// Nobel winner Albert Einstein from Switzerland, category
// Physics, year 1921
```

❶ Теперь `Citizen` — скорее объект, чем функция-конструктор. Воспринимайте его как основу для новых объектов, таких как `Winner`.

Повторю: прототипное наследование JavaScript нечасто применяется при визуализации, особенно в контексте библиотеки D3, ориентированной на декларативные и функциональные шаблоны, где для отображения на веб-странице используются *сырые* неинкапсулированные данные.

Этим хитроумным сравнением классов и прототипов завершим наше обсуждение. Далее мы рассмотрим некоторые общие принципы создания визуализаций с использованием Python и JS.

Примеры различий

Знать, чем отличается синтаксис JS и Python очень важно, но, к счастью, сходства в их синтаксисе гораздо больше, чем различий. Основные конструкции императивного программирования: циклы, условные операторы, объявление

переменных и манипулирование ими в целом одинаковые. Это особенно заметно в области обработки и визуализации данных, где функции первого порядка в этих языках поддерживают общие идиомы.

Ниже представлен далеко не полный список паттернов и идиом Python и JavaScript, полезных для специалистов по визуализации данных. Где это возможно, приводится перевод с одного языка на другой.

Цепочка методов

Цепочка методов (*method chaining*) — идиома JavaScript, которая активно применяется в таких популярных JS-библиотеках, как jQuery и в D3. Она заключается в том, что каждый метод вызывается на объекте, который вернул предыдущий метод. С помощью точечной нотации вызовы методов объединяются в одно выражение:

```
let sel = d3.select('#viz')
   .attr('width', '600px') ❶
   .attr('height', '400px')
   .style('background', 'lightgray');
```

❶ Метод `attr` возвращает вызвавшую его selection библиотеки D3, которая затем используется для вызова следующего метода `attr`.

В Python цепочки методов встречаются редко. Ради простоты и читабельности кода PEP-8 рекомендует не использовать несколько операторов на одной строке.

Перебор списка

Когда нужно итерировать список с отслеживанием индекса каждого элемента, в Python удобно использовать встроенную функцию `enumerate`:

```
names = ['Alice', 'Bob', 'Carol']

for i, n in enumerate(names):
   print(f'{i}: {n}')
Out:
0: Alice
1: Bob
2: Carol
```

SKLADCHIK.ORG

Методы массивов в JavaScript, такие как `forEach` и функциональные `map`, `reduce` и `filter`, передают итерируемый элемент и его индекс в коллбэк-функцию:

```
let names = ['Alice', 'Bob', 'Carol'];

names.forEach(function(n, i){
   console.log(i + ': ' + n);
});
Out:
0: Alice
1: Bob
2: Carol
```

Распаковка кортежей

Одним из интересных приемов Python является операция распаковки кортежа для обмена значениями переменных:

```
(a, b) = (b, a)
```

Примечание: скобки необязательны. Этот прием подходит и для более практических задач, например, позволяет не создавать временные переменные при вычислении чисел последовательности Фибоначчи:

```
def fibonacci(n):
   x, y = 0, 1
   for i in range(n):
      print(x)
      x, y = y, x + y
# fibonacci(6) -> 0, 1, 1, 2, 3, 5
```

Чтобы проигнорировать некоторые переменные при распаковке, используйте подчеркивание:

```
winner = 'Albert Einstein', 'Physics', 1921, 'Swiss'

name, _, _, nationality = winner ❶
print(f'{name}, {nationality}')
# Albert Einstein, Swiss
```

❶ В Python 3 есть оператор *, с помощью которого мы могли бы распаковать переменные таким образом: `name, *_, nationality = winner`.

Язык JavaScript быстро адаптируется, и в 2015 г. в нем появились возможности эффективной деструктуризации объектов. *Оператор spread* (`...`) позволяет компактно записывать манипуляции с переменными:

```
let a, b, rem ❶
[a, b] = [1, 2]
// обмен значений двух переменных
[a, b] = [b, a]
// использование spread-оператора
[a, b, ...rem] = [1, 2, 3, 4, 5, 6,] // rem = [3, 4, 5, 6]
```

❶ В отличие от Python необходимо предварительно объявить все переменные, которые будут использоваться.

Коллекции

Один из самых полезных встроенных модулей Python — `collections`. Он содержит специализированные контейнеры типов данных, дополняющих стандартный набор коллекций в Python. В него, среди прочих, входят: класс `deque`, который предоставляет двустороннюю очередь, позволяющую быстро добавлять и извлекать элементы; `OrderedDict`, который запоминает порядок вставки элементов; словарь `defaultdict`, предоставляющий фабричную функцию для словаря со значениями по умолчанию; контейнер `Counter`, предназначенный для подсчета количества вхождений неизменяемых элементов в последовательности. Я часто использую последние три компонента. Ниже приведу несколько примеров:

```
from collections import Counter, defaultdict, OrderedDict

items = ['F', 'C', 'C', 'A', 'B', 'A', 'C', 'E', 'F']

cntr = Counter(items)
print(cntr)
cntr['C'] -=1
print(cntr)
Out:
Counter({'C': 3, 'A': 2, 'F': 2, 'B': 1, 'E': 1})
```

```
Counter({'A': 2, 'C': 2, 'F': 2, 'B': 1, 'E': 1})

d = defaultdict(int) ❶

for item in items:
   d[item] += 1 ❷

d
Out:
defaultdict(<type 'int'>, {'A': 2, 'C': 3, 'B': 1, 'E': 1, 'F': 2})
OrderedDict(sorted(d.items(), key=lambda i: i[1])) ❸
Out:
OrderedDict([('B', 1), ('E', 1), ('A', 2), ('F', 2), ('C', 3)]) ❹
```

❶ Устанавливает словарь с целочисленными значениями по умолчанию, равными нулю.

❷ Если элемент отсутствует, к значению по умолчанию равному 0 прибавляется 1.

❸ Получает список элементов словаря `d` в виде кортежей из пар ключ-значение, сортирует по целочисленному значению и создает `OrderedDict` с отсортированным списком кортежей.

❹ `OrderedDict` запоминает порядок, в котором в него добавляются элементы.

Подробнее о модуле `collections` см. в документации Python.

Некоторые возможности модуля `collections` можно воспроизвести с помощью классических и удобных JS-библиотек: Underscore (или функционально близкой ей Lodash[1]). Эти библиотеки предоставляют набор утилит функционального программирования. Кратко рассмотрим их.

Underscore

Underscore — самая популярная JS-библиотека после вездесущей jQuery — предлагает ряд утилит функционального программирования, полезных для визуализации данных. Самый просто способ начать работу с Underscore — подключить ее через CDN (содержимое кэшируется браузером, повышая эффективность работы с распространенными библиотеками). Пример:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/
            underscore.js/1.13.1/underscore-min.js"></script>
```

[1] Я предпочитаю ее из-за лучшей производительности.

В Underscore содержится множество полезных функций. Например, метод `countBy`, выполняющий ту же задачу, что и Counter из модуля `collections` в Python:

```
let items = ['F', 'C', 'C', 'A', 'B', 'A', 'C', 'E', 'F'];

_.countBy(items) ❶
Out:
Object {F: 2, C: 3, A: 2, B: 1, E: 1}
```

❶ Понятно, почему библиотеку Underscore так назвали — Underscore переводится с английского как «нижнее подчеркивание».

Из-за появления в современном JavaScript итератора `forEach` и нативных функциональных методов (`map`, `reduce`, `filter`) для работы с массивами Underscore перестала быть незаменимой, но у нее осталось несколько замечательных утилит, которые дополняют чистый JS. С помощью небольших цепочек можно писать лаконичный и мощный код. Лично для меня библиотека Underscore стала первым шагом в функциональное программирование на JavaScript, и ее идиомы до сих пор актуальны. Перечень утилит Underscore см. на сайте Underscore.js.

Давайте применим Underscore для задачи посложнее:

```
journeys = [
   {period:'morning', times:[44, 34, 56, 31]},
   {period:'evening', times:[35, 33],},
   {period:'morning', times:[33, 29, 35, 41]},
   {period:'evening', times:[24, 45, 27]},
   {period:'morning', times:[18, 23, 28]}
];

let groups = _.groupBy(journeys, 'period');
let mTimes = _.pluck(groups['morning'], 'times');
mTimes = _.flatten(mTimes); ❶
let average = function(l){
   let sum = _.reduce(l, function(a,b){return a+b},0);
   return sum/l.length;
};
console.log('Average morning time is ' + average(mTimes));
Out:
Average morning time is 33.81818181818182
```

❶ Многомерный массив показателей утреннего времени (`[[44, 34, 56, 31], [33...]]`) нужно «*схлопнуть*» в одномерный массив чисел.

Функциональные методы массивов и генераторы списков (List Comprehensions)

После добавления в ES5 функциональных методов массивов JavaScript я стал реже использовать Underscore. К тому же, я совсем перестал использовать циклы `for`, и это хорошо, поскольку в JS они уродливы.

Как только привыкнешь к функциональной обработке массивов, не хочется возвращаться к старым способам. Функциональные методы массивов в сочетании с анонимными функциями делают JS гибким и выразительным языком программирования. При работе с массивами также вполне естественно применять цепочку методов. Ниже рассмотрим чисто умозрительный пример:

```
let nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

let sum = nums.filter(x => x%2) ❶
   .map(x => x * x) ❷
   .reduce((total, current) => total + current, 0); ❸

console.log('Sum of the odd squares is ' + sum);
```

❶ Отфильтровывает нечетные числа для массива (т. е. возвращает true, если остаток от деления на 2 по модулю (`%`) равен 1).

❷ Метод map создает новый массив, применяя функцию к каждому элементу получившегося массива (т. е. `[1, 3, 5...]` → `[1, 9, 25...]`).

❸ `reduce` последовательно обрабатывает новый массив, объединяя (в данном случае, суммируя) значение (`total`) и значение текущего элемента (`current`). Начальным значением первого аргумента (`total`) по умолчанию является 0, но здесь мы его указали явно как второй аргумент.

Теперь этот же пример реализуем довольно легко и просто с помощью генераторов списков Python:

```
nums = range(10) ❶

odd_squares = [x * x for x in nums if x%2] ❷
sum(odd_squares) ❸
```

```
Out:
165
```

❶ В Python есть встроенная функция `range`, которая принимает стартовое и финальное значения диапазона и шаг. Например: range(2, 8, 2) вернет (2, 4, 6).
❷ Условный оператор `if` проверяет `x` на нечетность, и все числа, прошедшие фильтр, возводятся в квадрат и вставляются в список.
❸ Встроенная функция `sum` в Python часто используется для суммирования элементов.



Генераторы списков (List Comprehensions) в Python позволяют использовать рекурсивные управляющие структуры, такие как вложенное выражение `for`/`if`. Хотя такой прием делает запись компактной, я не рекомендую его использовать — снижается читабельность кода. Вы рискуете написать сложный для восприятия код, ведь даже простые генераторы списка нельзя назвать интуитивно понятными.

Генераторы списков Python хороши для базовых операций фильтрации и сопоставления, хотя уступают в гибкости анонимным функциям JavaScript (полноценным, со своей областью видимости, блоками управления, обработкой исключений и т. д.), но против анонимных функций также есть возражения. Их нельзя использовать повторно, сложно отслеживать и отлаживать из-за отсутствия имени. Почему нужно избегать анонимных функций см. на сайте Ultimate Courses Однако в библиотеках вроде D3 замена небольших одноразовых анонимных функций на именованные функции для установки атрибутов и свойств DOM было бы обременительной и добавила бы boilerplate-кода.

В следующем подразделе мы рассмотрим лямбда-выражения Python, но для полноценной функциональной обработки в Python (вынужденно) и в JavaScript (чтобы попрактиковаться) мы можем использовать именованные функции, чтобы лучше контролировать выполнение кода. В нашем простом примере по возведению нечетных чисел в квадрат именованные функции — это простой трюк, но заметьте, что благодаря им генератор списка сразу становится удобнее читать, и чем сложнее функция, тем это важнее:

```
items = [1, 2, 3, 4, 5]

def is_odd(x):
    return x%2
```

```
def sq(x):
   return x * x

sum([sq(x) for x in items if is_odd(x)])
```

Аналогичный подход в JavaScript тоже может улучшить читабельность и содействовать выполнению принципа DRY[1]:

```
let isOdd = function(x){ return x%2; };

sum = l.filter(isOdd)
...
```

Методы map, reduce, filter и лямбда-функции Python

В Python нет анонимных функций, но есть безымянные *лямбда-функции*, принимающие аргументы. В них нет наворотов анонимных функций JavaScript, но они являются эффективным дополнением к набору функций Python, особенно в комбинации с функциональными методами.



У встроенных функциональных инструментов Python (методов `map`, `reduce`, `filter` и лямбда-функций) непростая история. Создатель Python даже хотел удалить их из языка, но его решение встретило всеобщее неодобрение. Хорошо, что эти инструменты остались, тем более что наблюдается тенденция возврата к функциональному программированию. Эти методы не идеальны, но лучше, чем ничего. К тому же они позволяют JavaScript-разработчикам применять навыки, приобретенные в JS.

Лямбда-функции Python принимают неограниченное число параметров и возвращают результат вычислений. Функциональный блок определяется двоеточием. В целом лямбда-функция напоминает обычную функцию, но урезанную до необходимого минимума и не имеющую явного оператора возврата. В примере ниже показано несколько лямбда-функций, применяемых в функциональном программировании:

[1] DRY (Don't Repeat Yourself, «Не повторяйся!») — принцип разработки ПО.

```
from functools import reduce # если используется Python 3+

nums = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

odds = filter(lambda x: x % 2, nums)
odds_sq = map(lambda x: x * x, odds)
reduce(lambda x, y: x + y, odds_sq) ❶
Out:
165
```

❶ Здесь метод `reduce` передает два аргумента в лямбда-функцию, которая возвращает результат выражения, указанного после двоеточия.

Замыкание JavaScript и паттерн «Модуль»

Замыкание (closure) в JavaScript — это вложенная функция, у которой есть доступ к переменным, объявленным в области видимости внешней функции (но не глобальной) *даже после того*, как внешняя функция завершила работу. Замыкания являются особенностью JS и позволяют реализовать ряд очень полезных паттернов программирования.

Пожалуй, самое распространенное использование замыканий, которое мы уже рассматривали в нашем паттерне «модуль» (пример 2-2), — создание ограниченного API, предоставляющего доступ к приватным переменным.

Простым примером замыкания является небольшой счетчик ниже:

```
function Counter(inc) {
   let count = 0;
   let add = function() { ❶
      count += inc;
      console.log('Current count: ' + count);
   }
   return add;
}

let inc2 = Counter(2); ❷
inc2(); ❸
Out:
Current count: 2
inc2();
```

```
Out:
Current count: 4
```

❶ Функция `add` получает доступ к приватным, находящимся вне текущей области видимости, переменным `count` и `inc`.
❷ Возвращает функцию `add` со скрытыми переменными, `count` (0) и `inc` (2).
❸ Вызов `inc2` вызывает `add`, обновляя скрытую переменную `count`.

Мы можем расширить `Counter`, чтобы добавить небольшой API. Эта техника лежит в основе модулей JS и многих простых библиотек, основанных на классическом скриптовом JavaScript[1]. По сути, он выборочно раскрывает публичные методы, скрывая при этом приватные методы и переменные, что обычно считается хорошей практикой программирования:

```
function Counter(inc) {
   let count = 0
   let api = {}
   api.add = function() {
      count += inc
      console.log('Current count: ' + count);
   }
   api.sub = function() {
      count -= inc
      console.log('Current count: ' + count)
   }
   api.reset = function() {
      count = 0;
      console.log('Count reset to 0')
   }
   return api
}

cntr = Counter(3);
cntr.add() // Current count: 3
cntr.add() // Current count: 6
cntr.sub() // Current count: 3
cntr.reset() // Count reset to 0
```

[1] В современный JavaScript реализованы модули, которые могут импортировать и экспортировать инкапсулированные переменные. Однако использование модулей требует этапа сборки кода перед запуском в браузере, в нашем случае это избыточно.

Замыкания JavaScript часто применяются в самых разных сценариях. Я рекомендую разобраться с ними, поскольку вы не раз на них наткнетесь, исследуя чужой код. Три статьи с множеством хороших примеров использования замыканий:

— Введение от Mozilla (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Closures)
— *JavaScript Module Pattern: In-Depth* Бена Черри (Ben Cherry)
— *Use Cases for JavaScript Closures* Юрия Зайцева

Python также поддерживает замыкания, но они используются не реже, чем в JavaScript, возможно, из-за пары странностей поведения, которые преодолимы, но код становится несколько неуклюжим. Я покажу в примере 2-7 переписанный на Python счетчик JavaScript из предыдущего примера:

Пример 2-7. *Счетчик-замыкание на Python, первая попытка*

```
def get_counter(inc):
    count = 0
    def add():
        count += inc
        print('Current count: ' + str(count))
    return add
```

Если вы создадите счетчик с помощью функции `get_counter` (пример 2-7) и запустите его, то получите ошибку `UnboundLocalError`:

```
cntr = get_counter(2)
cntr()
Out:
...
UnboundLocalError: local variable 'count' referenced before
assignment
```

Интересно, что мы можем прочитать значение `count` внутри функции `add` (закомментируйте строку `count += inc`, чтобы это проверить), но попытки изменить его приводят к ошибке. Это происходит, поскольку Python предполагает, что переменная, которой присвоено значение в теле функции, находится

SKLADCHIK.ORG
в области видимости этой функции. Ошибка генерируется, потому что локальной переменной `count` для функции `add` не существует.

В Python 3 ошибку в примере 2-7 можно обойти с помощью ключевого слова `nonlocal`, которое дает понять интерпретатору, что `count` находится в нелокальной области видимости:

```
...
def add():
    nonlocal count
    count += inc
...
```

Если вы используете Python 2+ (желательно обновить версию), поможет небольшой танец с бубном вокруг словаря, чтобы изменять закрытые переменные:

```
def get_counter(inc):
    vars = {'count': 0}
    def add():
        vars['count'] += inc
        print('Current count: ' + str(vars['count']))
    return add
```

Прием работает, потому что мы не присваиваем новое значение `vars`, а изменяем существующий контейнер, что полностью валидно, даже когда он находится за пределами локальной области видимости.

Как видите, JavaScript-разработчики могут использовать свои навыки работы с замыканиями (с небольшими поправками) в Python. Сценарии использования замыканий примерно те же, но Python богатый язык с множеством полезных библиотек, так что для решения проблем есть много возможностей помимо замыканий. Пожалуй, чаще всего замыкания в Python используются в декораторах.

Декоратор — это обертка, расширяющая возможности функции без изменения ее исходного кода. Декораторы — относительно сложная концепция, но вы найдете простое и понятное введение в эту тему на сайте The Code Ship.

На этом я завершаю рассказ о небольшой выборке из паттернов и приемов, которые часто использую при визуализации данных. Несомненно, у вас появятся собственные предпочтения, но надеюсь, моя подборка поможет вам.

Шпаргалка

На рисунках с 2.2 по 2.7 приводится простое справочное пособие: набор шпаргалок, где сопоставляются основные операции языков Python и JavaScript.

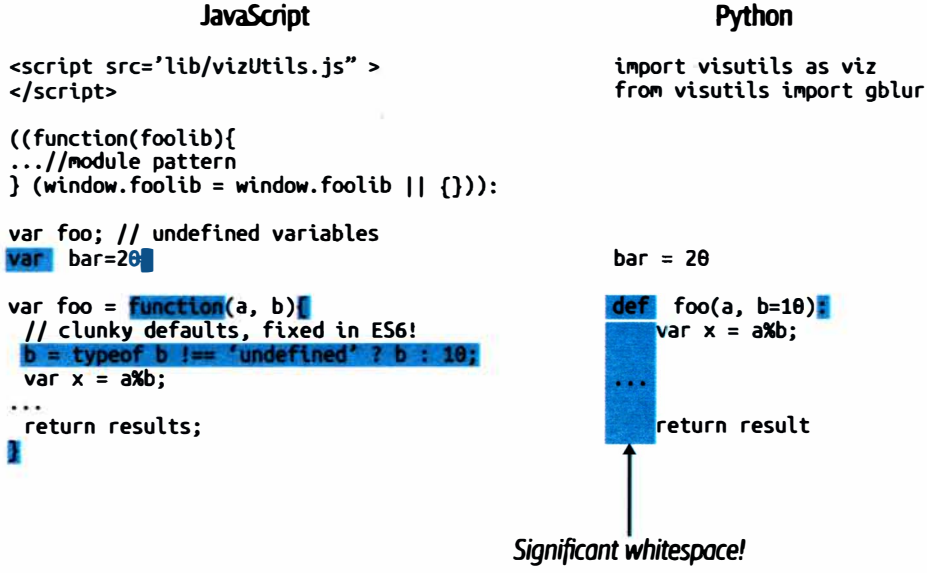


Рис. 2.2. Базовый синтаксис



Рис. 2.3. Логические операторы

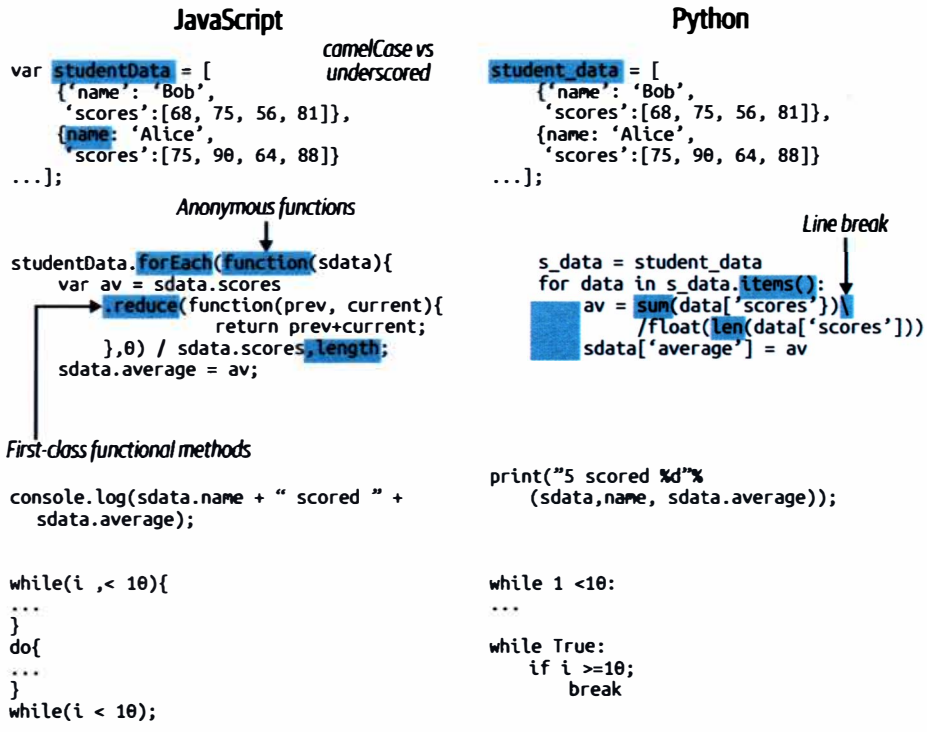


Рис. 2.4. Циклы и итерации

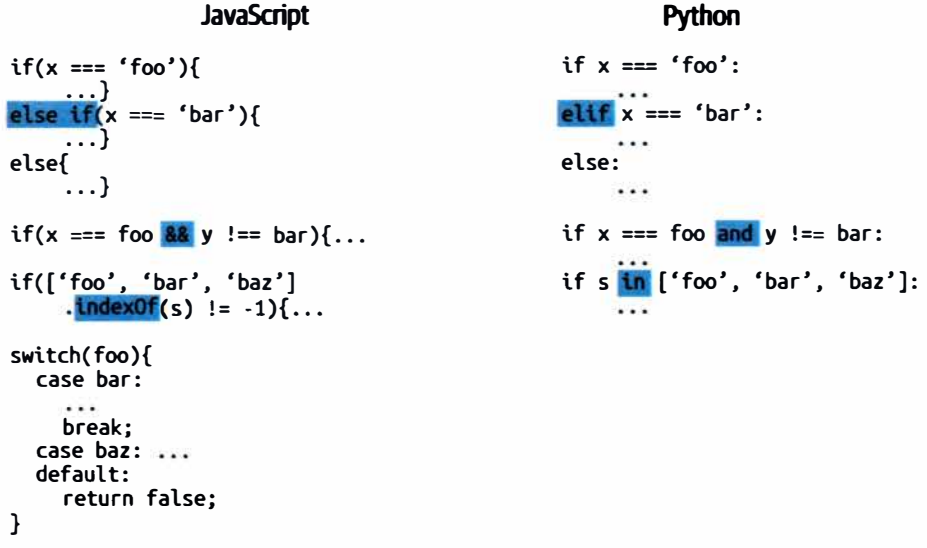


Рис. 2.5. Условные операторы

JavaScript

```
va1 l = [1, 2, 3, 4]:
l.push('foo'): // [...4, 'foo']
l.pop(); // 'foo', l=[..., 4]
l.slice(1,3) // [2,3]
l.slice(-3, -1) // [2, 3]

l.map(function(o)[ return o*o;})
// [1, 4, 9, 16]

d = {1:1, b:2, c:3};
d.a === d[1'] // 1
d.z // undefined

// OLD BROWSERS
for(key in d){
  if(d.hasOwnProperty(key){
    var item = d[key]:

// NEW AND BETTER
Object.keys(d).forEach(key, i){
  var item - d[key];
```

Python

```
l = [1, 2, 3, 4]
l.append('foo') # [...4, 'foo']
l.pop() # 'foo', l=[..., 4]
l[1,3] # [2,3]
l[-3, -1] # [2, 3]
1[0:4:2] # [1, 3] (stride of 2)

[o*o for o in l]
// [1, 4, 9, 16]

d = {'a':1, 'b':2, 'c':3};
d['a'] # 1
d.get['z'] # NoneType
d['z'] # KeyError!

for key, value in d.items(): ...
for key in d:
for value in d.values(): ...
```

Рис. 2.6. Контейнеры

JavaScript

```
var Foo = {
  initFoo: function(bar){
    this.bar = bar;
    return this:
  }
};

var Baz = Object.create(Foo);

Baz.initBaz = function(bar, qux){
  this.initFoo(bar);
  this.qux - qux;
  return this;
};

var baz = Object.create(Baz)
             .initBaz('answer'. 42);
```

Python

```
class Foo(object):
    def __init__(self, bar):
        self.bar = bar

class Baz(Foo):
    def __init__(self, bar, qux):
        self.qux = qux

baz = Baz('answer', 42)
baz.bar # 'answer'
```

Рис. 2.7. Классы и прототипы

Резюме

Надеюсь, мне удалось показать в этой главе, что в синтаксисе JavaScript и Python много общего и что большинство распространенных идиом и паттернов одного языка можно без особых усилий воспроизвести средствами другого. Ключевые концепции программирования — итерации, условные операторы и базовые

манипуляции с данными — в обоих языках такие же простые для понимания, как и перевод функций. Если вы программируете на одном из языков достаточно уверенно, то освоить другой несложно. В том и прелесть этих простых скриптовых языков, построенных на одних и тех же принципах.

Я привел ряд паттернов, идиом и приемов, которые сам часто использую при создании визуализаций. Наверняка, личные предпочтения как-то влияли, но я старался сделать правильный выбор.

Эта глава частично учебное пособие, частично справочник, к которому можно обращаться при изучении следующих глав. Все, что не охвачено в этой главе, мы разберем по мере необходимости.

Чтение и запись данных с помощью Python

Один из ключевых навыков специалиста по визуализации данных — это умение обращаться с данными. Независимо, в чем хранятся данные — в базе данных, CSV-файле или в каком-то экзотическом формате — вы должны легко их читать, конвертировать и записывать, при необходимости в более удобный формат. Одна из самых замечательных особенностей Python — простота манипуляций данными. Глава 3 познакомит вас с этим важнейшим звеном нашего тулчейна визуализации.

Эта глава сочетает в себе элементы обучения и справочного пособия, в последующих главах будут отсылки к ней. Если вы знакомы с основами чтения и записи данных в Python, можете просто освежить знания, выбирая отдельные фрагменты главы.

Просто ли это?

Когда я только начинал программировать (на низкоуровневых языках типа C), мне было неудобно манипулировать данными. Для чтения из файлов и записи в них приходилось составлять раздражающую смесь из boilerplate-кода, написанных вручную импровизаций и тому подобного. Чтение из баз данных было ничуть не легче, а уж про сериализацию данных вспоминать страшно. Python стал для меня глотком свежего воздуха. Скоростью он не может похвастаться, но открывать файл в нем проще простого:

```
file = open('data.txt')
```

Уже тогда в Python чтение и запись файлов были интуитивно понятными, а мощные методы обработки строк значительно облегчали парсинг данных в этих файлах. Потрясающий модуль Pickle мог сериализовать практически любой объект Python.

Со временем в стандартную библиотеку Python добавились надежные продуманные модули, благодаря которым очень просто работать с файлами CSV и JSON, стандартом при созданий визуализаций для браузера. Для работы с SQL DB также есть несколько замечательных библиотек, например SQLAlchemy — инструмент, который я от души рекомендую. Базы данных NoSQL также хорошо поддерживаются. MongoDB — самая популярная из подобных баз данных. Библиотека Python PyMongo, которую мы рассмотрим далее в этой главе, позволяет взаимодействовать с этой базой данных относительно легко.

Передача данных

Хороший способ продемонстрировать использование ключевых библиотек хранения данных — это передать между ними один пакет данных, считывая и записывая его. Так мы увидим в действии основные форматы и базы данных, которые используются в процессе визуализации.

Мы будем передавать список объектов-словарей, которые чаще всего используются при создании визуализаций для браузера (см. пример 3-1). Этот набор данных передается в браузер в формате JSON, который, как мы увидим, легко конвертируется из словаря Python.

Пример 3-1. *Наш целевой список объектов данных (словарей)*

```
nobel_winners = [
    {'category': 'Physics',
     'name': 'Albert Einstein',
     'nationality': 'Swiss',
     'gender': 'male',
     'year': 1921},
    {'category': 'Physics',
     'name': 'Paul Dirac',
     'nationality': 'British',
     'gender': 'male',
     'year': 1933},
    {'category': 'Chemistry',
     'name': 'Marie Curie',
     'nationality': 'Polish',
     'gender': 'female',
     'year': 1911}
]
```

Чтобы продемонстрировать открытие/чтение и запись системных файлов, создадим CSV-файл из списка Python (пример 3-1).

Здесь и далее предполагается, что вы находитесь в рабочем (корневом) каталоге, где лежит подкаталог *data*. Запустить код можно из интерпретатора или файла Python.

Работа с системными файлами

В этом разделе мы создадим CSV-файл из списка словарей Python (пример 3-1). Обычно для этого используют встроенный модуль `csv`, но мы его разберем чуть позже, а сейчас просто рассмотрим базовые манипуляции с файлами Python.

Для начала откроем новый файл, используя `w` как второй аргумент, который показывает, что мы будем записывать данные в файл.

```
f = open('data/nobel_winners.csv', 'w')
```

Теперь запишем в CSV-файл данные из словаря `nobel_winners` (пример 3-1):

```
cols = nobel_winners[0].keys() ❶
cols = sorted(cols) ❷

with open('data/nobel_winners.csv', 'w') as f: ❸
    f.write(','.join(cols) + '\n') ❹

    for o in nobel_winners:
        row = [str(o[col]) for col in cols] ❺
        f.write(','.join(row) + '\n')
```

❶ Получает имена столбцов данных из ключей первого объекта (т. е., `['category', 'name', ...]`).

❷ Сортирует столбцы по алфавиту.

❸ Оператор Python `with` гарантирует, что файл будет закрыт при выходе из блока или при возникновении исключений.

❹ Метод `join` объединяет строки из списка (здесь `cols`) в одну строку (т. е."category, name,…").

❺ Создаем список, используя ключи столбцов объектов в `nobel_winners`.

Проверим содержимое созданного CSV-файла с помощью Python:

```
with open('data/nobel_winners.csv') as f:
    for line in f.readlines():
        print(line)

Out:
category,name,nationality,gender,year
Physics,Albert Einstein,Swiss,male,1921
Physics,Paul Dirac,British,male,1933
Chemistry,Marie Curie,Polish,female,1911
```

Как показывает вывод данных выше, CSV-файл сформирован нормально. Теперь используем встроенный модуль `csv`, чтобы сначала прочитать, а затем создать CSV-файл.

CSV, TSV и табличные форматы данных

Формат CSV (Comma-Separated Values), использующий в качестве разделителей запятые, и близкий ему TSV (Tab Separated Values), где разделители — символы табуляции, — вероятно, самые распространенные файловые форматы данных, с которыми как специалист по визуализации данных вы будете иметь дело. Умение читать и записывать CSV-файлы, а также их вариации либо с разделителями в виде вертикальной черты или точки с запятой, либо использующие ` вместо стандартных двойных кавычек — основополагающий навык. Модуль `csv` Python возьмет на себя практически всю эту рутинную работу. Проверим, как он работает с чтением и записью данных о лауреатах Нобелевской премии `nobel_winners`:

```
nobel_winners = [
    {'category': 'Physics',
     'name': 'Albert Einstein',
     'nationality': 'Swiss',
     'gender': 'male',
     'year': 1921},
     ...
]
```

Записать в CSV-файл данные `nobel_winners` (см. пример 3-1) очень просто. Класс `DictWriter` из модуля `csv` преобразует словари в строки CSV. Нам

нужно лишь явно записать заголовок в CSV-файл, используя ключи словарей как поля (т. е. category, name, nationality, gender):

```
import csv

with open('data/nobel_winners.csv', 'w') as f:
    fieldnames = nobel_winners[0].keys() ❶
    fieldnames = sorted(fieldnames) ❷
    writer = csv.DictWriter(f, fieldnames=fieldnames)
    writer.writeheader() ❸
    for w in nobel_winners:
        writer.writerow(w)
```

❶ Указываем для writer, какие поля `fieldnames` использовать (в данном случае, ключи `'category'`, `'name'` и др.).

❷ Сортируем поля заголовка CSV по алфавиту для удобства чтения.

❸ Записываем заголовок в CSV-файл (category, name, …).

Пожалуй, CSV-файлы чаще приходится читать, чем записывать[1]. Давайте снова прочитаем файл *nobel_winners.csv*, который только что записали.

Если вы будете использовать модуль `csv` только для чтения файла, пара строк кода создаст удобный итератор, который представит строки CSV-файла в виде списков строк:

```
with open('data/nobel_winners.csv') as f:
    reader = csv.reader(f)
    for row in reader: ❶
        print(row)

Out:
['category', 'name', 'nationality', 'gender', 'year']
['Physics', 'Albert Einstein', 'Swiss', 'male', '1921']
['Physics', 'Paul Dirac', 'British', 'male', '1933']
['Chemistry', 'Marie Curie', 'Polish', 'female', '1911']
```

❶ Итерирует объект `reader`, перебирая строки файла.

Заметьте, что при считывании чисел мы получаем строковый тип данных. Если нужно их обрабатывать как числа, преобразуйте соответствующие столбцы, в данном случае, годы, в тип int.

[1] Предпочтительнее использовать JSON, а не CSV.

Более удобный способ использования CSV-данных — преобразование строк в словари Python. Нас это полностью устраивает, наша цель — получить список словарей. `DictReader` в `csv` предназначен именно для этого:

```
import csv
with open('data/nobel_winners.csv') as f:
    reader = csv.DictReader(f)
    nobel_winners = list(reader) ❶

nobel_winners

Out:
[OrderedDict([('category', 'Physics'),
              ('name', 'Albert Einstein'),
              ('nationality', 'Swiss'),
              ('gender', 'male'),
              ('year', '1921')]),
OrderedDict([('category', 'Physics'),
              ('name', 'Paul Dirac'),
              ('nationality', 'British'),
              ... ])]
```

❶ Вставляет все элементы `reader` в список.

Теперь нам остается преобразовать, как показано ниже, атрибуты year из словарей в целые числа, чтобы `nobel_winners` соответствовали целевым данным главы (пример 3-1):

```
for w in nobel_winners:
    w['year'] = int(w['year'])
```

Мы также можем получить даты, применив к колонке year встроенный модуль `datetime`:

```
from datetime import datetime

dt = datetime.strptime('1947', '%Y')
dt
# datetime.datetime(1947, 1, 1, 0, 0)
```

Csv-ридеры не определяют типы данных из файла, интерпретируя все как строку. Python-библиотека pandas, виртуозно работающая с данными, пытается определять тип данных в столбцах, и обычно ей это удается. Мы убедимся в этом далее, в главах, посвященных pandas.

В модуле csv есть несколько полезных аргументов для анализа элементов семейства CSV:

`dialect`
: По умолчанию имеет значение `'excel'`. Определяет набор параметров, специфичных для диалекта. Иногда в качестве альтернативы используется `excel-tab`.

`delimiter`
: Разделителями в файлах обычно служат запятые, но вместо них могут использоваться `|`, `:` или пробел `' '`

`quotechar`
: По умолчанию используются двойные кавычки, но вместо них могут быть `|` или `` ` ``.

Полную информацию о параметрах модуля csv см. в онлайн-документации Python.

После успешной записи и чтения наших целевых данных с помощью модуля csv передадим полученный словарь nobel_winners в модуль json.

JSON

Здесь мы будем читать и записывать данные nobel_winners с помощью модуля Python json. Припомним, какие данные мы используем:

```
nobel_winners = [
   {'category': 'Physics',
    'name': 'Albert Einstein',
    'nationality': 'Swiss',
    'gender': 'male',
    'year': 1921},
    ...
]
```

Для таких примитивов, как строки, целые числа и числа с плавающей точкой, словари Python легко сохраняются или, в терминах JSON, *выгружаются (dumped)* в файлы JSON с помощью модуля `json`. Метод `dump` принимает контейнер Python и путь к файлу, куда будут сохраняться данные из контейнера:

```
import json

with open('data/nobel_winners.json', 'w') as f:
      json.dump(nobel_winners, f)

open('data/nobel_winners.json').read()

Out: '[{"category": "Physics", "name": "Albert Einstein",
"gender": "male", "year": 1921,
"nationality": "Swiss"}, {"category": "Physics",
"nationality": "British", "year": 1933, "name": "Paul Dirac",
"gender": "male"}, {"category": "Chemistry", "nationality":
"Polish", "year": 1911, "name": "Marie Curie", "gender":
"female"}]'
```

Читать (или загружать) файл JSON так же просто. Просто передаем открытый JSON-файл в метод `load` модуля `json`:

```
import json
with open('data/nobel_winners.json') as f:
      nobel_winners = json.load(f)

nobel_winners
Out:
[{'category': 'Physics',
  'name': 'Albert Einstein',
  'nationality': 'Swiss',
  'gender': 'male',
  'year': 1921}, ❶
... }]
```

❶ Заметьте, что в отличие от преобразования CSV-файла, целочисленный тип для колонки year сохраняется.

В модуле `json` есть методы для работы со строками: `loads` загружает строки JSON в контейнеры Python, а `dumps` выгружает контейнеры Python в строки JSON.

Работа с датами и временем

Попытка выгрузить объект типа `datetime` в `json` приводит к ошибке `TypeError`:

```
from datetime import datetime

json.dumps(datetime.now())
Out:
...
TypeError: datetime.datetime(2021, 9, 13, 10, 25, 52, 586792)
is not JSON serializable
```

С сериализацией простых типов данных, таких как строки или числа, декодеры и энкодеры `json` справляются отлично. Но специализированные типы данных, например даты, придется кодировать и декодировать вручную. Это не так уж сложно и быстро становится привычной задачей. Для начала рассмотрим преобразование объектов datetime в строки JSON.

В Python проще всего преобразовать данные типа `datetime`, создав собственный энкодер (как в примере 3-2), который передается методу `json.dumps` в качестве аргумента `cls`. Такой энкодер применяется по очереди к каждому объекту данных и преобразует данные типа дата или дата-время в строку формата ISO (см. «Работа с датами, временем и сложными типами данных»).

***Пример 3-2.** Преобразование типа datetime в JSON*

```
import datetime
import json

class JSONDateTimeEncoder(json.JSONEncoder): ❶
   def default(self, obj):
      if isinstance(obj, (datetime.date, datetime.datetime)): ❷
         return obj.isoformat()
      else:
         return json.JSONEncoder.default(self, obj)
```

```
def dumps(obj):
    return json.dumps(obj, cls=JSONDateTimeEncoder) ❸
```

❶ Создаем подкласс от класса `JSONEncoder` для кастомизации энкодера, который будет обрабатывать даты.

❷ Проверяем, имеет ли объект тип `datetime`. Если true, то преобразуем в `isoformat` все date и datetime (например, 2021-11-16T16:41:14.650802.)

❸ Передаем наш кастомный энкодер в качестве аргумента `cls`.

Теперь рассмотрим, как обновленная функция `dumps` справляется с данными типа `datetime`:

```
now_str = dumps({'time': datetime.datetime.now()})
now_str
Out:
'{"time": "2021-11-16T16:41:14.650802"}'
```

Поле `time`, корректно сконвертированное в строку формата ISO, готово к декодированию в объект `Date` в JavaScript (см. «Работа с датами, временем и сложными типами данных»).

Вы можете написать с нуля универсальный декодер, который будет обрабатывать строки дат в произвольных файлах JSON[1], но не советую. Порой строки дат бывают настолько причудливыми, что работу с ними лучше всего выполнять вручную, опираясь на практически всегда известный набор данных.

Проверенный временем метод `strptime` из пакета `datetime.datetime` успешно преобразует строки в известном формате времени в экземпляр Python `datetime`:

```
In [0]: from datetime import datetime
In [1]: time_str = '2021/01/01 12:32:11'
In [2]: dt = datetime.strptime(time_str, '%Y/%m/%d %H:%M:%S') ❶
In [3]: dt
Out[2]: datetime.datetime(2021, 1, 1, 12, 32, 11)
```

❶ Метод `strptime` сопоставляет строки времени с форматом строк с помощью различных директив, например `%Y` (год в виде 4 цифр) и `%H` (час в 24-часовом формате с добавлением нулей). Если формат строки подходит,

[1] В модуле Python `dateutil` есть подкласс parser, который может парсить большинство форматов дат и времени, он может стать хорошей основой.

создается экземпляр Python `datetime`. Список всех директив см. в документации Python.

Если `strptime` получает строку времени неподходящего формата, то выдает ошибку `ValueError`:

```
dt = datetime.strptime('1/2/2021 12:32:11', '%Y/%m/%d %H:%M:%S')
------------------------------------------------------------
ValueError                Traceback (most recent call last)
<ipython-input-111-af657749a9fe> in <module>()
----> 1 dt = datetime.strptime('1/2/2021 12:32:11',\
    '%Y/%m/%d %H:%M:%S')
...
ValueError: time data '1/2/2021 12:32:11' does not match
            format '%Y/%m/%d %H:%M:%S'
```

Чтобы преобразовать поля даты известного формата в `datetime` для списка словарей `data`, вы можно сделать что-то вроде следующего:

```
data = [
    {'id': 0, 'date': '2020/02/23 12:59:05'},
    {'id': 1, 'date': '2021/11/02 02:32:00'},
    {'id': 2, 'date': '2021/23/12 09:22:30'},
]

for d in data:
    try:
        d['date'] = datetime.strptime(d['date'],\
              '%Y/%m/%d %H:%M:%S')
    except ValueError:
        print('Oops! - invalid date for ' + repr(d))
# Вывод:
# Oops! - invalid date for {'id': 2, 'date': '2021/23/12 09:22:30'}
```

Итак, мы разобрались с двумя самыми популярными форматами файлов данных, теперь перейдем к более серьезным вопросам и посмотрим, как считывать данные из баз данных SQL и NoSQL и как их записывать.

SQL

Самой популярной и, на мой взгляд, лучшей Python-библиотекой для работы с базами данных является SQLAlchemy. Она позволяет использовать «сырые» SQL-инструкции, если важна скорость и эффективность, но также предоставляет высокоуровневый API — мощную систему объектно-реляционного отображения (ORM), благодаря которой можно работать с таблицами SQL практически как с классами Python.

Непросто организовать чтение и запись данных с использованием SQL-запросов и одновременно позволить пользователям относиться к данным, как к контейнеру Python. Хотя работать с SQLAlchemy гораздо удобнее, чем с низкоуровневым SQL-движком, это довольно сложная библиотека. Я расскажу здесь об ее основах на примере наших данные, но советую также ознакомиться с отличной документацией по SQLAlchemy. Давайте для чтения и записи вновь используем набор данных `nobel_winners`:

```
nobel_winners = [
   {'category': 'Physics',
    'name': 'Albert Einstein',
    'nationality': 'Swiss',
    'gender': 'male',
    'year': 1921},
    ...
]
```

Сначала запишем данные с помощью SQLAlchemy в SQLite-файл, начав с создания движка БД.

Создание движка базы данных

Первое, что необходимо сделать при запуске сеанса SQLAlchemy, — это инициализировать движок БД. Этот движок установит соединение с соответствующей базой данных и выполнит все преобразования, необходимые для общих инструкций SQL, сгенерированных SQLAlchemy, и для возвращаемых данных.

Существуют движки для большинства популярных баз данных, а также опция *memory*, когда база данных хранится в оперативной памяти, что обеспечивает быстрый доступ для тестирования[1]. Самое замечательное — движки

[1] Важно: не рекомендуется использовать разные конфигурации баз данных для тестирования и продакшна.

взаимозаменяемы, так что вы можете в процессе разработки кода использовать удобную однофайловую SQLite, а затем на этапе продакшн перейти на более подходящую для промышленного применения базу данных, например, PostgreSQL, изменив всего одну строку в конфигурации. Полный список движков см. в документации SQLAlchemy https://docs.sqlalchemy.org/en/14/core/engines.html.

Формат URL-адреса базы данных следующий:

```
dialect+driver://username: password@host: port/database
```

Например, для подключения к базе данных MySQL `'nobel_winners'`, запущенной на локальном хосте, потребуется код вроде приведенного ниже. Обратите внимание, что на данном этапе `create_engine` фактически не выполняет SQL-запросов, а просто настраивает фреймворк[1]:

```
engine = create_engine(
             'mysql://kyran:mypsswd@localhost/nobel_winners')
```

Мы будем использовать базу данных SQLite, добавив параметр `echo` равно `True`, чтобы видеть все SQL-инструкции, сгенерированные SQLAlchemy. Обратите внимание, что после двоеточия используются три обратных слеша:

```
from sqlalchemy import create_engine

engine = create_engine(
          'sqlite:///data/nobel_winners.db', echo=True)
```

SQLAlchemy предлагает различные подходы к взаимодействию с базами данных, но я рекомендую использовать декларативный стиль, если нет веских причин выбрать что-то более низкоуровневое. При декларативном отображении вы создаете подклассы базовых классов Python для SQL-таблиц, а SQLAlchemy анализирует их структуру и взаимосвязи. Подробнее см. SQLAlchemy, Declarative Mapping.

Определение таблиц БД

Сначала создадим класс `Base`, применив функцию `declarative_base()`. Он будет базой для создания классов таблиц, из которых SQLAlchemy будет

[1] Подробнее о ленивой (отложенной) инициализации SQLAlchemy см. *Engine Configuration*.

создавать структуру таблиц БД. Классы таблиц позволяют взаимодействовать с базой данных в духе Python:

```
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()
```

Примечание: большинство SQL-библиотек требуют формально определять структуру таблиц. Это не касается MongoDB, которая является одним из вариантов базы данных NoSQL, не использующей табличную структуру. Немного позже в этой главе мы рассмотрим библиотеку Dataset.

При помощи класса `Base`, мы можем определять различные таблицы, в данном случае — одну таблицу `Winner`. В примере 3-3 показано, как унаследоваться от `Base` и, используя типы данных SQLAlchemy, определить схему таблицы. Обратите внимание на атрибут `__tablename__`, который будет устанавливать имя таблицы SQL и использоваться как ключевое слово для ее извлечения, а также на опциональный метод `__repr__` для настраиваемого вывода результата в строку.

***Пример 3-3.** Определение таблицы базы данных SQL*

```
from sqlalchemy import Column, Integer, String, Enum
// ...

class Winner(Base):
    __tablename__ = 'winners'
    id = Column(Integer, primary_key=True)
    category = Column(String)
    name = Column(String)
    nationality = Column(String)
    year = Column(Integer)
    gender = Column(Enum('male', 'female'))
    def __repr__(self):
        return "<Winner(name='%s', category='%s', year='%s')>"\
%(self.name, self.category, self.year)
```

После создания подкласса `Base` в примере 3-3 вызываем метод `metadata.create_all` к движку БД, чтобы создать базу данных[1]. Так как при создании движка мы установили `echo` равно `True`, все SQL-инструкции, сгенерированные SQLAlchemy, мы увидим в командной строке.

```
Base.metadata.create_all(engine)

2021-11-16 17:58:34,700 INFO sqlalchemy.engine.Engine BEGIN (implicit)
...
CREATE TABLE winners (
        id INTEGER NOT NULL,
        category VARCHAR,
        name VARCHAR,
        nationality VARCHAR,
        year INTEGER,
        gender VARCHAR(6),
        PRIMARY KEY (id)
)...
2021-11-16 17:58:34,742 INFO sqlalchemy.engine.Engine COMMIT
```

После объявления новой таблицы `winners` нужно добавить в нее инстансы записей о лауреатах.

Добавление записей с помощью Session

Для работы с базой данных требуется создать объект Session.

```
from sqlalchemy.orm import sessionmaker

Session = sessionmaker(bind=engine)
session = Session()
```

Теперь используем класс `Winner`, чтобы создать его экземпляры и строки таблицы, и добавим их в сессию:

```
albert = Winner(**nobel_winners[0]) ❶
session.add(albert)
```

[1] Предполагается, что этой таблицы в базе данных еще нет. Если она уже существует, то Base будет применен при вставке новых данных и преобразовании текущих.

SKLADCHIK.ORG

```
session.new ❷
Out:
IdentitySet([<Winner(name='Albert Einstein', category='Physics',
              year='1921')>])
```

❶ Оператор ** распаковывает первого участника `nobel_winners` в пары ключ-значение: (`name='Albert Einstein', category='Physics'...`).
❷ `new` позволяет просмотреть все элементы, добавленные в текущей сессии.

Обратите внимание, что все добавления в БД и удаления из нее выполняются на языке Python. Чтобы сохранить изменения в базе данных, применим метод `commit`.



Используйте вызовы commit как можно реже, пусть SQLAlchemy творит свою магию за кулисами. После коммита все действия с базой данных обобщаются SQLAlchemy и эффективно передаются. Коммиты завершают транзакций, а это медленный процесс, который желательно максимально ограничить, и SQLAlchemy дает такую возможность.

Как показывает метод `new`, мы добавили `Winner` в сессию. Можно удалить объект с помощью `expunge`, после чего останется пустой `IdentitySet`:

```
session.expunge(albert) ❶
session.new
Out:
IdentitySet([])
```

❶ Удаляет объект из сессии (метод `expunge_all` удаляет все объекты, добавленные в текущую сесию).

Пока никаких добавлений в базу данных или удалений из нее не производилось. Добавим всех лауреатов из списка `nobel_winners` в сессию и сохраним их в базу данных.

```
winner_rows = [Winner(**w) for w in nobel_winners]
session.add_all(winner_rows)
session.commit()
Out:
```

```
INFO:sqlalchemy.engine.base.Engine:BEGIN (implicit)
...
INFO:sqlalchemy.engine.base.Engine:INSERT INTO winners (name,
category, year, nationality, gender) VALUES (?, ?, ?, ?, ?)
INFO:sqlalchemy.engine.base.Engine:('Albert Einstein',
'Physics', 1921, 'Swiss', 'male')
...
INFO:sqlalchemy.engine.base.Engine:COMMIT
```

Теперь данные из `nobel_winners` сохранены в базе данных. Далее рассмотрим, как их обрабатывать и как воссоздать целевой список из примера 3.1.

Запросы к базе данных

Для доступа к данным используем метод `session.query`. Результат можно фильтровать, группировать и комбинировать разными способами, используя стандартные возможности SQL. Все доступные методы запросов см. в документации SQLAlchemy. Сейчас я пробегусь по некоторым самым распространенным запросам на примере нашего набора данных Nobel.

Сначала посчитаем количество строк в таблице winners:

```
session.query(Winner).count()
Out:
3
```

Теперь выберем лауреатов швейцарского происхождения:

```
result = session.query(Winner).filter_by(nationality='Swiss')
list(result)
Out:
[<Winner(name='Albert Einstein', category='Physics', year='1921')>]
```

❶ Метод `filter_by` использует ключевые выражения; его аналог в SQL — метод `filter`, например `filter(Winner.nationality == Swiss)`. Примечание: в `filter` используется логическое равенство ==.

Теперь получим лауреатов, чья национальность отличается от швейцарской:

```
result = session.query(Winner).filter(\
                Winner.category == 'Physics', \
                Winner.nationality != 'Swiss')
list(result)
Out:
[<Winner(name='Paul Dirac', category='Physics', year='1933')>]
```

Рассмотрим, как получить строку по ID:

```
session.query(Winner).get(3)
Out:
<Winner(name='Marie Curie', category='Chemistry', year='1911')>
```

Отсортируем лауреатов по годам:

```
res = session.query(Winner).order_by('year')
list(res)
Out:
[<Winner(name='Marie Curie', category='Chemistry',\
year='1911')>,
 <Winner(name='Albert Einstein', category='Physics',\
year='1921')>,
 <Winner(name='Paul Dirac', category='Physics', year='1933')>]
```

Чтобы восстановить наш целевой список, преобразуем объекты `Winner`, полученные через session.query, в словари Python. Напишем функцию, чтобы получить `dict` из класса SQLAlchemy. Чтобы получить метки столбцов, используем интроспекцию над таблицей (см. пример 3-4).

***Пример 3-4.** Преобразование объекта SQLAlchemy в dict*

```
def inst_to_dict(inst, delete_id=True):
    dat = {}
    for column in inst.__table__.columns: ❶
        dat[column.name] = getattr(inst, column.name)
    if delete_id:
        dat.pop('id') ❷
    return dat
```

❶ Обращается к классу таблицы экземпляра, чтобы получить список объектов столбцов.

❷ Если `delete_id` истина, удаляет SQL-поле первичного ключа.

Используем пример 3-4, чтобы преобразовать объекты `nobel_winners` обратно в список словарей:

```
winner_rows = session.query(Winner)
nobel_winners = [inst_to_dict(w) for w in winner_rows]
nobel_winners
Out:
[{'category': 'Physics',
  'name': 'Albert Einstein',
  'nationality': 'Swiss',
  'gender': 'male',
  'year': 1921},
  ...
]
```

Можно легко обновить строки базы данных, изменив свойства отражаемых в них объектов:

```
marie = session.query(Winner).get(3) ❶
marie.nationality = 'French'
session.dirty ❷
Out:
IdentitySet([<Winner(name='Marie Curie', category='Chemistry',
year='1911')>])
```

❶ Вызывает Марию Кюри, польку по национальности.

❷ `dirty` показывает все измененные экземпляры, которые еще не были сохранены в базе данных.

Сохраним для `Marie` изменение национальности с польской на французскую:

```
session.commit()
Out:
INFO:sqlalchemy.engine.base.Engine:UPDATE winners SET
nationality=? WHERE winners.id = ?
```

```
INFO:sqlalchemy.engine.base.Engine:('French', 3)
...

session.dirty
Out:
IdentitySet([])

session.query(Winner).get(3).nationality
Out:
'French'
```

Кроме обновления строк базы данных, вы также можете удалить результаты запроса:

```
session.query(Winner).filter_by(name='Albert Einstein').delete()
Out:
INFO:sqlalchemy.engine.base.Engine:DELETE FROM winners WHERE
winners.name = ?
INFO:sqlalchemy.engine.base.Engine:('Albert Einstein',)
1

list(session.query(Winner))
Out:
[<Winner(name='Paul Dirac', category='Physics', year='1933')>,
 <Winner(name='Marie Curie', category='Chemistry',\
 year='1911')>]
```

Если потребуется удалить всю таблицу, можно использовать атрибут `__table__` декларативного класса:

```
Winner.__table__.drop(engine)
```

До сих пор мы работали с одной таблицей лауреатов, не используя внешних ключей (foreign keys) или связей с табличными форматами, родственными CSV и JSON-файлам. SQLAlchemy обеспечивает то же удобство работы со связями между таблицами «один ко многим», «многие ко многим» и др., как и при выполнении базовых запросов с неявными объединениями, предоставляя методу `query` несколько классов таблиц или явно используя `join`. См. примеры в документации SQLAlchemy.

Модуль Dataset упрощает работу с SQL

Относительно недавно я стал пользоваться модулем Dataset, который делает работу с базами данных SQL для питонистов еще проще и интуитивно понятнее, чем это реализовано в таких мощных пакетах, как SQLAlchemy[1]. Dataset стремится обеспечить тот же уровень удобства, который характерен для баз данных NoSQL, вроде MongoDB, без необходимости заранее определять структуру таблицы. Он избавляет от массы формального boilerplate-кода, такого как определение схемы БД, которого требуют традиционные библиотеки. Dataset — это надстройка над SQLAlchemy, поэтому совместим с большинством баз данных и сохраняет все преимущества надежности и зрелости этой, лучшей в своем классе, библиотеки. Проверим, как он справляется с чтением и записью целевого набора данных из примера 3-1.

Чтобы протестировать Dataset, используем базу данных SQLite *nobel_winners.db*, которую мы недавно создали. Сначала подключимся к нашей базе данных SQL, используя тот же формат URL/файла, что и SQLAlchemy:

```
import dataset

db = dataset.connect('sqlite:///data/nobel_winners.db')
```

Извлечем таблицу из базы данных `db`, используя ее имя в качестве ключа, а затем применим метод `find` без аргументов, который возвращает всех лауреатов:

```
wtable = db['winners']
winners = wtable.find()
winners = list(winners)
winners
#Out:
#[OrderedDict([(u'id', 1), ('name', 'Albert Einstein'),
# ('category', 'Physics'), ('year', 1921), ('nationality',
# 'Swiss'), ('gender', 'male')]), OrderedDict([('id', 2),
# ('name', 'Paul Dirac'), ('category', 'Physics'),
# ('year', 1933), ('nationality', 'British'), ('gender',
# 'male')]), OrderedDict([('id', 3), ('name', 'Marie
# Curie'), ('category', 'Chemistry'), ('year', 1911),
# ('nationality', 'Polish'), ('gender', 'female')])]
```

[1] Официальный девиз Dataset — «Базы данных для ленивых». Модуль не входит в стандартный пакет Anaconda, его можно установить, набрав в командной строке: `$ pip install dataset`.

Обратите внимание, что метод Dataset find возвращает экземпляры записей в виде словарей OrderedDict. Эти полезные контейнеры — расширение класса dict языка Python — ведут себя почти как dict, но запоминают порядок, в котором были вставлены элементы, что позволяет при итерации извлечь элемент, вставленный последним, и сделать многое другое. Это очень удобная дополнительная возможность.



OrderedDict модуля Dataset ведет начало от библиотеки Python collections. Особенно полезны входящие в нее классы defaultdict и Counter. Подробнее о типах данных из collections см. в документации Python.

С помощью Dataset вновь создадим таблицу winners, сначала удалив существующую:

```
wtable = db['winners']
wtable.drop()

wtable = db['winners']
wtable.find()
#Вывод:
#[]
```

Теперь для воссоздания удаленной таблицы winners не требуется определять ее схему, как мы делали раньше с помощью SQLAlchemy. Dataset сам ее определит, исходя из данных, которые мы добавили, и неявно выполнит создание SQL. Именно к такому удобству привыкаешь при работе с базами данных NoSQL на основе коллекций. Вставим несколько словарей из набора данных nobel_winners (пример 3-1). Для доступа к базе данных используем транзакции и оператор with, чтобы вставить объекты, и сохраним их[1]:

```
with db as tx: ❶
    tx['winners'].insert_many(nobel_winners)
```

❶ Оператор with гарантирует, что транзакция tx завершится и данные запишутся в базу данных.

[1] Подробнее об использовании транзакций для групповых обновлений см. документацию https://dataset.readthedocs.io/en/latest/quickstart.html#using-transactions.

Проверим, что получилось:

```
list(db['winners'].find())
Out:
[OrderedDict([('id', 1), ('name', 'Albert Einstein'),
('category', 'Physics'), ('year', 1921), ('nationality',
'Swiss'), ('gender', 'male')]),
...
]
```

Лауреаты были вставлены корректно, а `OrderedDict` сохранил порядок их добавления.

Dataset отлично подходит для работы с базами данных SQL, особенно для извлечения данных, которые нужно обработать или визуализировать. Для более сложных манипуляций он позволяет перейти к основному API SQLAlchemy, и использовать метод `query`.

Теперь, когда мы ознакомились с основами работы с базами данных SQL, рассмотрим, как Python делает работу с самой популярной базой данных NoSQL такой же удобной.

MongoDB

Документоориентированные хранилища данных, в том числе MongoDB, предлагают большие преимущества при преобразовании сырых данных в структурированные. Как и у всех инструментов, у баз данных NoSQL есть свои плюсы и минусы. Если данные, которые есть у вас, уже очищены и обработаны, и вряд ли возникнет необходимость в мощном языке запросов SQL, позволяющем объединять данные нескольких таблиц, то, вероятно, на начальном этапе вам будет проще работать с MongoDB. MongoDB отлично подходит для визуализации данных в браузере, так как использует формат данных BSON (бинарный JSON). BSON поддерживает двоичные данные и объекте `datetime`, а также прекрасно совместим с JavaScript.

Вспомним целевой набор данных, который собираемся записать и прочитать:

```
nobel_winners = [
   {'category': 'Physics',
    'name': 'Albert Einstein',
    'nationality': 'Swiss',
```

```
    'gender': 'male',
    'year': 1921},
    ...
]
```

Создание коллекции MongoDB с помощью Python — минутное дело:

```
from pymongo import MongoClient

client = MongoClient() ❶
db = client.nobel_prize ❷
coll = db.winners ❸
```

❶ Создает клиента Mongo, со значениями по умолчанию для хоста и порта.
❷ Создает базу данных `nobel_prize` или получает доступ к ней.
❸ Если коллекция winners существует, обновит ее, иначе (как в данном случае) — создаст.

Использование констант для доступа к MongoDB

Чтобы получить доступ к базе данных MongoDB или создать ее с помощью Python, используется одна и та же операция с применением либо точечной нотации, либо ключа доступа в квадратных скобках:

```
db = client.nobel_prize
db = client['nobel_prize']
```

Хотя это удобно, одна-единственная орфографическая ошибка, например `noble_prize`, повлечет за собой как создание базы с неверным названием, так и невозможность затем ее обновить, указывая правильное. Поэтому для доступа к базам данных и коллекциям MongoDB я советую использовать константы:

```
DB_NOBEL_PRIZE = 'nobel_prize'
COLL_WINNERS = 'winners'

db = client[DB_NOBEL_PRIZE]
coll = db[COLL_WINNERS]
```

По умолчанию MongoDB работают на локальном порту 27017, но могут находиться где угодно в интернете. Они также могут принимать необязательные имя пользователя и пароль. В примере 3-5 показано, как создать простую утилитарную функцию для подключения к нашей базе данных, со стандартными умолчаниями.

Пример 3-5. *Доступ к базе данных MongoDB*

```
from pymongo import MongoClient
def get_mongo_database(db_name, host='localhost',\
                       port=27017, username=None, password=None):
   """ Get named database from MongoDB with/out authentication """
   # make Mongo connection with/out authentication
   if username and password:
      mongo_uri = 'mongodb://%s:%s@%s/%s'%\ ❶
      (username, password, host, db_name)
      conn = MongoClient(mongo_uri)
   else:
      conn = MongoClient(host, port)
   return conn[db_name]
```

❶ Укажем имя базы данных в MongoDB URI (Uniform Resource Identifier, «унифицированный идентификатор ресурса»), так как у пользователя может не быть привилегий для доступа к данной базе данных.

Теперь создадим базу данных лауреатов Нобелевской премии и добавим целевой набор данных (пример 3-1). Получим коллекцию лауреатов, используя для доступа константы:

```
db = get_mongo_database(DB_NOBEL_PRIZE)
coll = db[COLL_WINNERS]
```

После этого вставить набор данных о лауреатах Нобелевской премии проще простого:

```
coll.insert_many(nobel_winners)
coll.find()
Out:
[{'_id': ObjectId('61940b7dc454e79ffb14cd25'),
  'category': 'Physics',
```

```
 'name': 'Albert Einstein',
 'nationality': 'Swiss',
 'year': 1921,
 'gender': 'male'},
{'_id': ObjectId('61940b7dc454e79ffb14cd26'), ... }
...]
```

Полученный массив, состоящий из уникальных идентификаторов объекта `ObjectId` можно было бы использовать для следующего извлечения, но MongoDB уже оставила свой след в списке `nobel_winners`, добавив скрытое свойство `id`[1].



`ObjectId` имеют довольно много скрытых функций и представляют собой нечто большее, чем просто случайный идентификатор. Например, можно получить время генерации `ObjectId`, что даст удобную временную метку:

```
import bson
oid = bson.ObjectId()
oid.generation_time
Out: datetime.datetime(2015, 11, 4, 15, 43, 23...
```

Подробнее см. в документации MongoDB BSON.

Теперь, когда в коллекции лауреатов есть несколько элементов, MongoDB значительно упрощает их поиск с помощью метода `find`, принимающего запрос к словарю:

```
res = coll.find({'category':'Chemistry'})
list(res)
Out:
[{'_id': ObjectId('55f8326f26a7112e547879d6'),
  'category': 'Chemistry',
  'name': 'Marie Curie',
  'nationality': 'Polish',
  'gender': 'female',
  'year': 1911}]
```

[1] Один из плюсов MongoDB — то, что идентификаторы `ObjectId` генерируются на стороне клиента, что устраняет необходимость иметь базу данных для них.

Существует несколько операторов с префиксом "$", которые позволяют выполнять сложные запросы. Например, с помощью оператора `$gt` (greater-than) найдем всех лауреатов после 1930 года.

```
res = coll.find({'year': {'$gt': 1930}})
list(res)
Out:
[{'_id': ObjectId('55f8326f26a7112e547879d5'),
  'category': 'Physics',
  'name': 'Paul Dirac',
  'nationality': 'British',
  'gender': 'male'
```

Также можно использовать логические выражения, например найти всех лауреатов после 1930 года или всех лауреатов-женщин:

```
res = coll.find({'$or':[{'year': {'$gt': 1930}},\
{'gender':'female'}]})
list(res)
Out:
[{'_id': ObjectId('55f8326f26a7112e547879d5'),
  'category': 'Physics',
  'name': 'Paul Dirac',
  'nationality': 'British',
  'gender': 'male',
  'year': 1933},
 {'_id': ObjectId('55f8326f26a7112e547879d6'),
  'category': 'Chemistry',
  'name': 'Marie Curie',
  'nationality': 'Polish',
  'gender': 'female',
  'year': 1911}]
```

Полный список выражений запросов см. в документации MongoDB (https://www.mongodb.com/docs/manual/tutorial/query-documents/).

Завершающий тест: превратим нашу новую коллекцию лауреатов обратно в список словарей Python. Создадим маленькую функцию для этой задачи:

```
def mongo_coll_to_dicts(dbname='test', collname='test',\
                        query={}, del_id=True, **kw): ❶
    db = get_mongo_database(dbname, **kw)
    res = list(db[collname].find(query))

    if del_id:
        for r in res:
            r.pop('_id')

    return res
```

❶ Пустой `query dict {}` найдет все документы в коллекции. Флаг `del_id` по умолчанию удаляет идентификаторы ObjectId из элементов.

Теперь мы можем создать целевой набор данных, вызвав созданную выше функцию:

```
mongo_coll_to_dicts(DB_NOBEL_PRIZE, COLL_WINNERS)
Out:
[{'category': 'Physics',
  'name': 'Albert Einstein',
  'nationality': 'Swiss',
  'gender': 'male',
  'year': 1921},
  ...
]
```

«Бессхемные» базы данных MongoDB отлично подходят для быстрого прототипирования при работе в одиночку или небольшой командой. Вероятно, что по мере роста кодовых баз возникнет потребность в формальной структуре данных для проверки целостности и поддержки версий кода. В таком случае важно будет выбрать модель, которая позволит легко адаптироваться к изменениям. Есть еще пара удобных особенностей: передача словарей Python в качестве запросов к PyMongo и доступ к сгенерированным на стороне клиента идентификаторам `ObjectId`.

Используя данные `nobel_winners` из примера 3-1, мы рассмотрели все нужные нам форматы файлов и баз данных. Прежде чем подводить итоги, рассмотрим особый случай работы с датами и временем.

Работа с датами, временем и сложными типами данных

Удобство обработки дат и времени — фундаментальная потребность при работе над визуализацией данных, но добиться этого непросто. Существует множество способов представления даты и времени в виде строки, и каждый требует особого подхода к кодированию или декодированию. По этой причине полезно выбрать для своей работы один формат и рекомендовать другим делать то же самое. Я советую использовать международный стандарт ISO 8601 https://oreil.ly/HePpN для строкового представления дат и времени, а также указывать время в формате UTC (всемирное координированное время)[1]. Вот примеры строк в формате ISO 8601:

```
2021-09-23                  Дата (формат кода Python/C '%Y-%m-%d')
2021-09-23T16:32:35Z        Дата и время в UTC (Z после времени,
                            формат 'T%H:%M:%S')
2021-09-23T16:32+02:00      Положительное смещение +02:00 от UTC
                            (например, Центральноевропейское время)
```

Обратите внимание: важно учитывать разные часовые пояса. Они не всегда привязаны к долготе (см. статью «Часовой пояс» в Википедии), и часто точное время проще получить, комбинируя UTC с географическими координатами.

ISO 8601 — стандарт, используемый в JavaScript, и его легко поддерживает Python. При визуализации данных для интернета ключевая задача — создать строковое представление, которое можно передавать между Python и JavaScript с помощью JSON и легко обрабатывать на обеих сторонах.

Возьмем дату и время в формате Python `datetime`, преобразуем их в строку, а затем посмотрим, как использовать эту строку в JavaScript.

Создадим `datetime` в Python:

```
from datetime import datetime

d = datetime.now()
d.isoformat()
```

[1] Для получения локального времени из UTC можно сохранить смещение часового пояса или, что предпочтительнее, определить его по географическим координатам. Это связано с тем, что часовые пояса не строго соответствуют меридианам.

```
Out:
'2021-11-16T22:55:48.738105'
```

Сохраняем строку в JSON/CSV-файле, читаем в JavaScript и создаем объект `Date`:

```
// JavaScript
d = new Date('2021-11-16T22:55:48.738105')
> Tue Nov 16 2021 22:55:48 GMT+0000 (Greenwich Mean Time)
```

Возвращаем дату и время в формате строки ISO 8601 с помощью `toISOString`:

```
// JavaScript
d.toISOString()
> '2021-11-16T22:55:48.738Z'
```

И наконец, прочитаем строку обратно в Python с помощью модуля `dateutil`[1]. Проверим результат:

```
from dateutil import parser
d = parser.parse('2021-11-16T22:55:48.738Z')
d
Out:
datetime.datetime(2021, 11, 16, 22, 55, 48, 738000,\
tzinfo=tzutc())
```

Обратите внимание, что при передаче данных от Python к JavaScript теряется точность, поскольку JS использует миллисекунды, а Python — микросекунды. Для визуализации это редко бывает критично, но об этом полезно помнить на случай возникновения странных временных ошибок.

Резюме

Цель этой главы — научить вас уверенно работать с данными в Python, используя различные форматы файлов и базы данных, с которыми вы можете столкнуться

[1] Установите `dateutil` командой `pip install python-dateutil`. Это расширение для `datetime` с большими возможностями. Подробнее см. на Read the Docs.

при визуализации данных. Эффективное использование баз данных требует времени для освоения, но теперь вы должны уметь выполнять базовые операции чтения и записи данных для большинства задач визуализации. Теперь, когда мы подготовили все необходимое для плавной работы нашего тулчейна, перейдем к основам веб-разработки, которые понадобятся для следующих глав.

Основы веб-разработки

В этой главе вы познакомитесь с основами веб-разработки, необходимыми для понимания веб-страниц, из которых вы извлекаете данные, и для структурирования тех из них, которые послужат каркасом для JavaScript-визуализаций. Вы увидите, что даже базовых знаний достаточно для современной веб-разработки, особенно если вы фокусируетесь на самодостаточных визуализациях, а не на полноценных сайтах.

Глава, как обычно, может использоваться и как справочная, и как учебная. То, что вы уже знаете, смело пропускайте и переходите к новому материалу.

Общая картина

Простая веб-страница — основной структурный блок Всемирной паутины (World Wide Web, WWW), той части интернета, которую используют люди, — состоит из файлов различных типов. Помимо мультимедийных файлов (изображений, видео, звуковых и др.), ключевыми элементами являются текстовые файлы HTML (язык гипертекстовой разметки), CSS (каскадные таблицы стилей) и JavaScript. Эти три компонента вместе с файлами необходимых данных доставляются через протокол HTTP и используются для создания страницы, которую вы видите и с которой взаимодействуете в браузере. Структура страницы описывается объектной моделью документа — DOM (Document Object Model) — иерархическим деревом, на котором строится контент. Для создания современных веб-визуализаций необходимо понимать, как взаимодействуют эти элементы. Цель главы — подготовить вас к работе.

Веб-разработка — обширная область, так что я не ставлю задачи сделать из вас гуру веб-разработки. Предполагаю, что вы хотите минимизировать объем веб-разработки, сосредоточившись только на том, что нужно для создания современных визуализаций. Чтобы создать визуализации подобные представленным на сайте d3js.org, опубликованным в *New York Times* или включенным в базовые дашборды, требуется на удивление мало навыков веб-разработки. На крупный сайт плоды ваших трудов может выложить кто-нибудь, создающий сайты профессионально. На небольшой персональный сайт вы сможете добавить визуализацию самостоятельно.

Одностраничные приложения

Одностраничное приложение (*Single-page application*, SPA) — это веб-приложение, чьи данные динамически заполняются при помощи JavaScript через атрибуты классов и ID, определенные в минималистичном каркасе из HTML и CSS. Под это описание подходят многие современные визуализации данных, в том числе визуализация данных о нобелевских лауреатах, к созданию которой ведет эта книга.

SPA часто самодостаточны: их корневую папку легко интегрировать в существующий веб-сайт или запускать напрямую (в этом случае требуется только HTTP-сервер, например Apache или NGINX).

Рассматривая визуализацию данных как SPA, мы снижаем умственную нагрузку, связанную с веб-разработкой визуализаций в JavaScript, и фокусируемся на самом программировании. Для размещения визуализации в интернете вам будет достаточно базовых навыков, которые легко приобретаются, но быстро окупаются. Часто эту работу можно делегировать другому специалисту.

Настройка инструментов

Для создания современных визуализаций данных требуются всего лишь приличный текстовый редактор, современный браузер и терминал (см. рисунок 4.1). Я расскажу о минимальных, на мой взгляд, требованиях к редактору веб-разработки, а также о необязательных, но полезных функциях.

В качестве встроенного в браузер набора инструментов веб-разработки я выбрал Chrome DevTools, доступный бесплатно на всех платформах. Он предоставляет множество функций, распределенных по вкладкам. В этой главе мы рассмотрим:

— вкладку *Elements* для изучения структуры веб-страницы, ее HTML-кода, CSS-стилей и элементов DOM;

— вкладку *Sources*, которая используется для отладки JavaScript.

Вам понадобится терминал для вывода результатов, запуска локального веб-сервера и, возможно, для экспериментов с интерактивной оболочкой IPython. Сейчас я чаще всего использую в браузере блокноты Jupyter https://jupyter.org/ в качестве *блокнота для набросков* веб-визуализаций данных. Это удобно, так как сессия сохраняется в файле *.ipynb*, и ее можно перезапустить позже. Кроме того, Jupyter Notebook позволяет постепенно исследовать данные с помощью встроенных диаграмм. Мы воспользуемся этими возможностями в разделе III.

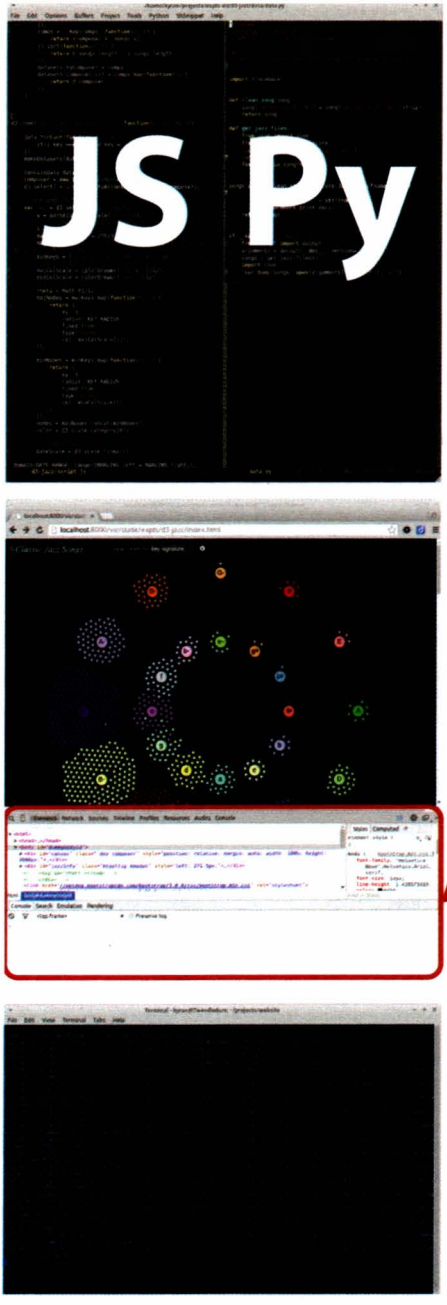


Рис. 4.1. *Основные инструменты веб-разработки*

Прежде, чем выполнять нужные шаги, разберемся, чего делать не нужно, и попутно развенчаем несколько мифов.

Мифы об IDE, фреймворках и инструментах

Многие начинающие JavaScript-разработчики считают, что для веб-программирования обязательно нужны сложные наборы инструментов и IDE, которые используют разработчики в больших корпорациях (да и много где еще). Такие инструменты могут дорого стоить и требуют времени на освоение. Хорошая новость: вы можете создавать профессиональные веб-визуализации, используя только приличный текстовый редактор. Пока вы не работаете с современными фреймворками JavaScript (с этим лучше повременить, пока как следует не освоите веб-разработку), IDE почти не дает преимуществ и часто подтормаживает. Еще хорошие новости: де-факто стандартом в веб-разработке стала бесплатная и легковесная среда разработки Visual Studio Code (VSCode). Если вы уже используете VSCode, продолжайте — это подходящий инструмент для работы по нашей книге.

Также распространен миф, что без фреймворков[1] невозможно продуктивно работать с JavaScript. Сейчас множество фреймворков (часто созданных крупными компаниями) борются за доминирование в экосистеме JS. Они появляются и исчезают так быстро, что я бы советовал новичкам просто их игнорировать, пока не наберутся опыта. Используйте небольшие узкоспециализированные библиотеки, например, из экосистемы jQuery или функции-утилиты библиотеки Underscore.js и вы увидите, как далеко можно с ними продвинуться, прежде чем вам понадобится какой-нибудь навороченный фреймворк. Подключайте фреймворки только при явной необходимости, а не из-за хайпа[2]. Еще один важный момент — D3 (главная библиотека веб-визуализации) плохо совместима с крупными фреймворками, которые мне известны, особенно с теми, что стремятся контролировать DOM. Интеграция D3 с фреймворками — задача для продвинутых.

На форумах веб-разработчиков (Reddit и Stack Overflow) вы обнаружите множество инструментов. Среди них JS+CSS-минификаторы и наблюдатели (watchers) для автоматического обнаружения изменений файлов и перезагрузки веб-страниц во время разработки. По моему опыту, хотя пара-тройка из них полезна, в основном это нестабильно работающие инструменты, которые скорее отнимают время, чем повышают производительность. Повторюсь, можно быть очень продуктивным и без них. Используйте их только при острой необходимости. Кое-какие из них стоящие, но лишь считаные единицы можно назвать средством первой необходимости для визуализации данных.

---

[1] Несколько интересных альтернатив полнофункциональным фреймворкам сейчас вызывают ажиотаж, например, Alpine.js и htmx. Они хорошо работают с такими фреймворками Python, как Django и Flask.

[2] Я немало времени зря потратил на различные инструменты, не повторяйте моих ошибок.

Надежный текстовый редактор

Главный инструмент веб-разработки — удобный для вас текстовый редактор, который, как минимум, должен подсвечивать синтаксис для нескольких языков, в нашем случае: HTML, CSS, JavaScript и Python. Можно, конечно, обойтись редактором без подсветки, но в долгосрочной перспективе это усложнит работу. Подсветка синтаксиса, проверка кода (линтер), автоматические отступы и им подобные ценные функции значительно снимают умственную нагрузку. Без них продуктивность резко падает. Минимальные требования к редактору:

— подсветка синтаксиса для всех используемых языков;
— настройка отступов (например, «мягкая табуляция», включающая 4 пробела для Python, 2 — для JavaScript);
— поддержка нескольких окон, панелей или вкладок для удобной навигации по коду;
— хороший линтер (см. рисунок 4.2).

В современных редакторах эти функции есть по умолчанию, кроме линтинга — его иногда нужно настраивать.

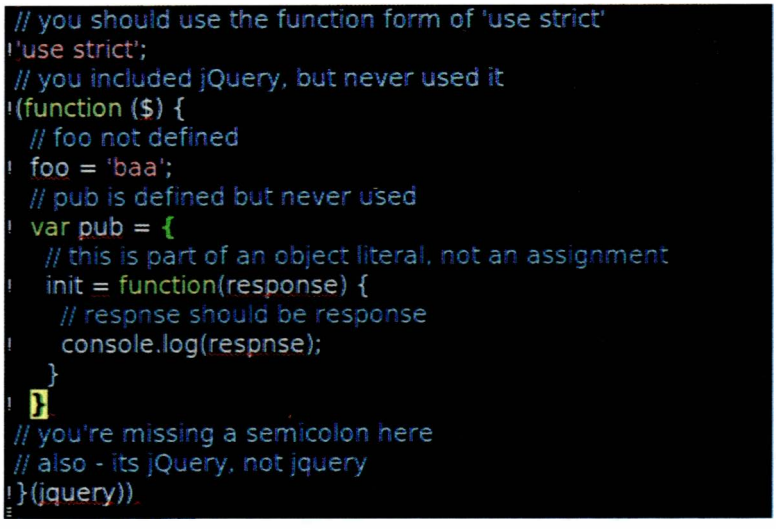


Рис. 4.2. *Запущенный линтер непрерывно анализирует код JavaScript, подсвечивая синтаксические ошибки красным и помечая строки с ошибками восклицательным знаком*

Браузер с инструментами разработчика

Одна из причин, по которой полнофункциональная IDE не столь необходима в современной веб-разработке, заключается в том, что отладку удобнее проводить в самом браузере, а темпы обновления браузеров таковы, что IDE сложно угнаться за ними. Кроме того, для современных веб-браузеров созданы мощные наборы инструментов отладки и разработки. Лучший из них Chrome DevTools. Он предоставляет обширный функционал: от продвинутой (для питониста) отладки (параметрические точки останова, отслеживание переменных и др.) до профилирования и оптимизации использования памяти и процессора, эмуляции устройств (например, проверки отображения веб-страницы на смартфоне или планшете) и многого другого. Chrome DevTools — мой основной инструмент для отладки, именно его я и буду использовать в книге. Он бесплатный, как и все, что я здесь предлагаю.

Терминал или командная строка

Терминал или командная строка позволяют запускать серверы и выводить полезную информацию, такую как логи. Еще в них можно протестировать модули Python и запустить интерпретатор Python (по многим параметрам лучший из них IPython).

Создание веб-страницы

Типичная веб-визуализация включает четыре элемента:

— HTML-каркас с плейсхолдерами для автоматизированной визуализации;
— свойства CSS, которые определяют стиль оформления, например: ширину границ, цвета, размер шрифта, размещение и контент блоков;
— скрипты JavaScript для создания визуализации;
— данные, которые нужно преобразовать.

Первые три элемента — просто текстовые файлы, созданные с помощью редактора и доставленные в браузер веб-сервером (см. главу 12). Рассмотрим каждый из них по очереди.

Передача данных через HTTP

Доставка файлов HTML, CSS и JS, формирующих конкретную веб-страницу (наряду с файлами данных, мультимедиа и др.), согласовывается между веб-сервером и браузером с помощью HTTP (HyperText Transfer Protocol, «протокола передачи гипертекста»). HTTP поддерживает ряд методов, из которых чаще всего используется GET — он запрашивает веб-ресурс, получая данные с сервера при успешном выполнении или возвращая ошибку в случае сбоя. В главе 6 мы будем использовать GET с модулем Python Requests для извлечения данных с веб-страниц.

Для обработки HTTP-запросов, которые генерирует браузер, необходим сервер. При разработке можно локально использовать встроенный веб-сервер Python из модуля `http` (часть стандартной библиотеки). Запустите сервер в командной строке с опциональным номером порта (по умолчанию 8000), как показано ниже:

```
$ python -m http.server 8080
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
```

Теперь сервер работает локально на порту 8080. Доступ к нему можно получить через браузер по адресу URL *http://localhost:8080*.

Хотя модуль `http.server` удобен для демонстраций, он обладает ограниченной функциональностью. Поэтому в разделе IV, мы перейдем к использованию полноценного сервера для разработки и продакшена, например Flask, который я выбрал для этой книги.

DOM

HTML-файлы, передаваемые через HTTP, преобразуются браузером в DOM (Document Object Model), древовидную структуру, которую JavaScript может динамически изменять. Именно программируемый DOM лежит в основе библиотек визуализации данных, таких как D3. DOM представляет собой иерархию узлов. Верхний узел — это объект document, соответствующий всей веб-странице.

По сути, весь HTML-код, который вы пишете или генерируете с помощью шаблона, преобразуется браузером в иерархическое дерево узлов, каждый из которых представляет HTML-элемент. Верхний узел называется корневым и является объектом документа *Document Object*, а все остальные узлы располагаются по принципу «родитель-дочерний элемент». Поскольку программное управление DOM лежит в основе таких библиотек, как jQuery и мощная D3, крайне

важно четко представлять, как это работает. Отличный способ получить представление о DOM — использовать один из веб-инструментов для изучения ветвей дерева, (я рекомендую *Chrome DevTools*).

Все, что отображается на веб-странице, учет состояния объекта (явное или скрытое, преобразование с помощью трансформации и т. д.) выполняется с помощью DOM. Важным нововведением D3 была связь данных непосредственно с элементами DOM и использование их для управления визуальными изменениями (отсюда название D3, Data-Driven Documents, «документы, управляемые данными»).

HTML-каркас

Как правило, веб-визуализации реализуются с помощью JavaScript поверх HTML-каркаса страницы.

HTML — язык, на котором описывается контент веб-страницы. В 1980-х его придумал физик Тим Бернерс-Ли (Tim Berners-Lee), сотрудник швейцарского исследовательского института CERN. Для структурирования содержимого страницы используются HTML-теги, например: `<div>`, `<img>` и `<h>`, а CSS отвечает за стиль оформления[1].

В версии HTML5 значительно сократилась доля boilerplate-кода, но суть языка остается практически неизменной уже тридцать лет.

HTML раньше включал в себя множество сбивающих с толку тегов шапки сайта, но в HTML5 появился более удобный для пользователя минимализм. Ниже показан необходимый минимум для стартового шаблона[2]:

```
<!DOCTYPE html>
<meta charset="utf-8">
<body>
  <!-- page content -->
</body>
```

Нужно всего лишь объявить документ HTML, набор символов 8-битного Unicode и тег `<body>`, под которым добавляется контент страницы. Это значительное улучшение по сравнению с тем, что требовалось ранее. Теперь

[1] Можно задать стиль в тегах HTML с помощью атрибута style, но, как правило, это плохая практика. Лучше использовать классы и идентификаторы CSS.

[2] Майк Бостоk (Mike Bostock), разработчик D3, даже отдельно поблагодарил у себя в блоге за такую оптимизацию Пола Айриша (Paul Irish), разработчика HTML5.

довольно просто научиться создавать документы, которые будут преобразованы в веб-страницы. Примечание: комментарии выглядят так: `<!-- comment -->`.

Если подходить к делу более реалистично, следует добавить немного CSS и JavaScript. Чтобы включить их непосредственно в HTML-документ, используются теги `<style>` и `<script>`, например:

```
<!DOCTYPE html>
<meta charset="utf-8">
<style>
/* CSS */
</style>
<body>
   <!-- page content -->
   <script>
   // JavaScript...
   </script>
</body>
```

Одностраничная форма с единым HTML-файлом часто используется в таких примерах визуализации, как на d3js.org. Эта форма удобна для демонстрации кода или отслеживания файлов, но в целом я бы рекомендовал создавать отдельные HTML, CSS и JavaScript- файлы. Таким образом вы, во-первых, упрощаете навигацию, особенно когда кодовая база разрастается, во-вторых, получаете все преимущества редактора при работе с конкретным языком: подсветку синтаксиса и линтинг кода (по сути, проверку синтаксиса на лету). Хотя и говорят, что есть редакторы и библиотеки, которые работают со встроенным в HTML документ кодом CSS и JavaScript, я таких не нашел.

Для подключения CSS и JavaScript-файлов применяют HTML-теги `<link>` и `<script>` следующим образом:

```
<!DOCTYPE html>
<meta charset="utf-8">
<link rel="stylesheet" href="style.css" />
<body>
   <!-- page content -->
   <script type="text/javascript" src="script.js"></script>
</body>
```

Разметка контента

Для визуализации обычно используют небольшое количество существующих HTML-тегов, поскольку страницы создаются автоматически путем добавления элементов к DOM-дереву.

Чаще всего используется тег `<div>`, обозначающий блок контента. В `<div>` могут содержаться другие `<div>`, формируя иерархию. Это позволяет выбирать элементы и обрабатывать события пользовательского интерфейса (UI), например, клики. Пример:

```
<div id="my-chart-wrapper" class="chart-holder dev">
   <div id="my-chart" class="bar chart">
      this is a placeholder, with parent #my-chart-wrapper
   </div>
</div>
```

Обратите внимание на атрибуты `id` и `class`. Они используются при выборе элементов DOM и для применения стилей CSS. ID — это уникальные идентификаторы (один на страницу и элемент). Класс можно применить к нескольким элементам, чтобы осуществить групповой выбор, а каждый элемент может иметь несколько классов.

Основные теги для работы с текстом — `<p>`, `<h1>-<h6>` и `<br>`. Мы их будем использовать в большом количестве. Результат работы следующего кода см. на рисунке 4.3:

```
<h2>A Level-2 Header</h2>
<p>A paragraph of body text with a line break here..</br>
and a second paragraph...</p>
```

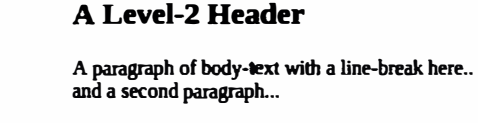


Рис. 4.3. *Заголовок второго уровня (h2) и текст*

Самый большой размер шрифта в заголовке задает тег `<h1>`, чем больше цифра в теге, тем мельче шрифт.

Теги `<div>`, `<h1>-<h6>` и `<p>` относятся к блочным элементам. Они начинаются и заканчиваются с новой строки. Еще один класс тегов — *строчные*

элементы, которые располагаются друг за другом на одной строке. К ним относятся теги изображений <img>, гиперссылок <a>, ячеек таблиц <td> и тег , который применяется к вставленному внутрь него тексту:

```
<div id="inline-examples">
   <img src="path/to/image.png" id="prettypic"> ❶
   <p>This is a <a href="link-url">link</a> to
      <span class="url">link-url</span></p> ❷
</div>
```

❶ Обратите внимание, что для <img> закрывающий тег не нужен.
❷ Ссылка и span находятся в неразрывной строке.

Часто используются теги списков: нумерованных и маркированных <ul>:

```
<div style="display: flex; gap: 50px"> ❶
   <div>
      <h3>Ordered (ol) list</h3>
      <ol>
         <li>First Item</li>
         <li>Second Item</li>
      </ol>
   </div>
   <div>
      <h3>Unordered (ul) list</h3>
      <ul>
         <li>First Item</li>
         <li>Second Item</li>
      </ul>
   </div>
</div>
```

❶ Здесь мы применили стиль CSS непосредственно в теге div. Введение в свойство display: flex см. в подразделе «Позиционирование и изменение размера контейнеров с помощью Flex» на стр. 146.
На рисунке 4.4 показаны сформированные списки.



Рис. 4.4. *Списки HTML*

В HTML также есть специальный тег `<table>`, который полезно использовать, чтобы представить сырые данные в визуализации. Приведенный ниже HTML-код создаст заголовок и строку, как на рисунке 4.5:

```
<table id="chart-data">
   <tr> ❶
      <th>Name</th>
      <th>Category</th>
      <th>Country</th>
   </tr>
   <tr> ❷
      <td>Albert Einstein</td>
      <td>Physics</td>
      <td>Switzerland</td>
   </tr>
</table>
```

❶ Строка заголовка

❷ Первая строка данных



Рис. 4.5. *HTML-таблица*

При создании веб-визуализаций из перечисленных выше тегов чаще всего используются теги для работы с текстом, которые предоставляют инструкции, информационные поля и т. д. Но основная часть нашего программирования на JavaScript, вероятно, будет направлена на создание ветвей DOM, основанных на тегах масштабируемой векторной графики (SVG) `<svg>` и `<canvas>`. В большинстве современных браузеров тег `<canvas>` также поддерживает 3D-контекст *WebGL*, позволяя встраивать визуализации *OpenGL* в страницу[1].

[1] OpenGL (Open Graphics Library) и его веб-аналог WebGL — это кросс-платформенные API для рендеринга векторной 2D и 3D-графики (подробнее см. Википедию).

Мы рассмотрим графику SVG, которая находится в центре внимания книги и является форматом, поддерживаемым библиотекой D3, в подразделе «Масштабируемая векторная графика» из главы 4. А сейчас разберемся, как добавлять стиль к блокам контента.

CSS

Каскадные таблицы стилей *CSS* (Cascading Style Sheets) — язык для описания внешнего вида веб-страницы. Хотя стили можно встраивать напрямую в HTML, это считается плохой практикой[1]. Лучше использовать идентификаторы (`id`) или классы (`class`), применяя стили через отдельный CSS-файл.

Ключевой принцип CSS — *каскадность*. При конфликте стилей с одинаковыми весами будет применен тот, который указан ближе к концу HTML-документа. Так что порядок следования в таблицах стилей играет роль. Обычно таблицу стилей загружают последней, чтобы переопределить как настройки браузера по умолчанию, так и стили, определенные сторонними библиотеками.

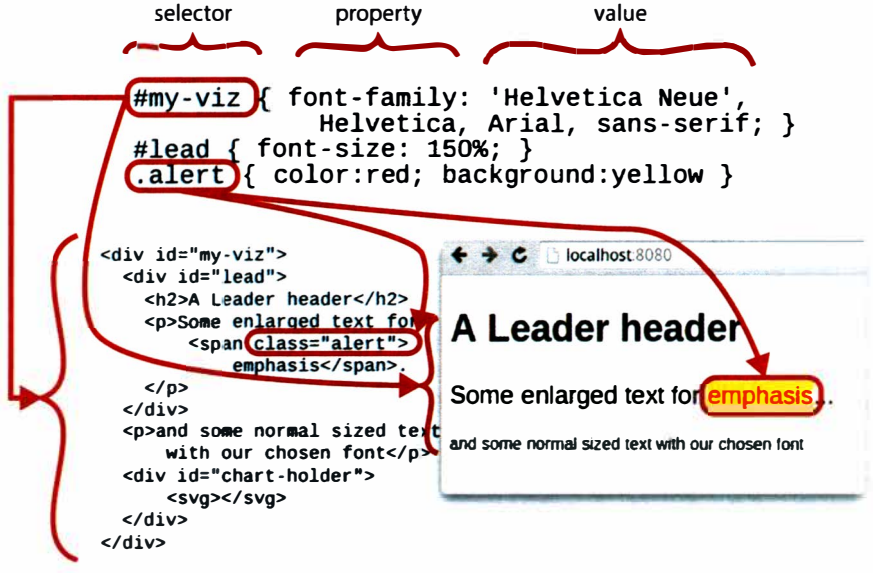


Рис. 4.6. *Оформление страницы с помощью CSS*

На рисунке 4.6 показано, как стили CSS применяются к элементам HTML. Сначала выбираем элементы, используя хештеги (#) для указания уникальных ID

[1] Это не то же самое, что программная настройка стилей, — мощная техники, позволяющей адаптировать оформление под действия пользователя.

и точки (.) для выбора членов класса. Затем определяем одну или несколько пар свойство/значение. Обратите внимание, что свойство `font-family` может содержать список альтернативных шрифтов в порядке предпочтения. На рисунке показано, что мы хотим изменить `font-family`. Заданный в браузере по умолчанию шрифт с засечками (`serif`) меняем на более современный шрифт без засечек (`sans-serif`) и выбираем `Helvetica Neue` в качестве основного варианта.

Необходимо понимать, как определяется приоритет стилей CSS, чтобы корректно их применять. Вкратце правила следующие:

1. `!important` после значения свойства CSS — самый высокий приоритет.
2. Чем специфичнее селектор, тем выше приоритет, например, идентификатор приоритетнее, чем класс.
3. Порядок объявления: если правила *1* и/или *2* имеют одинаковую специфичность, то приоритетнее будет правило, определенное последним.

В качестве примера рассмотрим `<span>` из класса `alert`:

```
<span class="alert" id="special-alert">
something to be alerted to</span>
```

Добавление следующего кода в наш файл *style.css* сделает текст класса `alert` красным и жирным:

```
.alert { font-weight:bold; color:red }
```

Затем добавим следующий код в *style.css* и, так как id специфичнее класса, color: black переопределит color: red, а `font-weight:bold` останется, так как для него id не задан:

```
#special-alert {background: yellow; color:black}
```

Чтобы принудительно задать красный цвет для alert, мы можем использовать директиву `!important`[1]:

```
.alert { font-weight:bold; color:red !important }
```

[1] Это обычно считается плохой практикой и указанием на неудачно структурированные CSS. Используйте эту директиву с особой осторожностью, она может сильно осложнить жизнь разработчику.

SKLADCHIK.ORG

Если затем после *style.css* мы добавим еще один файл *style2.css*:

```
<link rel="stylesheet" href="style.css" type="text/css" />
<link rel="stylesheet" href="style2.css" type="text/css" />
```

и *style2.css* будет содержать следующий код:

```
.alert { font-weight:normal }
```

то применится `font-weight: normal`, поскольку новый стиль класса был объявлен последним.

JavaScript

JavaScript — единственный язык для веб-программирования, чей интерпретатор включен во все современные браузеры. Чтобы сделать что-то более-менее сложное (как все современные веб-визуализации), необходимо иметь хотя бы базовые знания JavaScript. Сейчас набирает популярность TypeScript — надмножество JavaScript, добавляющее строгую типизацию. TypeScript компилируется в JavaScript, поэтому требует знания этого языка.

99 % эталонных веб-визуализаций написаны на JavaScript. Модные альтернативы со временем теряют актуальность. По сути, уверенное использование JavaScript — обязательное условие для создания интересных веб-визуализаций.

Хорошая новость для питонистов: JavaScript довольно приятен, если разобраться с его особенностями[1]. Как показано в главе 2, у JavaScript и Python много общего и переходить с одного на другой обычно просто.

Данные

Данные для веб-визуализации предоставляются веб-сервером в виде статических файлов (например, JSON или CSV-файлов) или динамически через API-интерфейсы (например, RESTful API). Данные обычно получают из базы данных на стороне сервера. Мы рассмотрим эти форматы в разделе IV.

Раньше доминировал формат XML, но современная веб-визуализация преимущественно использует JSON и, в меньшей степени, CSV или TSV-файлы.

[1] Они кратко изложены в известной книге Дугласа Крокфорда (Douglas Crockford) JavaScript: The Good Parts, в русском переводе «JavaScript: сильные стороны».

JSON (JavaScript Object Notation) — де-факто стандартный формат данных для веб-визуализации, научитесь его ценить. Он отлично работает с JavaScript, но питонистам его структура также знакома. Как мы видели в подразделе «JSON», в чтении и записи данных JSON с помощью Python нет ничего сложного. Ниже приводится маленький пример данных в формате JSON:

```
{
   "firstName": "Groucho",
   "lastName": "Marx",
   "siblings": ["Harpo", "Chico", "Gummo", "Zeppo"],
   "nationality": "American",
   "yearOfBirth": 1890
}
```

Chrome DevTools

Соревнование между JavaScript-движками в последние годы привело к значительному росту производительности и появлению усовершенствованных инструментов разработки, встроенных в браузеры. Некоторое время лидировал Firebug от Firefox, но затем вперед вырвался Chrome DevTools, постоянно добавляя новые функции. Сейчас возможности инструментальных вкладок Chrome огромны, но здесь я расскажу о двух самых полезных: *Elements* для работы с HTML и CSS и *Sources* для отладки JavaScript. Они дополняют консоль разработчика Chrome, описанную в подразделе «JavaScript».

Вкладка Elements

Чтобы открыть вкладку Elements, выберите More Tools→Developer Tools в меню справа или используйте сочетание клавиш Ctrl-Shift-I (Cmd-Option-I на Mac).

На рисунке 4.7 показана вкладка Elements в работе. С помощью лупы слева можно выбрать на странице элементы DOM и просмотреть их HTML-ветвь на левой панели. Правая панель позволяет просматривать их стили CSS, а также все слушатели событий или свойства DOM.

Одна из самых крутых фич вкладки Elements — возможность интерактивно изменять внешний вид элементов, манипулируя как их стилями, так и атрибутами[1]. Это отличный способ улучшить интерфейс визуализаций данных.

[1] Например, весьма полезна возможность менять атрибуты при работе с масштабируемой векторной графикой (SVG).

Вкладка Chrome Elements — отличный способ исследовать структуру страницы, проверяя, как позиционированы различные элементы. Таким образом хорошо усваивается, как позиционируются блоки контента с помощью свойств `position` и `float`. Наблюдение за методами роботы профессионалов — один из лучших способов научиться применять CSS.

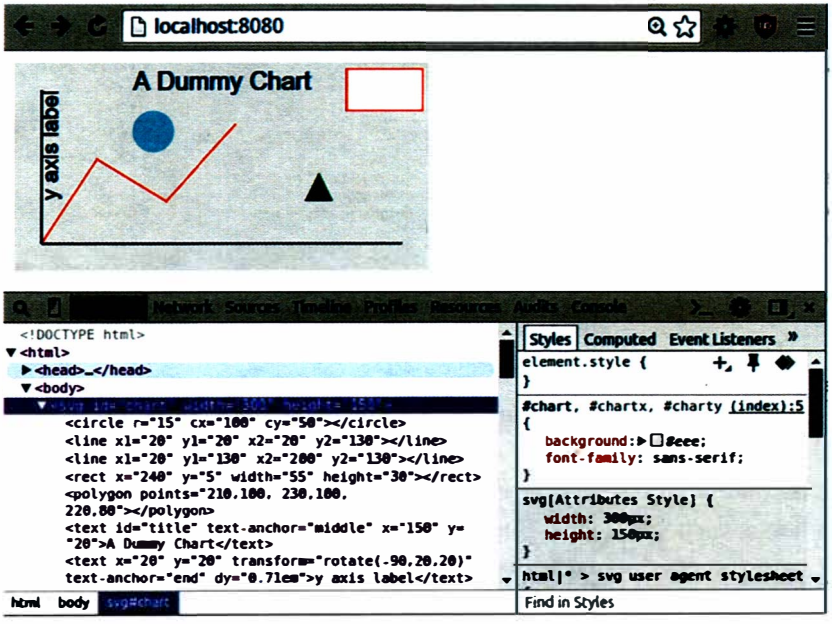


Рис. 4.7. *Chrome DevTools, вкладка Elements*

Вкладка Sources

Вкладка Sources позволяет изучить любой JavaScript-код, включенный в веб-страницу. На рисунке 4.8 показана эта вкладка в работе. На левой панели можно выбрать скрипт или же HTML-файл с JS-кодом, обрамленным тегами `<script>`. Как показано на рисунке ниже, вы можете установить точку останова на любую строку кода, загрузить страницу и при останове увидеть стек вызовов, глобальные и локальные переменные. Для точек останова можно задать параметры, определив условия их срабатывания, что полезно для пошаговой отладки конкретных скриптов. При срабатывании точки останова доступны стандартные действия: шаг с заходом (внутрь функции), шаг с выходом (из функции) и шаг с обходом функции.

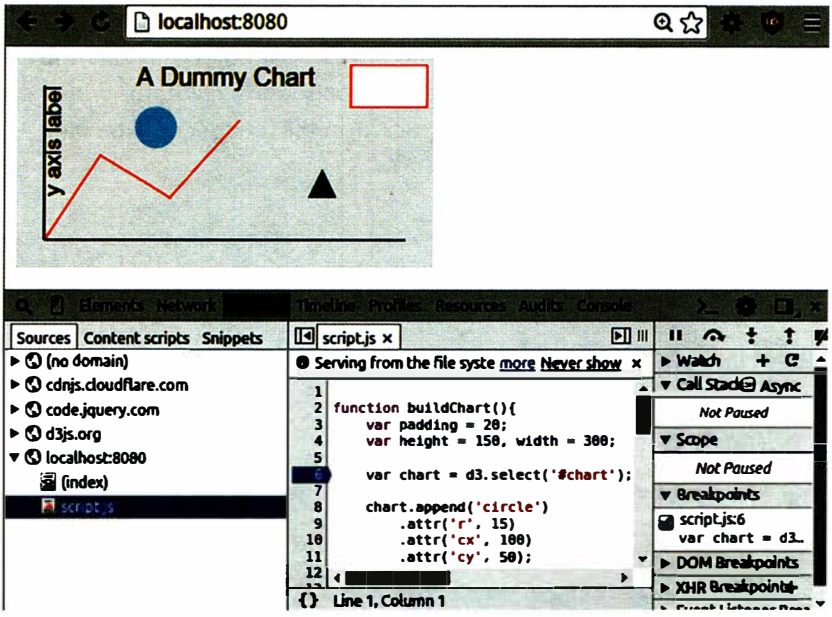


Рис. 4.8. Chrome DevTools, вкладка Sources

Вкладка Sources — фантастически полезное подспорье, она значительно снижает необходимость использовать console.log()[1] при отладке JavaScript. Благодаря Sources отладка JS-кода из головной боли превратилась почти в удовольствие.

Другие инструменты Chrome DevTools

Вкладки Chrome DevTools обладают огромными возможностями и обновляются почти ежедневно. Вы можете анализировать загрузку памяти и процессора с помощью временной шкалы и профилирования, отслеживать сетевую нагрузку и тестировать свои страницы в различных форм-факторах. Но при создании визуализаций львиную долю времени вы будете работать во вкладках Elements и Sources.

Базовая страница с плейсхолдерами

Итак, мы рассмотрели основные элементы веб-страницы, теперь соберем их вместе. Большинство веб-визуализаций начинаются с создания HTML- и CSS-каркаса с плейсхолдерами, которые будут позже заменены JavaScript-кодом и данными.

[1] Вывод сообщений на консоль — эффективный способ отслеживания потока данных через ваше приложение. Рекомендую придерживаться в этом последовательного подхода.

Сначала создадим HTML-каркас с помощью кода из примера 4-1. В него входит дерево блоков контента , определяющее три элемента структуры диаграммы: header (заголовок), main (главная часть) и sidebar (боковая панель). Сохраним этот файл как *index.html*.

***Пример 4-1.** Файл index.html, HTML-каркас нашей страницы*

```
<!DOCTYPE html>
<meta charset="utf-8">

<link rel="stylesheet" href="style.css" type="text/css" />

<body>

  <div id="chart-holder" class="dev">
    <div id="header">
      <h2>A Catchy Title Coming Soon...</h2>
      <p>Some body text describing what this visualization is
      all about and why you should care.</p>
    </div>
    <div id="chart-components">
      <div id="main">
        A placeholder for the main chart.
      </div><div id="sidebar">
        <p>Some useful information about the chart,
        probably changing with user interaction.</p>
      </div>
    </div>
  </div>

  <script src="script.js"></script>
</body>
```

Теперь добавим к HTML-каркасу немного стилей CSS. Для настройки размера, положения, цвета фона и других свойств блоков контента будем использовать атрибуты class и ID. Чтобы применить CSS, в примере 4-1 мы импортируем файл *style.css*, показанный в примере 4-2.

Пример 4-2. *Содержимое файла style.css*

```
body {
   background: #ccc;
   font-family: Sans-serif;
}
div.dev { ❶
   border: solid 1px red;
}
div.dev div {
   border: dashed 1px green;
}
div#chart-holder {
   width: 600px;
   background :white;
   margin: auto;
   font-size :16px;
}
div#chart-components {
   height: 400px;
   position: relative; ❷
}
div#main, div#sidebar {
   position: absolute; ❸
}
div#main {
   width: 75%;
   height: 100%;
   background: #eee;
}
div#sidebar {
   right: 0; ❹
   width: 25%;
   height: 100%;
}
```

❶ Этот класс `dev` — удобный способ отобразить границы видимых блоков, что полезно при создании визуализаций.

- Делаем `chart-components` родительским элементом для относительного позиционирования.
- Задает позиции `main` и `sidebar` относительно `chart-components`.
- Позиционирует данный блок вровень с правым краем `chart-components`.

Для элементов main и sidebar мы используем абсолютное позиционирование (пример 4-2). Есть различные способы позиционирования блоков контента с помощью CSS, но абсолютное позиционирование обеспечивает явный контроль их размещения, что необходимо, если вы хотите добиться нужного результата.

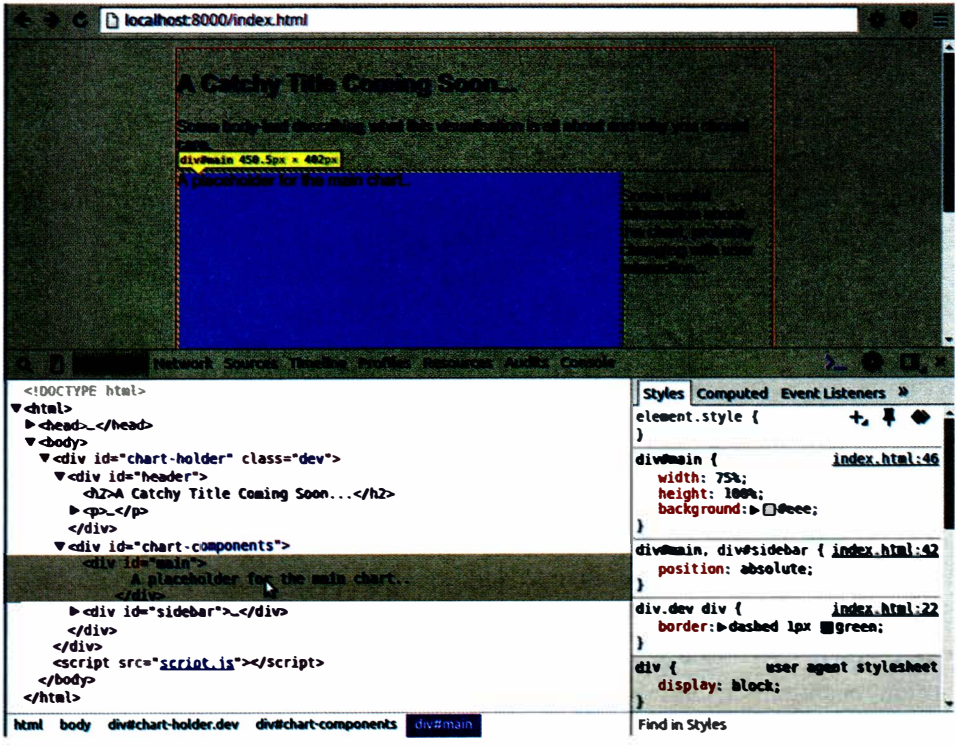


Рис. 4.9. *Создание базовой веб-страницы*

После того, как задан размер контейнера `chart-components`, его дочерние элементы `main` и `sidebar` меняют размеры и позицию относительно родителя, согласно указанным процентам. То есть любые изменения размера `chart-components` будут отражаться на его дочерних элементах.

Определив HTML и CSS, мы можем изучить каркас, запустив однострочный HTTP-сервер Python в каталоге проекта, содержащем файлы *index.html* и *style.css* (см. примеры 4-1 и 4-2), например так:

```
$ python -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 ...
```

На рисунке 4.9 показана полученная страница с открытой вкладкой Elements, отображающей DOM-дерево.

Блоки содержимого диаграммы теперь правильно размещены и имеют нужный размер. Все готово к добавлению привлекательного контента с помощью JavaScript.

Позиционирование и изменение размера контейнеров с помощью Flex

Раньше работа с позиционированием и размерами блоков (обычно контейнеров) средствами CSS было в чем-то сродни шаманству. Ситуация усугублялась кросс-браузерной несовместимостью и разногласиями по поводу внешних (margin) и внутренних (padding) отступов. Но даже без этого этого свойства CSS казались хаотичными. Достижение простых целей, вроде центрирования по горизонтали и вертикали или изменения размеров, требовали скрытых знаний о CSS из глубин Stack Overflow. Один из таких примеров — центрирование `div` по горизонтали и вертикали[1]. Все изменилось с появлением модуля CSS Flexbox, который с помощью новых эффективных свойств CSS обеспечивает практически все, что может понадобиться при работе с размерами и позиционированием.

Flexbox не вытеснил все остальные свойства CSS. Например, абсолютное позиционирование, о котором мы говорили выше, широко применяется, особенно при визуализации данных. Однако Flexbox предоставляет набор чрезвычайно действенных свойств, применение которых обеспечивает простейший, а иногда и единственный способ решения конкретной задачи по позиционированию/масштабированию. Теперь даже новички могут добиться тех же эффектов, которые раньше требовали экспертных знаний CSS. Вишенка на торте: флексбоксы отлично справляются с отображением адаптивных макетов в зависимости от размера и соотношения сторон экрана устройства. Помня об этом, рассмотрим, что мы можем сделать с помощью базового набора свойств Flexbox.

Сначала напишем немного HTML-кода, чтобы создать контейнер `div` с тремя дочерними `div` (box 1, 2, 3). Дочерние элементы будут входить в класс `box` с ID, позволяющий применять определенный стиль CSS:

[1] В этом треде приведены различные способы решения проблемы. Ни один из них не назовешь изящным.

```
<div class="container" id="top-container">
   <div class="box" id="box1">box 1</div>
   <div class="box" id="box2">box 2</div>
   <div class="box" id="box3">box 3</div>
</div>
```

Традиционные свойства CSS задают классу container красную границу, ширину и высоту (600x400). Для элементов класса box задана ширина и высота в 100 пикселей (80px плюс по 10px внутренних отступов) и зеленая граница. Новое свойство CSS для класса container — `display: flex` — делает его флекс-контейнером, позволяя гибко отображать контекст. Результат показан на рисунке 4.10 (`display: flex`), все элементы box устанавливаются в ряд, а не один под другим в колонку по умолчанию:

```
.container {
   display: flex;
   width: 600px;
   height: 400px;
   border: 2px solid red;
}
.box {
   border: 2px solid green;
   font-size: 28px;
   padding: 10px;
   width: 80px;
   height: 80px;
}
```

Дочерние элементы со свойством `flex` подгоняют размеры под доступное им пространство дочерним элементам флекс-контейнеров. Это делает элементы box гибкими, они расширяются, чтобы заполнить ряд внутри контейнера. На рисунке 4.10 (flex-direction: row) показан результат. Обратите внимание: свойство `flex` переопределяет свойство width блоков, позволяя им расширяться:

```
.box {
   /* ... */
   flex: 1;
}
```

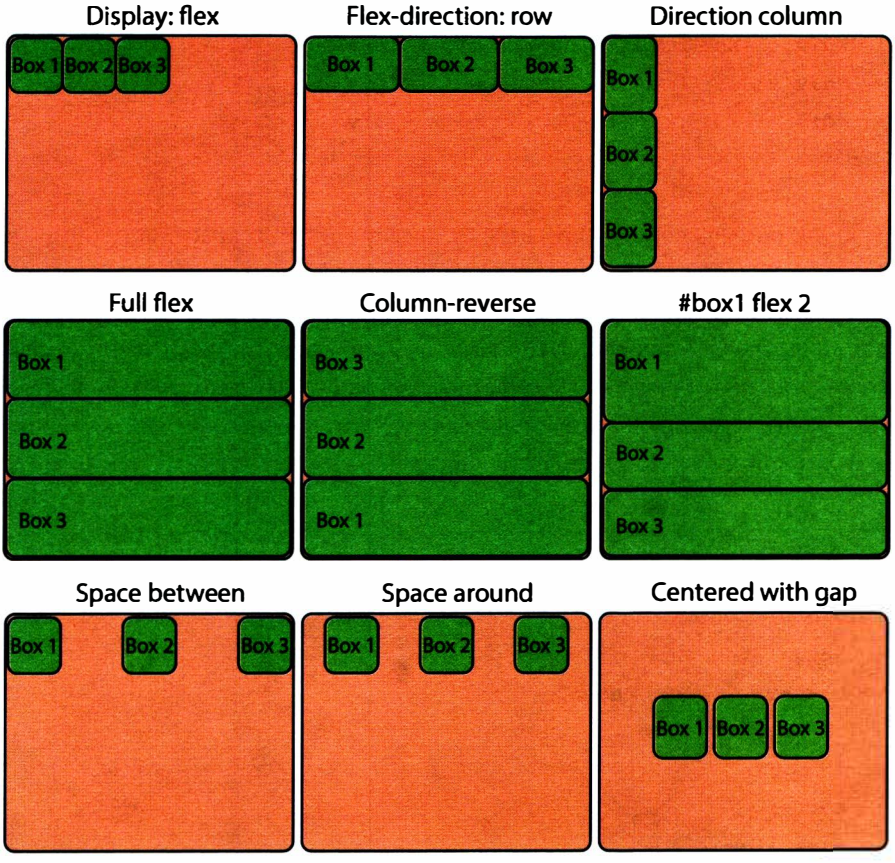


Рис. 4.10. *Позиционирование и указание размеров с помощью флексбоксов*

Свойство `flex-direction` по умолчанию `row`. При установке его в значение `column` элементы box выстраиваются в колонку, а свойство height переопределяется так, что они растягиваются на всю высоту контейнера. См. результат на рисунке 4.10 (direction column):

```
.container {
   /* ... */
   flex-direction: column;
}
```

Если удалить или закомментировать свойства width и height, то элементы box становятся полностью гибкими, способными расширяться по горизонтали и вертикали, как на рисунке 4.10 (full flex):

SKLADCHIK.ORG

```
.box {
   /* ... */
   /* width: 80px;
   height: 80px; */
   flex: 1;
}
```

Чтобы изменить порядок флексбоксов на обратный, используются `flex-direction: row-reverse` и `flex-direction: column-reverse`. См. результат изменения порядка в колонке на рисунке 4.10 (column reverse):

```
.container {
   /* ... */
   flex-direction: column-reverse;
}
```

Значение свойства `flex` для элементов box — значение их размеров. Изначально значение для всех box — 1, то есть они все одного размера. Если присвоить box1 значение 2, он займет половину (2 / (1 + 1 + 2)) доступного пространства в указанном направлении ряда или столбца. См. результат увеличения значения свойства flex для `box1` на рисунке 4.10 (#box1 flex 2):

```
#box1 {
   flex: 2;
}
```

Если мы вернем высоту и ширину в 100 пикселей (с учетом внутренних отступов) для элементов box и удалим их свойство `flex`, то сможем оценить гибкость позиционирования в полном объеме. Также нужно закомментировать директиву flex для `box1`:

```
.box {
   width: 80px;
   height: 80px;
   /* flex: 1; */
}
#box1 {
   /* flex: 2; */
}
```

Благодаря фиксированным размерам контента можно применить ряд свойств флекс-контейнера, которые позволяют точно размещать контент. Раньше для такого рода манипуляций использовались всевозможные хитрые CSS-хаки. Сначала давайте распределим флекс-элементы box равномерно по горизонтали. Это может сделать свойство `justify-content` со значением `space-between`. См. результат на рисунке 4.10:

```
.container {
  /* ... */
  flex-direction: row;
  justify-content: space-between;
}
```

Существует дополнение к `space-between` — значение `space-around`, которое размещает контент равномерно, добавляя слева и справа равные отступы от границ контейнера. См. результат на рисунке 4.10 (space around):

```
.container {
  /* ... */
  justify-content: space-around;
}
```

Комбинируя свойства `justify-content` и `align-items`, можно достичь заветной цели позиционирования CSS — центрирования контента по вертикали и горизонтали. Добавим по 20 пикселей между элементами box, используя свойство `gap`:

```
.container {
  /* ... */
  gap: 20px;
  justify-content: center;
  align-items: center;
}
```

На рисунке 4.10 (с выравниванием по центру, с интервалами `gap`) контент расположен прямо по центру контейнера.

Еще одно преимущество Flexbox — полная рекурсивность. Блоки `div` могут как быть гибкими контейнерами (то есть *иметь* свойство `display: flex`), так и *быть* гибким контентом. Это значительно упрощает создание сложных

макетов размещения контента. Устроим небольшую демонстрацию вложенных флексбоксов, чтобы прояснить этот вопрос.

С помощью HTML-кода создадим вложенное дерево блоков box (включая главный контейнер). Присвоим каждому box и контейнеру ID и класс:

```
<div class="main-container">

   <div class="container" id="top-container">
      <div class="box" id="box1">box 1</div>
      <div class="box" id="box2">box 2</div>
   </div>

   <div class="container" id="middle-container">
      <div class="box" id="box3">box 3</div>
   </div>

   <div class="container" id="bottom-container">
      <div class="box" id="box4">box 4</div>
      <div class="box" id="box5">
         <div class="box" id="box6">box 6</div>
         <div class="box" id="box7">box 7</div>
      </div>
   </div>

</div>
```

Следующий CSS-код задает для класса main container высоту 800 пикселей (по умолчанию он заполнит всю доступную ширину), а также свойства display: flex и flex-direction: column, выстраивая контент вертикально, сверху вниз.

Три контейнера, которые выстраиваются в стопку, являются одновременно флекс-элементами и флекс-контейнерами для своего контента. У элементов класса box красная граница и не определена ширина и высота, то есть они полностью гибкие. По умолчанию их значение flex — единица.

У среднего контейнера (middle container) фиксированная ширина (width 66 %) и он отцентрован по горизонтали (`justify-content: center`).

У bottom container значение flex равно 2, поэтому он вдвое выше элементов одного с ним уровня иерархии. Он состоит из двух box с равным значением, один из которых (box 5) содержит два элемента box, расположенных вертикально (`flex-direction: column`). Довольно сложная верстка (см. рисунок 4.11)

достигается с помощью небольшого количества CSS и легко адаптируется путем изменения нескольких свойств флекс-контейнера:

```
.main-container {
   height: 800px;
   padding: 10px;
   border: 2px solid green;
   display: flex;
   flex-direction: column;
}

.container {
   flex: 1;
   display: flex;
}

.box {
   flex: 1;
   border: 2px solid red;
   padding: 10px;
   font-size: 30px;
}

#middle-container {
   justify-content: center;
}

#box3 {
   width: 66%;
   flex: initial;
}

#bottom-container {
   flex: 2;
}

#box5 {
   display: flex;
   flex-direction: column;
}
```

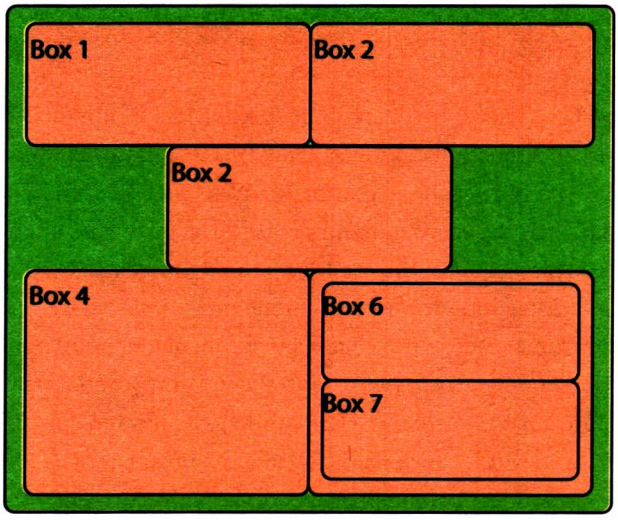


Рис. 4.11. *Вложенные флексбоксы*

Flexbox предоставляет очень мощные возможности изменения размера и позиционирования для HTML-контента, который зависит от размера контейнера и может быть легко адаптирован. Если вы хотите расположить контент в столбец, а не в ряд, достаточно изменить единственное свойство. Для более точного контроля над позиционированием и размером существует двумерная сетка CSS Grid Layout, но я советую для начала сосредоточиться на Flexbox. Вы сразу начнете получать наибольшую отдачу от изучения CSS. Дополнительные примеры Flexbox см. в статье на https://css-tricks.com/snippets/css/a-guide-to-flexbox/.

Заполнение плейсхолдеров контентом

Итак, блоки контента определяются с помощью HTML, позиционируются — CSS, а для создания интерактивных диаграмм, меню, таблиц и тому подобного в современной визуализации данных используется JavaScript. Основные способы создания визуального контента в современном браузере (помимо тегов изображений или мультимедиа) перечислены ниже:

— масштабируемая векторная графика (Scalable Vector Graphics, SVG) с использованием специальных HTML-тегов;
— рисование двухмерного графического контекста на пиксельном холсте (`canvas`);
— рисование на `canvas` трехмерного WebGL контекста с использованием подмножества команд OpenGL;

— создание анимации, графических примитивов и многого другого средствами современного CSS.

Поскольку SVG — язык, который поддерживает D3, крупнейшая JS-библиотека визуализации, многие из замечательных веб-визуализаций созданы с его использованием, (например те, что сделаны *New York Times*). Если не вдаваться в детали, SVG вам, скорее всего, подойдет, если только вы не планируете создать визуализацию со множеством (более 1000) движущихся элементов или использовать специфическую библиотеку на основе `canvas`.

Используя векторы вместо пикселей для отрисовки своих примитивов, SVG, как правило, создает более чистую плавно масштабируемую графику. К тому же SVG отлично работает с текстом, что критично для визуализаций. Еще одно ключевое преимущество SVG — взаимодействие с пользователем (например, наведение курсора или клик) нативно поддерживается браузером, так как является частью стандартной обработки событий DOM[1]. Завершающий аргумент: поскольку графические компоненты построены на DOM, вы можете проверять и изменять их с помощью инструментов разработчика, встроенных в браузер (см. «Chrome DevTools»). Все это может значительно упростить отладку и совершенствование визуализаций по сравнению с поиском ошибок в довольно таки «черном ящике» `canvas`-графики.

Графические `canvas`-контексты становятся незаменимы, когда нужно выйти за рамки графических примитивов (окружностей и линий), например, при работе с растровыми изображениями (PNG, JPG). Обычно технология `canvas` гораздо производительнее SVG, поэтому элементы с большим количеством анимированных частей[2] лучше рендерить на canvas. Если вы хотите реализовать честолюбивые замыслы или просто выйти за пределы 2D-графики, можно задействовать мощность современных видеокарт через специальный контекст `canvas` — WebGL на основе OpenGL. Однако учтите: то, что в SVG реализуется просто (например, клик по визуальному элементу), в canvas часто требует расчета координат курсора вручную, что добавляет целый уровень сложности.

Визуализация данных о лауреатах Нобелевской премии, которую мы реализуем в конце этой книги, в основном выстроена с помощью *D3.js*, так что мы будем фокусироваться на SVG-графике. Умение использовать SVG — фундаментальное требование современной веб-визуализации, так что познакомимся с основами этой графики.

---

[1] При использовании графического canvas-контекста, как правило, приходится придумывать собственную обработку событий.

[2] Это число зависит от браузера и возрастает со временем, но, как правило, SVG начинает работать с перебоями при количестве объектов от одной до двух тысяч.

Масштабируемая векторная графика

Создание SVG начинается с корневого тега <svg>. Все графические элементы, такие как отдельные окружности и линии, а также их группы, определяются на этой ветви DOM-дерева. В примере 4-3 показан небольшой SVG-контекст, который мы будем использовать и далее — светло-серая прямоугольная область с ID chart. Мы также подключим JS-файл *script.js* из каталога проекта и библиотеку D3, скачав ее с d3js.org.

***Пример 4-3.** Базовый SVG-контекст*

```
<!DOCTYPE html>
<meta charset="utf-8">
<!-- A few CSS style rules -->
<style>
   svg#chart {
      background: lightgray;
   }
</style>

<svg id="chart" width="300" height="225">
</svg>

<!-- Third-party libraries and our JS script. -->
<script src="http://d3js.org/d3.v7.min.js"></script>
<script src="script.js"></script>
```

Наша SVG-область для рисования готова, давайте что-нибудь нарисуем.

Элемент <g>

С помощью элемента <g> можно группировать фигуры внутри элемента <svg> Как мы увидим в подразделе «Работа с группами» на стр. 165, можно одновременно изменять положение, масштаб, уровень прозрачности и прочие свойства всех фигур, включенных в группу.

Круги

Создание SVG-визуализаций, от простой статичной столбчатой диаграммы до полнофункциональных интерактивных географических шедевров, требует

объединять элементы из ограниченного набора графических примитивов: линий, кругов и очень мощного path («путь»).

У каждого из этих элементов есть собственный тег DOM, который будет обновляться по мере изменения элемента[1]. Например, его атрибуты *x* и *y* будут изменяться, чтобы отражать любые преобразования в пределах его `<svg>`-контекста или контекста его группы (`<g>`).

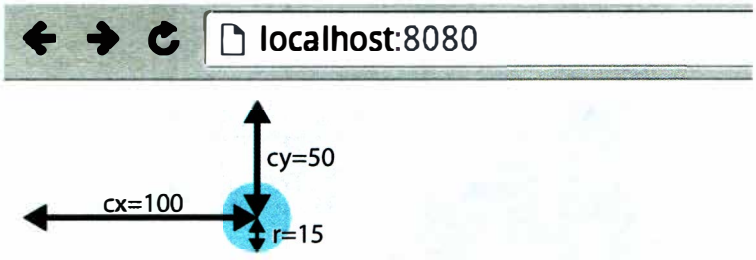


Рис. 4.12. *SVG: круг*

Добавим круг в наш `<svg>` контекст:

```
<svg id="chart" width="300" height="225">
   <circle r="15" cx="100" cy="50"></circle>
</svg>
```

Немного CSS, чтобы задать цвет заливки круга:

```
#chart circle{ fill: lightblue }
```

Результат см. на рисунке 4.12. Обратите внимание, что координата *y* измеряется от верхнего края контейнера `<svg>` `'#chart'`, это общепринятое соглашение в графике.

Теперь рассмотрим, как применять стили к элементам SVG.

[1] Чтобы видеть обновление атрибутов тегов в режиме реального времени, вы должны уметь пользоваться инструментами разработчика, встроенными в браузер.

Применение стилей CSS

Круг на рисунке 4.12 залит голубым цветом с помощью CSS-правил:

```
#chart circle{ fill: lightblue }
```

В современных браузерах можно задавать с помощью CSS большинство визуальных стилей SVG, включая заливку (`fill`), цвет и толщину обводки (`stroke` и `stroke-width`) и уровень прозрачности (`opacity`). Создадим толстую полупрозрачную зеленую линию (с идентификатором `total`) с помощью следующего кода CSS:

```
#chart line#total {
   stroke: green;
   stroke-width: 3px;
   opacity: 0.5;
}
```

Также можно задать стили как атрибуты SVG-тегов, хотя CSS предпочтительнее:

```
<svg>
<circle r="15" cx="100" cy="50" fill="lightblue"></circle>
</svg>
```



Есть некоторые моменты недопонимания и ловушки относительно того, какие свойства элементов SVG можно настроить с помощью CSS, а какие нет. Спецификация SVG различает свойства и атрибуты элемента, причем допустимые стили CSS правильно передавать в свойства. Вы можете исследовать допустимые свойства CSS во вкладке Chrome Elements с помощью автозаполнения. Будьте готовы к некоторым сюрпризам. Например, цвет SVG-текста задается с помощью `fill`, а не `color`.

Есть несколько возможностей выбора цвета для `fill` и `stroke`:

— названия цветов, принятые в HTML, например lightblue;
— hex-коды HTML (#RRGGBB), например белый имеет код #FFFFFF;

— формат RGB, например красный = rgb(255, 0, 0);
— значение RGBA, где A означает альфа-канал (от 0 до 1), позволяющий управлять прозрачностью. Например, полупрозрачный синий — это rgba(0, 0, 255, 0.5).

Кроме настройки альфа-канала в значениях RGBA, можно изменять прозрачность SVG-элементов с помощью свойства `opacity`. Библиотека D3 часто использует opacity в анимации.

Толщина обводки (stroke-width) по умолчанию измеряется в пикселях (px), но можно использовать пункты (pt).

Линии, прямоугольники и многоугольники

Добавим еще несколько элементов к нашей диаграмме, чтобы получился рисунок 4.13.

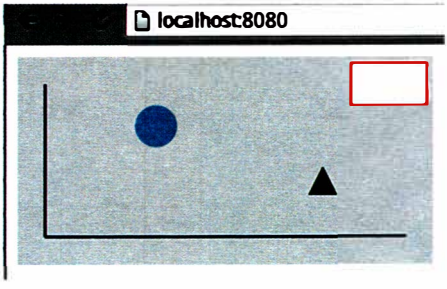


Рис. 4.13. *Добавление элементов к учебной диаграмме*

Сначала с помощью тега `<line>` добавим пару простых осевых линий. Позиции линий определяются начальной координатой (x1, y1) и конечной координатой (x2, y2):

```
<svg>
   <line x1="20" y1="20" x2="20" y2="130"></line>
   <line x1="20" y1="130" x2="280" y2="130"></line>
</svg>
```

Также добавим поле для легенды в правом верхнем углу, используя SVG-прямоугольник. Прямоугольники определяются координатами *x* и *y* относительно родительского контейнера, а также шириной и высотой:

SKLADCHIK.ORG

```
<svg>
   <rect x="240" y="5" width="55" height="30"></rect>
</svg>
```

С помощью тега <polygon>, который принимает список пары координат, можно создавать неправильные многоугольники. Создадим треугольный маркер в правом нижней углу на нашей диаграмме:

```
<svg>
   <polygon points="210,100, 230,100, 220,80"></polygon>
</svg>
```

Применим к элементам CSS-стили:

```
#chart circle {fill: lightblue}
#chart line {stroke: #555555; stroke-width: 2}
#chart rect {stroke: red; fill: white}
#chart polygon {fill: green}
```

После создания графических примитивов рассмотрим, как добавить текст в нашу учебную диаграмму.

Текст

Одно из ключевых преимуществ SVG перед растровым контекстом canvas — возможности обработки теста. Векторный текст выглядит более четко, чем его пиксельные аналоги, и к тому же его можно плавно масштабировать. Для текста можно менять свойства обводки и заливки, как и для других элементов SVG.

Добавим текст в учебную диаграмму и подпишем ось Y (см. рисунок 4.14).

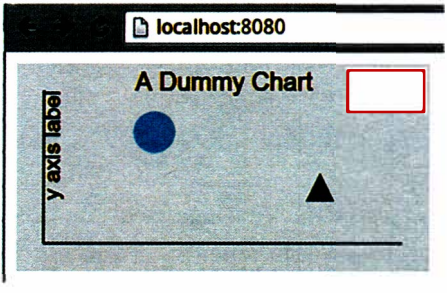


Рис. 4.14. *Немного SVG-текста*

Место текста задается координатами *x* и *y*. Важное свойство `text-anchor` предопределяет положение текста относительно заданной точки на оси X. Другие варианты: `start`, `middle` и `end`. По умолчанию используется `start`.

Используем свойство `text-anchor`, чтобы отцентрировать заголовок диаграммы. Установим координату *x* равной половине ширины диаграммы и затем установим `text-anchor` в `middle`:

```
<svg>
   <text id="title" text-anchor="middle" x="150" y="20">
      A Dummy Chart
   </text>
</svg>
```

Текст, как и все примитивы SVG, можно масштабировать и вращать. Чтобы добавить подпись к оси Y, нужно повернуть текст вертикально (пример 4-14). По соглашению вращение выполняется на заданное число градусов по часовой стрелке, поэтому для поворота против часовой стрелки нужно указать -90 градусов. По умолчанию повороты выполняются вокруг точки (0,0) контейнера элемента (`<svg>` или группы `<g>`). Мы будем поворачивать текст вокруг его собственной позиции, поэтому сначала переместим точку вращения с помощью дополнительных аргументов функции `rotate`. Также установим `text-anchor` в конец строки `y axis label`, чтобы она поворачивалась относительно конца.

Пример 4-4. *Поворот текста*

```
<svg>
   <text x="20" y="20" transform="rotate(-90,20,20)"
      text-anchor="end" dy="0.71em">y axis label</text>
</svg>
```

В примере 4-4 мы используем атрибут текста `dy`, который наряду с `dx` применяется для точной настройки позиции текста. Здесь он помогает опустить текст так, чтобы при вращении против часовой стрелки он оказался справа от оси Y.

К элементам SVG-текста можно применять стили CSS. Здесь мы устанавливаем `font-family` диаграммы в значение `sans-serif`, а значение `font-size` увеличиваем до `16px`, используя `title` ID:

```
#chart {
   background: #eee;
   font-family: sans-serif;
}
#chart text{ font-size: 16px }
#chart text#title{ font-size: 18px }
```

Обратите внимание, что свойства `font-family` и `font-size` наследуются элементами `text` из CSS диаграммы, так что не требуется задавать их для каждого элемента `text`.

Пути

Пути (paths) — самые сложные и мощные элементы SVG, которые позволяют создавать многокомпонентные контуры из прямых и кривых. Эти контуры можно замкнуть и залить цветом, что позволяет получить практически любую желаемую форму. Для примера добавим ломаную линию в учебную диаграмму, как на рисунке 4.15.

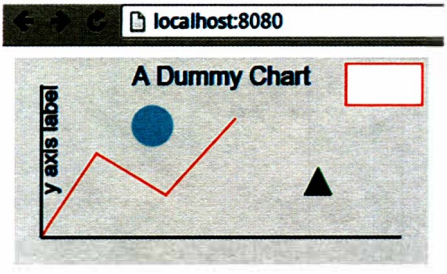


Рис. 4.15. *Красная линия пути, выходящая из начала координат*

Красный путь на рисунке 4.15 создан следующим кодом SVG:

```
<svg>
   <path d="M20 130L60 70L110 100L160 45"></path>
</svg>
```

В теге `path` атрибут `d` определяет последовательность операций для построения красной линии. Разберем их по порядку:

— "M20 130" — перейти к точке с координатами (20, 130);
— "L60 70" — нарисовать линию к точке с координатами (60, 70);

— "L110 100" — нарисовать линию к точке с координатами (110, 100);
— "L160 45" — нарисовать линию к точке с координатами (160, 45).

Можно представить `d` как набор инструкций для пера, чтобы оно двигалось к заданным точкам, при этом `M` поднимает перо над холстом.

Нужно добавить немного CSS. Обратите внимание, что `fill = none`. В противном случае путь будет замкнут линией, нарисованной из его конца в начальную точку, а полученные при этом замкнутые области будут залиты черным цветом по умолчанию:

```
#chart path {stroke: red; fill: none}
```

Кроме `M` (moveTo) и `L` (lineTo) путь имеет ряд других команд, чтобы рисовать дуги, кривые Безье и тому подобное. Кривые и дуги SVG часто используются для создания визуализаций, многие из которых сделаны с помощью библиотеки D3[1]. На рисунке 4.16 показаны эллиптические дуги SVG, созданные с помощью следующего кода:

```
<svg id="chart" width="300" height="150">
<path d="M40 40
         A30 40 ❶
         0 0 1  ❷
         80 80
         A50 50 0 0 1 160 80
         A30 30 0 0 1 190 80
">
</svg>
```

❶ Переместившись в точку координат (40, 40), рисуем эллиптическую дугу с радиусом (x=30, y=40) и конечной точкой (80, 80.)

❷ Первый флаг (0) устанавливает вращение по оси X, в данном случае это ноль по умолчанию. См. визуальную демонстрацию на сайте разработчиков Mozilla. Второй и третий флаги (0,1) — это `large-arc-flag`, указывающий, какую часть дуги использовать: большую (1) или меньшую (0), и `sweep-flag`, который определяет какой из двух возможных эллипсов, определенных начальной и конечной точками, использовать.

[1] Прекрасный пример — хордовая диаграмма Майка Бостока, в которой использована функция `d3.chord()`.

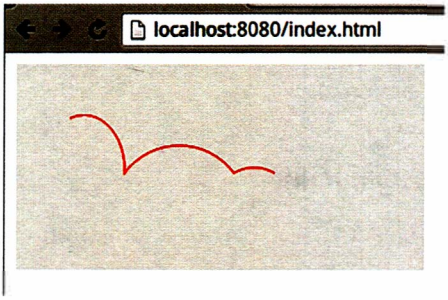


Рис. 4.16. *Эллиптические дуги SVG*

Основные флаги, которые используются в эллиптической дуге (`large-arc-flag` и `sweep-flag`), как и большинство геометрических понятий, проще показать, чем объяснить. На рисунке 4.17 показан эффект от изменения значений флагов при сохранении относительных начальных и конечных точек.

```
<svg id="chart" width="300" height="150">
   <path d="M40 80
            A30 40 0 0 1 80 80
            A30 40 0 0 0 120 80
            A30 40 0 1 0 160 80
            A30 40 0 1 1 200 80
">
</svg>
```

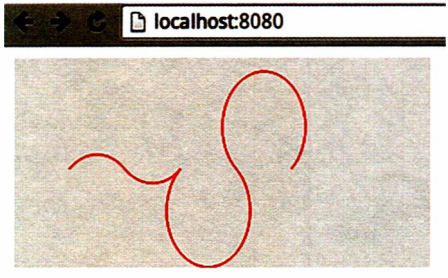


Рис. 4.17. *Изменение флагов эллиптических дуг*

Помимо линий и дуг, элемент `path` предлагает ряд кривых Безье, в том числе квадратичные, кубические и их сочетания. Приложив некоторые усилия, с их помощью можно создать любой линейный путь. См. отличный обзор с хорошими иллюстрациями на SitePoint.

Полный список элементов `path` и их аргументов см. на ресурсе World Wide Web Consortium (W3C). Более подробный обзор см. на jenkov.com во введении от Якоба Дженкова (Jakob Jenkov).

Масштабирование и вращение

Поскольку все элементы SVG векторные, их можно преобразовывать с помощью геометрических операций. Чаще всего используются `rotate`, `translate` и `scale`, но также можно применять наклон по одной оси с помощью `skewX` и `skewY` или использовать многоцелевую матрицу трансформации (*matrix*).

Рассмотрим наиболее популярные преобразования на примере набора идентичных прямоугольников. Преобразование прямоугольников, показанное на рисунке 4.18, получено следующим образом:

```
<svg id="chart" width="300" height="150">
   <rect width="20" height="40" transform="translate(60, 55)"
      fill="blue"/>
   <rect width="20" height="40" transform="translate(120, 55),
      rotate(45)" fill="blue"/>
   <rect width="20" height="40" transform="translate(180, 55),
      scale(0.5)" fill="blue"/>
   <rect width="20" height="40" transform="translate(240, 55),
      rotate(45),scale(0.5)" fill="blue"/>
</svg>
```

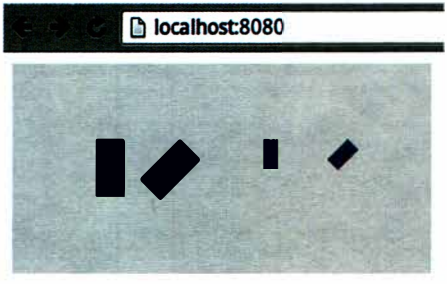


Рис. 4.18. *Несколько SVG-трансформаций: rotate(45), scale(0.5), scale(0.5), затем rotate(45)*



Важен порядок, в котором применяется трансформация. Поворот на 45 градусов по часовой стрелке с последующим перемещением вдоль оси X переместит элемент на юго-восток, а при обратном порядке операций он переместится влево, а затем повернется.

Работа с группами

Часто при создании визуализаций полезно сгруппировать элементы. Два конкретных случая использования:

— когда требуется локальная система координат (например, у вас есть текстовая метка для иконки, и вы хотите указать позицию метки относительно иконки, а не всей области рисования `<svg>`);

— когда нужно применить масштабирование и/или поворот к подмножеству визуальных элементов.

Для группирования элементов в SVG есть тег `<g>`, который можно считать мини-холстом внутри основного холста `<svg>`. Группы могут быть вложенными, что позволяет создавать очень гибкие геометрические отображения[1].

Пример 4-5 демонстрирует группировку фигур в центре холста (рисунок 4.19). Обратите внимание, что позиция элементов `circle`, `rect` и `path` задана относительно перемещаемой группы.

***Пример 4-5.** Группирование фигур SVG*

```
<svg id="chart" width="300" height="150">
  <g id="shapes" transform="translate(150,75)">
    <circle cx="50" cy="0" r="25" fill="red" />
    <rect x="30" y="10" width="40" height="20" fill="blue" />
    <path d="M-20 -10L50 -10L10 60Z" fill="green" />
    <circle r="10" fill="yellow">
  </g>
</svg>
```



***Рис. 4.19.** Группирование фигур с помощью SVG-тега `<g>`*

[1] Например, группа «тело» содержит группу «рука», которая в свою очередь содержит группу «кисть руки», включающую в себя элементы «палец».

Применение трансформаций к группе влияет на все ее элементы. На рисунке 4.20 показан результат масштабирования группы с рисунка 4.19 с коэффициентом 0.75 и последующего ее поворота на 90 градусов, для чего переписали атрибут transform следующим образом:

```
<svg id="chart" width="300" height="150">
   <g id="shapes",
      transform = "translate(150,75),scale(0.5),rotate(90)">
      ...
</svg>
```



Рис. 4.20. *Трансформация SVG-группы*

Создание слоев и прозрачность

Порядок добавления SVG-элементов в DOM-дерево важен: элементы, добавленные позднее, имеют приоритет и располагаются поверх других слоев. Например, на рисунке 4.19 треугольный путь перекрывает красный круг и синий прямоугольник, а он сам перекрывается желтым кругом.

Изменение порядка элементов DOM — важная часть визуализации с помощью JavaScript. Например, метод `insert` в D3 позволяет размещать SVG-элемент перед существующим.

Прозрачностью элемента можно управлять через альфа-канал цвета `rgba (R, G, B, A)` или более удобное свойство `opacity`. Оба параметра задаются с помощью CSS. Для наложенных элементов непрозрачность суммируется, как показано на примере цветового треугольника (рисунок 4.21), созданного с помощью следующего SVG-кода:

```
<style>
   #chart circle { opacity: 0.33 }
</style>
```

```
<svg id="chart" width="300" height="150">
  <g transform="translate(150, 75)">
    <circle cx="0" cy="-20" r="30" fill="red"/>
    <circle cx="17.3" cy="10" r="30" fill="green"/>
    <circle cx="-17.3" cy="10" r="30" fill="blue"/>
  </g>
</svg>
```

В этом примере элементы SVG вручную написаны внутри HTML, но в реальных проектах визуализации данных их добавление, как правило, автоматизировано. Соответственно, базовый процесс работы D3 заключается в добавлении SVG-элементов в визуализацию и указании их атрибутов и свойств через данные.



Рис. 4.21. *Управление прозрачностью в SVG*

Создание SVG с помощью JavaScript

То, что SVG-графика описывается тегами DOM, дает ряд преимуществ перед таким «черным ящиком», как `<canvas>`. Например, это позволяет не-программистам создавать или адаптировать графику и значительно упрощает отладку.

Очень многие SVG-элементы для веб-визуализации данных создаются с помощью JavaScript-библиотек вроде D3. Результаты выполнения скриптов можно изучать на вкладке Elements браузера (см. Chrome DevTools на стр. 92) — это отличный способ дорабатывать и отлаживать проект (например, устранять визуальный глитч).

Давайте бросим взгляд на то, что нас ожидает: с помощью D3 создадим несколько красных кругов на SVG-холсте. Размеры холста и кругов задаются объектом `data`, который передается в функцию `chartCircles`.

Подготовим HTML плейсхолдер для элемента `<svg>`:

```
<!DOCTYPE html>
<meta charset="utf-8">

<style>
   #chart { background: lightgray; }
   #chart circle {fill: red}
</style>

<body>
   <svg id="chart"></svg>
   <script src="http://d3js.org/d3.v7.min.js"></script>
   <script src="script.js"></script>
</body>
```

Используя плейсхолдер для SVG-элемента `chart`, команды D3, написанные в файле *script.js*, преобразуют данные в круги, рассеянные по холсту (см. рисунок 4.22):

```
// script.js

var chartCircles = function(data) {

   var chart = d3.select('#chart');
   // Set the chart height and width from data
   chart.attr('height', data.height).attr('width', data.width);
   // Create some circles using the data
   chart.selectAll('circle').data(data.circles)
      .enter()
      .append('circle')
      .attr('cx', function(d) { return d.x })
      .attr('cy', d => d.y)
      .attr('r', d => d.r);
};

var data = {
   width: 300, height: 150,
   circles: [
      {'x': 50, 'y': 30, 'r': 20},
      {'x': 70, 'y': 80, 'r': 10},
```

SKLADCHIK.ORG

```
        {'x': 160, 'y': 60, 'r': 10},
        {'x': 200, 'y': 100, 'r': 5},
    ]
};
chartCircles(data);
```

❶ Эта краткая анонимная стрелочная функция эквивалентна развернутой записи функции на предыдущей строке. D3 активно использует такие функций для доступа к свойствам привязанных объектов данных, что делает синтаксис более лаконичным.

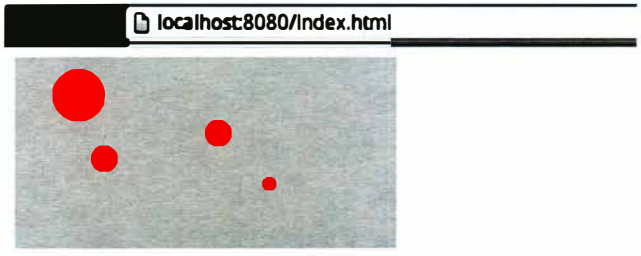


***Рис. 4.22.** Круги, созданные через D3*

Мы рассмотрим, как D3 творит свою магию, в главе 17. Пока подведем итоги этой главы.

Резюме

В этой главе представлены базовые навыки современной веб-разработки, необходимые начинающему специалисту по визуализации данных. В главе показано, как различные элементы веб-страниц (HTML-разметка, CSS-таблицы стилей, JavaScript и медиафайлы) передаются по протоколу HTTP в браузер, объединяющий их в веб-страницу, которую видит пользователь. Мы рассмотрели, как с помощью HTML-тегов, таких как `div` и `p`, описывать блоки контента, а затем стилизовать и позиционировать их, используя CSS. Мы также рассмотрели ключевые инструменты разработчика из браузера Chrome — вкладки Elements и Sources. В завершение мы познакомились с основами SVG, языка, на котором создается большая часть современных веб-визуализаций данных. Мы расширим эти навыки, когда наш тулчейн дойдет до визуализации с помощью D3, и попутно получим новые.

РАЗДЕЛ II

Получение данных

В этом разделе мы начнем знакомство с тулчейном визуализации (см. рисунок II-1). Две главы раздела расскажут о методах сбора данных, если те не предоставлены заранее.

Глава 5 посвящена извлечению данных из интернета с помощью Python-библиотеки Requests для загрузки веб-файлов и работы с RESTful API. Мы также увидим, как использовать две библиотеки Python, упрощающие взаимодействие со сложными API, а именно Twitter (через Tweepy) и Google Sheets (Google Таблицы). В конце главы приведен пример базового веб-скрейпинга с использованием библиотеки Beautiful Soup.

В главе 6 мы получим для дальнейшей визуализации набор данных о лауреатах Нобелевской премии с помощью Scrapy — веб-скрейпера промышленного уровня, написанного на Python. Получив этот набор *«грязных»* данных, мы будем готовы перейти к разделу III.

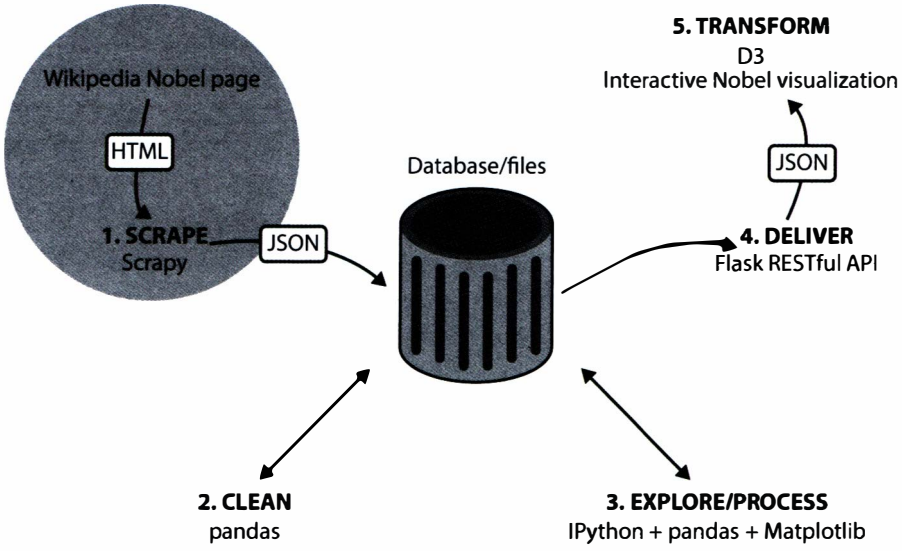


Рис. II-1. *Наш тулчейн для визуализации: получение данных*



Исходный код для этого раздела доступен в репозитории книги на GitHub: https://github.com/Kyrand/dataviz-with-python-and-js-ed-2.

Получение данных из интернета с помощью Python

Фундаментальная часть навыков для специалиста по визуализации данных — умение получить нужный набор данных максимально чистым. Иногда для анализа предоставляют готовый набор чистых данных, но чаще всего приходится самому искать и/или очищать данные.

В современных условиях данные добывают чаще всего из интернета. Сделать это можно различными способами, и Python предлагает библиотеки, которые облегчают сбор данных.

Основные способы получения данных из интернета:

— получить файл сырых данных в стандартном формате (например, JSON или CSV) через HTTP;
— получить данные с помощью специальных API;
— скрейпить данные с веб-страниц, то есть получать их через HTTP, и парсить их локально для извлечения необходимых данных.

Мы рассмотрим все три способа по очереди, но для начала познакомимся с лучшей HTTP-библиотекой Python — Requests.

Получение данных из интернета с помощью библиотеки Requests

Как было сказано в главе 4, файлы, которые используют браузеры для построения веб-страниц, передаются через протокол HTTP, разработанный Тимом Бернерс-Ли. Чтобы получать веб-контент для парсинга, необходимо отправлять HTTP-запросы.

Согласование контента в HTTP-запросах — важная составляющая любого языка общего назначения, но раньше получать веб-страницы с помощью Python было утомительно. В старой библиотеке urllib2 был неудобный API, с которым

непросто было разобраться. Ситуацию изменила созданная Кеннетом Рейцемом (Kenneth Reitz) библиотека Requests.

Она заметно упростила работу с HTTP и быстро стала самой популярной HTTP-библиотекой Python.

Requests не входит в стандартную библиотеку Python[1], но поставляется в пакете Anaconda (См. главу 1). Если вы не используете Anaconda, установите Requests с помощью `pip`:

```
$ pip install requests
Downloading/unpacking requests
...
Cleaning up...
```

При использовании Requests в версии Python ниже 2.7.9 (настоятельно рекомендую перейти на Python 3+) могут возникать предупреждения SSL (Secure Sockets Layer) https://oreil.ly/8D08s. Для их устранения[2] обновите библиотеки SSL:

```
$ pip install --upgrade ndg-httpsclient
```

После установки Requests перейдем к первой задаче главы — загрузке файлов с сырыми данными из интернета.

Получение файлов данных с помощью Requests

Работа с интерпретатором Python — это хороший способ проверить возможности Requests. Запустите IPython из командной строки и импортируйте `requests`:

```
$ ipython
Python 3.8.9 (default, Apr 32021, 01:02:10)
...

In [1]: import requests
```

[1] На самом деле это осознанная политика разработчиков.

[2] Некоторые зависимости, специфичные для платформы, все-таки могут приводить к ошибкам. Тред на https://stackoverflow.com/questions/29099404/ssl-insecureplatform-error-when-using-requests-package может подсказать, что делать, если проблемы не исчезли.

Для демонстрации работы библиотеки загрузим страницу из Википедии. Используем метод `get` из библиотеки Requests, чтобы получить страницу и присваиваем результат объекту `response`:

```
response = requests.get(\
"https://en.wikipedia.org/wiki/Nobel_Prize")
```

С помощью метода `dir` получим список атрибутов объекта `response`:

```
dir(response)
Out:
...
   ['content',
   'cookies',
   'elapsed',
   'encoding',
   'headers',
   ...
   'iter_content',
   'iter_lines',
   'json',
   'links',
   ...
   'status_code',
   'text',
   'url']
```

Большинство этих атрибутов интуитивно понятны, и вместе они дают полную информацию об HTTP-ответе. Обычно используется лишь малая их часть. Для начала проверим статус запроса:

```
response.status_code
Out: 200
```

Как известно любому веб-разработчику, 200 (OK) — это код состояния HTTP, указывающий, что транзакция прошла успешно. Другие распространенные коды:

401 (Unauthorized, «не авторизован»)
Попытка несанкционированного доступа

400 (Bad Request, «некорректный запрос»)
Запрос к веб-серверу содержит ошибки

403 (Forbidden, «доступ запрещен»)
Похоже на 401, но вход в систему недоступен

404 (Not Found, «не найдено»)
Попытка доступа к несуществующей странице

500 (Internal Server Error, «внутренняя ошибка сервера»)
Универсальный код для ряда ошибок

Если допустить опечатку в URL (например запросить `SNoble_Prize`), получим ошибку 404 (Not Found):

```
not_found_response = requests.get(\
"http://en.wikipedia.org/wiki/SNobel_Prize")
not_found_response.status_code
Out: 404
```

Посмотрим некоторую информацию из ответа на корректный запрос (200 OK) с помощью свойства `headers`:

```
response.headers
Out: {
   'date': 'Sat, 23 Oct 2021 23:58:49 GMT',
   'server': 'mw1435.eqiad.wmnet', 'content-encoding': 'gzip', ...
   'last-modified': 'Sat, 23 Oct 2021 17:14:09 GMT', ...
   'content-type': 'text/html; charset=UTF-8'...
   'content-length': '88959'
}
```

Здесь мы видим, в числе прочего, что страница сжата с помощью gzip, ее размер 87 КБ, тип контента `text/html` и кодировка UTF-8.

Поскольку мы узнали, что был возвращен текст, используем свойство `text` объекта response и посмотрим, что он из себя представляет:

```
response.text
#Out: u'<! DOCTYPE html>\n<html lang="en"
```

```
#dir="ltr" class="client-nojs">\n<head>\n<meta charset="UTF-8" #/>\
n<title>Nobel Prize — Wikipedia, the free #encyclopedia</title>\
n<script>document.documentElement... =
```

Мы видим, что действительно получили HTML-страницу из Википедии со встроенным кодом JavaScript. Как будет показано в «Скрейпинге данных» на стр. 190, чтобы разобраться в этом контенте, нам понадобится парсер для чтения HTML и разделения контента на блоки.

Итак, мы получили сырую веб-страницу, а теперь перейдем к использованию Requests для работы с web API.

Использование Python для получения данных через web API

Если файл с данными недоступен в интернете напрямую, возможно, существует API, позволяющий эти данные получить. Для их получения потребуется отправить запрос на сервер, который вернет нужные данные в фиксированном или в указанном в запросе формате.

Самые популярные форматы данных для web API — JSON и XML, хотя существуют и менее распространенные. Очевидно, что при создании визуализаций с использованием JavaScript предпочтительнее формат JSON (JavaScript Object Notation). К счастью, этот формат доминирует.

Существуют различные подходы к проектированию web API. В течение нескольких лет велись «архитектурные войны» между тремя основными типами:

REST
: (REpresentational State Transfer, «передача репрезентативного состояния») — использует комбинации HTTP-методов (GET, POST и т. д.) и унифицированных идентификаторов ресурсов (URI; например */user/kyran*) для доступа, создания и изменения данных.

XML-RPC
: (Remote Procedure Call, протокол удаленного вызова процедур) — кодирует в XML и передает через HTTP.

SOAP
: (Simple Object Access Protocol) также использует XML и HTTP.

Победителем, похоже, станут интерфейсы RESTful API, что радует. Помимо их изящества и простоты использовании и внедрения (см. главу 13), стандартизация REST повышает вероятность быстрого освоения новых API. В идеале, вы сможете повторно использовать существующий код. Новый игрок GraphQL позиционируется, как улучшенная альтернатива REST, но для визуализаций вы, скорее всего, будете использовать традиционные RESTful API.

Основные операции с удаленными данными описываются аббревиатурой CRUD: create (создать), retrieve (получить), update (обновить), delete (удалить). Первоначально ее ввели для описания функций реляционных баз данных. HTTP предоставляет аналогичные методы: POST (создание), GET (получение), PUT (обновление) и DELETE (удаление). Архитектура REST строится на этих методах, применяя их к универсальному идентификатору ресурса (Uniform Resource Identifier, URI).

Споры о том, что считать «истинным интерфейсом RESTful», могут быть сложными, но суть в следующем: URI (например, *https://example.com/api/items/2*) должен содержать всю информацию, необходимую для выполнения CRUD-операций, а конкретная операция (например: GET или DELETE) определяется HTTP-методом. Это исключает архитектуры вроде SOAP, где состояние сохраняется в метаданных заголовка запросов. Представим URI как виртуальный адрес данных, а CRUD — как все операции над ними.

Как специалисты по визуализации мы в основном являемся получателями нужных нам наборов данных, поэтому чаще всего используем HTTP-метод GET. Примеры ниже сфокусированы на извлечении данных через популярные web API. Надеюсь, что вы заметите некоторые закономерности.

Хотя требования к RESTful API (несохранение состояния URI и использование CRUD) задают четкие рамки, пространство для маневров еще остается.

Использование RESTful web API с помощью Requests

В Requests есть широкий набор функций для работы с основными HTTP-методами. Подробнее см. Requests quickstart. Для получения данных используются, как правило, GET и POST, причем GET — гораздо чаще. Метод POST позволяет эмулировать веб-формы, включая передачу данных для авторизации, значений полей и др. В тех случаях, когда приходится автоматизировать взаимодействие с веб-формой (например, со множеством селекторов), Requests упрощает это при помощи POST. GET охватывает все остальное, включая вездесущие RESTful API, которые предоставляют все больше структурированных данных в интернете.

Рассмотрим более сложный сценарий — обработку URL с параметрами. Организация экономического сотрудничества и развития (Organisation for Economic

SKLADCHIK.ORG

Cooperation and Development, OECD) предоставляет полезные наборы данных на своем сайте. Эти данные в основном содержат экономические показатели и статистику по странам-участницам OECD и могут лечь в основу интересных визуализаций. Сама OECD предоставляет несколько собственных визуализаций, например https://oreil.ly/aFmUv сравнение какой-либо страны с другими странами из OECD.

Описание web API OECD см. в документации. Запросы создаются с использованием имени набора данных (dsname) и групп размерностей (dimensions), причем размерности объединяются через + внутри группы и через точку между группами. URL-адрес также может принимать стандартные HTTP-параметры, начинающиеся после знака вопроса ? и разделенные амперсандами &:

```
<root_url>/<dsname>/<dim 1>.<dim 2>...<dim n>
/all?param1=foo&param2=baa..
<dim 1> = 'AUS'+'AUT'+'BEL'...
```

Ниже указан валидный URL:

```
http://stats.oecd.org/sdmx-json/data/QNA ❶
    /AUS+AUT.GDP+B1_GE.CUR+VOBARSA.Q ❷
    /all?startTime=2009-Q2&endTime=2011-Q4 ❸
```

❶ Задает набор данных QNA (Quarterly National Accounts).
❷ Четыре размерности: страны, показатели, тип данных и период времени.
❸ Данные за период со второго квартала 2009 г. по четвертый квартал 2011 г.

Создадим функцию на Python для запроса к OECD API (пример 5-1).

Пример 5-1. *Формирование URL-адреса для OECD API*

```
OECD_ROOT_URL = 'http://stats.oecd.org/sdmx-json/data'
def make_OECD_request(dsname, dimensions, params=None, \
root_dir=OECD_ROOT_URL):
    """ Формирует URL для OECD API и возвращает ответ """
    if not params: ❶
        params = {}

dim_args = ['+'.join(d) for d in dimensions] ❷
dim_str = '.'.join(dim_args)
```

```
    url = root_dir + '/' + dsname + '/' + dim_str + '/all'
    print('Requesting URL: ' + url)
    return requests.get(url, params=params) ❸
```

❶ Не используйте изменяемые значения, такие как {}, в качестве аргумента функции Python по умолчанию. См. руководство по Python, где дается объяснение этому требованию https://docs.python-guide.org/writing/gotchas/.

❷ Сначала при помощи генератора списков Python и метода `join` создаем список из dimensions, с объединением элементов d через разделитель «+» (например [*USA+AUS*, ...]). Затем используем `join` повторно, чтобы объединить элементы списка в строку `dim_str` (разделителем служит точка).

❸ Обратите внимание: `requests.get` может принимать в качестве второго аргумента параметр словарь, используя его при формировании строки запроса URL.

Давайте напишем функцию, чтобы получить экономические данные для США и Австралии с 2009 по 2010 гг.

```
response = make_OECD_request('QNA',
   (('USA', 'AUS'),('GDP', 'B1_GE'),('CUR', 'VOBARSA'), ('Q')),
   {'startTime':'2009-Q1', 'endTime':'2010-Q1'})
Requesting URL: http://stats.oecd.org/sdmx-json/data/QNA/
   USA+AUS.GDP+B1_GE.CUR+VOBARSA.Q/all
```

Теперь проверим статус ответа, и если он окажется «OK», посмотрим ключи словаря:

```
if response.status_code == 200:
   json = response.json()
   json.keys()
Out: [u'header', u'dataSets', u'structure']
```

В результате мы получили данные JSON в формате SDMX, международном стандарте обмена статистическими данными. Нельзя сказать, что это интуитивно понятный формат, но наборы данных часто имеют далеко не идеальную структуру. Хорошо, что Python отлично подходит для того, чтобы привести данные в порядок. Для работы с ним рекомендуется Python-библиотека pandas (см. главу 8) вместе с pandaSDMX, поддерживающей XML-версию формата.

OECD API по сути является RESTful, поскольку весь запрос содержится в URL-адресе, а HTTP-метод GET задает операцию получения данных. Если для использования API (например, Tweepy для Twitter) нет готовой Python-библиотеки, можно использовать шаблон из примера 5-1. Requests — очень удобная, хорошо сконструированная библиотека для взаимодействия с любыми web API.

Получение данных о странах для визуализации нобелевских лауреатов

Для визуализации, которую мы создаем с помощью нашего тулчейна, пригодятся некоторые данные национальной статистики. Численность населения, трехбуквенные коды государств (например: GDR, USA) и их географические центры могут потребоваться при визуализации данных о международной премии и ее распределении. REST Countries — удобный RESTful ресурс, предоставляющий различные статистические данные о странах. Получим с его помощью некоторые данные.

Запрос к REST Countries имеет следующую форму:

```
https://restcountries.com/v3.1/<field>/<name>?<params>
```

Как и в случае с OECD API (см. пример 5-1), создадим простую функцию для работы с API:

```
REST_EU_ROOT_URL = "https://restcountries.com/v3.1"

def REST_country_request(field='all', name=None, params=None):

    headers={'User-Agent': 'Mozilla/5.0'} ❶

    if not params:
        params = {}

    if field == 'all':
        response = requests.get(REST_EU_ROOT_URL + '/all')
        return response.json()

    url = '%s/%s/%s'%(REST_EU_ROOT_URL, field, name)
    print('Requesting URL: ' + url)
```

```
    response = requests.get(url, params=params, headers=headers)

    if not response.status_code == 200: ❷
        raise Exception('Request failed with status code ' \
        + str(response.status_code))

    return response.json() # JSON encoded data
```

❶ Указывать валидный `User-Agent` в заголовке запроса — это важно: некоторые сайты блокируют запросы без этого параметра.

❷ Перед возвращением ответа убедитесь, что статус ответа HTTP — 200 (OK), в противном случае выбрасываем исключение с описанием ошибки.

Используем функцию `REST_country_request`, чтобы получить список всех стран, где официальная валюта — доллар США:

```
response = REST_country_request('currency', 'usd')
response
Out:
[{u'alpha2Code': u'AS',
  u'alpha3Code': u'ASM',
  u'altSpellings': [u'AS',
  ...
  u'capital': u'Pago Pago',
  u'currencies': [u'USD'],
  u'demonym': u'American Samoan',
  ...
  u'latlng': [12.15, -68.266667],
  u'name': u'Bonaire',
  ...
  u'name': u'British Indian Ocean Territory',
  ...
  u'name': u'United States Minor Outlying Islands',
  ... ]}]
```

Полный набор данных на REST countries не так уж велик, поэтому для удобства сохраним копию как файл JSON. В последующих главах будем использовать его как для анализа, так и для демонстрации данных:

```
import json

country_data = REST_country_request() # all world data

with open('data/world_country_data.json', 'w') as json_file:
    json.dump(country_data, json_file)
```

Дважды мы выступили в роли потребителей API, а теперь рассмотрим готовые библиотеки, которые упрощают использование популярных web API.

Доступ к web API с помощью библиотек

Хотя библиотека Requests способна работать с большинством web API, по мере добавления в них аутентификации и усложнения структур данных, хорошая библиотека-обертка может значительно упростить процесс, избавив от рутинной работы. Сейчас мы рассмотрим пару самых популярных библиотек-оберток, чтобы получить представление о работе с ними и узнать о полезных отправных точках.

Использование Google Таблиц

Сегодня все чаще динамически обновляемые наборы данных хранятся *в облаке*. Так, например, вам может потребоваться визуализировать какие-либо данные из Google Таблиц, которые служат общим хранилищем для совместной работы. Лично я предпочитаю импортировать эти данные в pandas для анализа (см. главу 11), но хорошая библиотека позволяет работать с данными напрямую, обрабатывая веб-запросы по мере необходимости.

gspread — самая популярная Python-библиотека для доступа к Google Таблицам, значительно упрощающая эту задачу.

Чтобы использовать API, потребуются учетные данные для авторизации через OAuth 2.0[1]. Актуальные инструкции см. на сайте разработчиков Google. Следуя этим инструкциям, вы получите JSON-файл с вашим секретным ключом.

Установите *gspread* и последнюю версию клиентской библиотеки *google-auth*. Как это сделать с помощью `pip`:

```
$ pip install gspread
$ pip install --upgrade google-auth
```

[1] Доступ по OAuth1 больше не поддерживается.

В зависимости от вашей ОС может потребоваться pyOpenSSL:

```
$ pip install PyOpenSSL
```

Подробности и рекомендации по устранению проблем см. в документации.



Google API требует, чтобы электронные таблицы, к которым вы пытаетесь получить доступ, создавались через ваш API-аккаунт (или были ему доступны), а не через учетную запись. Адрес электронной почты для предоставления доступа к таблице имеется в консоли разработчиков Google и в JSON-файле с учетными данными, необходимыми для использования API. Он выглядит примерно так: *account-1@My Project…iam.gserviceaccount.com*.

После установки библиотек вы сможете получить доступ к любой из ваших электронных таблиц с помощью нескольких строк кода. Я использую таблицы Microbe-scope. В примере 5-2 показано, как загрузить таблицу.

Пример 5-2. *Открытие Google-таблицы*

```
import gspread

gc = gspread.service_account(\
        filename='data/google_credentials.json') ❶

ss = gc.open("Microbe-scope") ❷
```

❶ JSON-файл с учетными данными предоставляется службами Google и обычно выглядит как *My Project-b8ab5e38fd68.json*.

❷ Здесь мы открываем таблицу по имени. Альтернативные способы — `open_by_url` или `open_by_id`. Подробнее см. в документации gspread.

Теперь мы можем посмотреть листы таблицы:

```
ss.worksheets()
Out:
[<Worksheet 'bugs' id:0>,
 <Worksheet 'outrageous facts' id:430583748>,
 <Worksheet 'physicians per 1,000' id:1268911119>,
 <Worksheet 'amends' id:1001992659>]
ws = ss.worksheet('bugs')
```

Выбрав лист `bugs`, вы сможете читать и изменять значения столбцов, строк и ячеек (если лист не защищен от записи). Например, получим значения второго столбца с помощью команды `col_values`:

```
ws.col_values(1)
Out: [None,
 'grey = not plotted',
 'Anthrax (untreated)',
 'Bird Flu (H5N1)',
 'Bubonic Plague (untreated)',
 'C.Difficile',
 'Campylobacter',
 'Chicken Pox',
 'Cholera',...]
```



Если при доступе к Google Таблицам с помощью `gspread` возникает ошибка `BadStatusLine`, скорее всего, истекло время сессии. В этом случае повторно откройте таблицу. Подробнее об обсуждении этой проблемы см. https://github.com/burnash/gspread/issues/157#issuecomment-53970265.

Хотя API *gspread* позволяет стоить графики с помощью Matplotlib, я предпочитаю отправлять весь лист в pandas — мощный инструмент Python для анализа таблиц. Для этого используйте `get_all_records`, метод из библиотеки `gspread`, возвращающий список словарей. Его можно сразу преобразовать в pandas DataFrame (см. “DataFrame” на стр. 256):

```
df = pd.DataFrame(ws.get_all_records(expected_headers=[]))
df.info()
Out:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 41 entries, 0 to 40
Data columns (total 23 columns):
                                          41 non-null object
average basic reproductive rate           41 non-null object
case fatality rate                        41 non-null object
infectious dose                           41 non-null object
...
upper R0                                  41 non-null object
```

```
viral load in acute stage                    41 non-null object
yearly fatalities                            41 non-null object
dtypes: object(23)
memory usage: 7.5+ KB
```

В главе 11 мы рассмотрим, как как интерактивно анализировать данные в DataFrame.

Использование Twitter API с помощью Tweepy

С появлением социальных медиа стали генерироваться огромные объемы данных, и пробудился интерес к визуализации социальных сетей, содержащихся в них популярных хештегов и происходящих медиаштормов. Пожалуй, широковещательная сеть Twitter — самый богатый источник интересных данных для визуализаций, а ее API предоставляет твиты[1], отфильтрованные по пользователю, хештегу, дате и др.

Python Tweepy — простая в использовании библиотека для Twitter, предоставляющая полезные фичи, например, класс `StreamListener` для получения обновлений в режиме реального времени. Чтобы начать работу, понадобится токен доступа Twitter. Вы можете получить его, следуя инструкциям в документации Twitter по созданию своего Twitter-приложения. После этого нужно получить ключи и токены доступа по ссылке на странице вашего Twitter-приложения.

Tweepy обычно требует четыре элемента авторизации:

```
# The user credential variables to access Twitter API
access_token = "2677230157-Ze3bWuBAw4kwoj4via2dEntU86...TD7z"
access_token_secret = "DxwKAvVzMFLq7WnQGnty49jgJ39Acu...paR8ZH"
consumer_key = "pIorGFGQHShuYQtIxzYWk1jMD"
consumer_secret = "yLc4Hw82G0Zn4vTi4q8pSBcNyHkn35BfIe...oVa4P7R"
```

После этого доступ к твитам становится максимально простым. Ниже с помощью полученных токенов и ключей мы создаем объект OAuth `auth` и используем его для запуска сеанса API. Затем мы можем получить самые последние твиты из нашей ленты:

```
In [0]: import tweepy
        auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
        auth.set_access_token(access_token, access_token_secret)
```

[1] В настоящее время ограничение для бесплатного API — примерно 350 запросов в час.

```
api = tweepy.API(auth)
public_tweets = api.home_timeline()
for tweet in public_tweets:
    print(tweet.text)
```

```
RT @Glinner: Read these tweets https://t.co/QqzJPsDxUD
Volodymyr Bilyachat https://t.co/VIyOHlje6b +1 bmeyer
#javascript
RT @bbcworldservice: If scientists edit genes to
make people healthier does it change what it means to be
human? https://t.co/Vciuyu6BCx h...
RT @ForrestTheWoods:
Launching something pretty cool tomorrow. I'm excited. Keep
...
```

Класс `API` в Tweepy предлагает ряд удобных методов (подробнее см. документацию Tweepy). Распространенный вид визуализаций — сетевой граф, отображающий отношения друзей и подписчиков среди разных категорий пользователей Twitter. Для построения такой сети можно использовать методы Tweepy `followers_ids` (получить подписчиков конкретного пользователя) и `friends_ids` (получить всех, на кого пользователь подписан):

```
my_follower_ids = api.get_follower_ids() ❶

followers_tree = {'followers': []}
for id in my_follower_ids:
    # get the followers of your followers
    try:
        follower_ids = api.get_follower_ids(user_id=id) ❷
    except tweepy.errors.Unauthorized:
        print("Unauthorized to access user %d's followers"\
            %(id))
    followers_tree['followers'].append(\
        {'id': id, 'follower_ids': follower_ids})
```

❶ Получает ID ваших подписчиков (например, `[1191701545, 1554134420, ...]`).

❷ Первый аргумент метода `follower_ids` может быть ником или ID пользователя.

Примечание: если у пользователя более 100 подписчиков, то при создании сети высока вероятность появления ошибок, связанных с ограничением на количество запросов. В таком случае нужно снизить базовое число запросов до 180 за каждые 15 минут или оформить премиум-аккаунт на Twitter.

Сопоставляя взаимосвязи у подписчиков, можно создать сеть, которая позволит выявить интересные особенности групп и подгрупп, собравшихся вокруг определенного человека или темы. Хороший пример подобного анализа Twitter можно найти в блоге Гейба Сохни (Gabe Sawhney).

Класс `StreamListener` — один из самых интересных инструментов Tweepy. Он упрощает сбор и обработку отфильтрованных твитов в режиме реального времени. Моментальное обновление стримов Twitter использовалось во многих запоминающихся визуализациях (см. эти примеры из FlowingData и DensityDesign и вдохновляйтесь). Теперь настроим небольшой стрим для записи твитов, в которых упоминаются Python, JavaScript и визуализация (dataviz). Здесь мы просто выведем результаты на экран (в методе `on_data`), но обычно их сохраняют в файл или в базу данных (или и то, и другое с помощью SQLite):

```
import json

class MyStream(tweepy.Stream):
    """ Кастомизированный tweet-стрим """

    def on_data(self, tweet):
        """Что-то делает с данными твита..."""
        print(tweet)

    def on_error(self, status):
        return True # держит стрим открытым

stream = MyStream(consumer_key, consumer_secret,\
                  access_token, access_token_secret)
# Запустить этот стрим, отслеживая список ключевых слов track
stream.filter(track=['python', 'javascript', 'dataviz'])
```

Получив представление о типах API, с которыми можно столкнуться во время поиска интересных данных, рассмотрим основную технику, которую вы будете использовать (если вам не предоставят нужные данные в удобной форме) — сбор данных с помощью Python.

SKLADCHIK.ORG

Скрейпинг данных

Скрейпинг (scraping, «соскребать», «скоблить») — основная метафора процесса получения данных, не предназначенных для программной обработки вне сети. Метафора удачная, поскольку скрейпинг часто сводится к поиску оптимального баланса между изъятием слишком большого и слишком малого объема информации. Создание алгоритмов, по которым извлекаются только нужные данные с веб-страниц настолько чисто, насколько возможно, — занятие довольно хлопотное. Однако результат стоит усилий, вы получаете доступ к данным для визуализации, которые часто невозможно приобрести иным способом. При правильном подходе от скрейпинга можно даже получить истинное удовлетворение.

Зачем нужен скрейпинг

Если бы виртуальный мир был идеальным, онлайн-данные были бы организованы, как в библиотеке — с каталогизацией согласно системе библиотечной классификации, адаптированной для веб-страниц. К сожалению для страстных охотников за данными, интернет развивался, как правило не учитывая потребности визуализаторов данных в удобном доступе к информации. В реальности интернет напоминает гигантскую свалку данных: часть из них чиста и пригодна для использования (к счастью, этот процент растет), но бóльшая часть предназначена для восприятия человеком и плохо структурирована. Относительно «тупые» компьютеры не справляются с такими хаотичными и неформатированными данными, которые способны анализировать люди[1].

Скрейпинг — это создание шаблонов для извлечения нужных данных и отсеивания ненужных. Если повезет, веб-страницы с данными будут содержать полезные ориентиры: именованные таблицы, конкретные идентификаторы вместо общих классов и др. Если не повезет, придется использовать другие шаблоны или, что еще хуже, порядковые указатели вроде «*третья таблица в главном div*». Очевидно, что они ненадежны: добавление новой таблицы выше третьей разрушит всю логику.

В этом подразделе мы снова получим данные о нобелевских лауреатах, но уже с помощью скрейпинга. Для этого простого скрейпинга мы используем лучшую в своем классе Python-библиотеку Beautiful Soup, приберегая тяжелую артиллерию в виде Scrapy для следующей главы.

[1] Многие современные исследования в области машинного обучения и искусственного интеллекта (ИИ) посвящены созданию программного обеспечения, способного справляться с хаотичными, шумными, нечеткими, неформализованными данными. Однако на момент публикации этой книги я не слышал ни об одном готовом решении.



Важно: наличие данных и изображений в сети не означает возможности их свободного использования. В примерах мы будем работать с Википедией, которая разрешает полное повторное использование данных по лицензии Creative Commons. Всегда проверяйте правовой статус данных, и если есть сомнения, обращайтесь к администраторам сайта. Вас могут попросить хотя бы указать автора оригинала.

Beautiful Soup и lxml

Основные легковесные Python-библиотеки для скрейпинга — Beautiful Soup и lxml. Основной синтаксис селекторов в них отличается, но, как ни странно, одна может использовать парсеры другой. Принято считать, что парсер lxml значительно быстрее, но Beautiful Soup устойчивее при обработке плохо отформатированного HTML. Лично я считаю, что lxml достаточно надежен, а его синтаксис на основе на XPath-выражений мощнее и интуитивно понятнее. Полагаю, что для веб-разработчиков, знакомых с CSS и jQuery, выбор через CSS-селекторы будет естественнее. В большинстве систем lxml обычно является парсером по умолчанию для Beautiful Soup. Мы будем применять его в следующих подразделах.

Beautiful Soup входит в пакет Anaconda (см. главу 1) и легко устанавливается с помощью `pip`:

```
$ pip install beautifulsoup4
$ pip install lxml
```

Скрейпинг: первая попытка

Вооружившись Requests и Beautiful Soup, поставим себе задачу получить имена, годы, номинации и национальности всех лауреатов Нобелевской премии. Начнем со страницы Википедии List of Nobel laureates, «Нобелевские премии по году вручения». Прокрутив страницу вниз, мы увидим таблицы всех лауреатов, сгруппированные по годам и номинациям, — хороший старт, соответствующий нашим минимальным требованиям к данным.

Для веб-скрейпинга необходим инструмент анализа HTML. Лучший из известных мне — вкладка Elements в инструментах веб-разработчика Chrome (см. «Вкладка Elements» на стр. 140). На рисунке 5.1 показаны ключевые элементы для анализа структуры веб-страницы. Нужно научиться извлекать данные (в данном случае — таблицу Википедии), игнорируя остальные элементы на странице.

Создание точных шаблонов селекторов — основа эффективного скрейпинга. При выделении элемента DOM в инспекторе отображается шаблон CSS, а при щелчке правой кнопкой мыши — XPath. XPath — самый мощный язык запросов для выбора элементов DOM и основа Scrapy, решения скрейпинга промышленного класса.

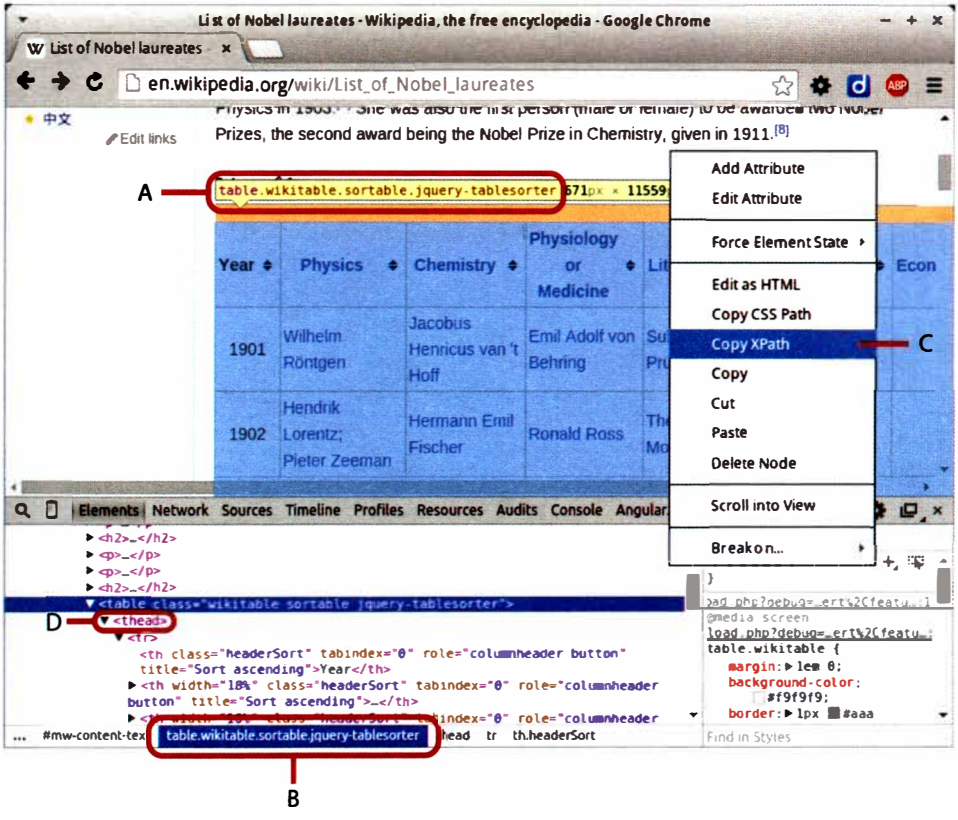


Рис. 5.1. *Страница Википедии List of Nobel laureates. A и B показывают CSS-селектор таблицы. Щелкнем правой кнопкой мыши и выберем C (Copy XPath) что даст нам XPath таблицы (`//*[@id="mw-content-text"]/table[1]`). D указывает на тег thead, сгенерированный jQuery.*

Получение объекта BeautifulSoup

Перед тем, как приступить к скрейпингу целевой веб-страницы, нужно проанализировать ее с помощью Beautiful Soup, преобразовав HTML в иерархическое дерево тегов (объект BeautifulSoup или soup — так называемый «суп»):

```
from bs4 import BeautifulSoup
import requests

BASE_URL = 'http://en.wikipedia.org'
# Википедия отклонит запрос, если не добавить
# атрибут 'User-Agent' в http-заголовок.
HEADERS = {'User-Agent': 'Mozilla/5.0'}

def get_Nobel_soup():
    """ Возвращает обработанное дерево тегов страницы нобелевских
    лауреатов"""
    # Сделаем запрос с корректными заголовками
    response = requests.get(
        BASE_URL + '/wiki/List_of_Nobel_laureates',
        headers=HEADERS)
    # Возвращаем контент ответа, обработанный Beautiful Soup
    return BeautifulSoup(response.content, "lxml") ❶
```

❶ Второй аргумент задает парсер lxml, который мы хотим использовать.

Получив «суп», рассмотрим, как найти нужные теги.

Выбор тегов

Beautiful Soup предлагает несколько способов выбора тегов из обработанного «супа», с тонкими различиями, в которых можно запутаться. Перед демонстрацией методов выбора получим «суп» для страницы Нобелевских лауреатов:

```
soup = get_Nobel_soup()
```

В нашей целевой таблице (см. рисунок 5.1) два определяющих класса: `wikitable` и `sortable` (на странице есть несколько несортируемых таблиц). Чтобы найти первый тег <table> с этими классами, используем метод `find` из Beautiful Soup. Метод `find` первым аргументом принимает имя тега, а вторым — словарь с классами, ID и другими идентификаторами:

```
In[3]: soup.find('table', {'class':'wikitable sortable'})
Out[3]:
```

```
<table class="wikitable sortable">
<tr>
<th>Year</th>
...
```

Хотя мы успешно нашли таблицу по ее классам, этот метод не очень надежен. Посмотрим, что произойдет, если изменить порядок CSS-классов:

```
In[4]: soup.find('table', {'class':'sortable wikitable'})
# ничего не возвращается
```

Метод find учитывает порядок классов, используя строку классов для поиска тегов. Если поменяется порядок классов, что может произойти при редактировании HTML, find завершится неудачей. По этой причине такие селекторы Beautiful Soup, как find и find_all трудно рекомендовать. Для ускорения работы с кодом HTML я предпочитаю использовать более простые и интуитивно понятные CSS-селекторы lxml.

Используя метод soup.select (доступный, если вы задали lxml-парсер, когда создавали «суп»), можно указывать HTML-элементы через CSS-классы, ID и др. Этот CSS-селектор преобразуется в синтаксис xpath, который lxml поддерживает[1].

Чтобы получить wikitable, мы выбираем таблицу в «супе», просто используя точечную нотацию для обозначения классов:

```
In[5]: soup.select('table.sortable.wikitable')
Out[5]:
[<table class="wikitable sortable">
   <tr>
   <th>Year</th>
   ...
]
```

Обратите внимание, что select возвращает массив результатов, находя все совпадающие теги в soup. Чтобы выбрать только один HTML-элемент, lxml предлагает удобный метод select_one. Посмотрим, какие заголовки в полученной таблице:

```
In[8]: table = soup.select_one('table.sortable.wikitable')
In[9]: table.select('th')
```

[1] Этот синтаксис отбора CSS будет знаком тем, кто работал с JavaScript-библиотекой jQuery; он также похож на тот, что использует D3.

```
Out[9]:
[<th>Year</th>,
   <th width="18%"><a href="/wiki/..._in_Physics..</a></th>,
   <th width="16%"><a href="/wiki/..._in_Chemis..</a></th>,
   ...
]
```

Искать по документу можно двумя способами: либо при помощи `select`, либо вызывая тег напрямую. Два примера ниже эквивалентны:

```
table.select('th')
table('th')
```

Beautiful Soup (с парсером lxml) предоставляет различные фильтры для поиска тегов: поиск по простому строковому имени (как в примере выше), по регулярному выражению, по списку имен тегов и многое другое. Подробнее см. в полном списке: https://www.crummy.com/software/BeautifulSoup/bs4/doc/#kinds-of-filters.

Наряду с `select` и `select_one` от lxml, в Beautful Soup есть 10 удобных методов для поиска по сформированному дереву. По сути, это варианты `find` и `find_all`, уточняющие область поиска. Например, `find_parent` и `find_parents` ищут родительские теги, двигаясь по дереву вверх. Все 10 методов см. в официальной документации Beautiful Soup.

Теперь, зная как выбирать таблицу Википедии и вооружившись методами поиска lxml, перейдем к созданию шаблонов для извлечения нужных данных.

Создание шаблонов для отбора

Выбрав нужную таблицу, разработаем шаблоны выбора, чтобы скрейпить нужные данные. Во вкладке Elements мы видим, что информация о лауреатах содержится в ячейках `<td>`, где ссылка `<a>` с атрибутом href ведет на страницу Википедии с биографией лауреата. Ниже показана типичная строка с классами CSS, которую можно использовать, как целевую для получения данных из ячеек `<td>`:

```
<tr>
   <td align="center">
      1901
   </td>
   <td>
      <span class="sortkey">
```

```
            Rontgen, Wilhelm
        </span>
        <span class="vcard">
            <span class="fn">
                <a href="/wiki/Wilhelm_R%C3%B6ntgen" \
                title="Wilhelm Rontgen">
                Wilhelm Rontgen
                </a>
            </span>
            </span>
    </td>
    <td>
    ...
</tr>
```

Если мы пройдемся в цикле по этим ячейкам, отслеживая их строку (year) и столбец (category), то сможем собрать список лауреатов со всеми данными, кроме национальности.

Ниже показана функция `get_column_titles`, которая извлекает заголовки столбцов таблицы (название номинаций), игнорируя первый столбец Year. Часто в заголовках таблицы Википедии содержится тег `'a'` со ссылкой. Все номинации Нобелевской премии построены на этой модели и ссылаются на соответствующие страницы Википедии. Если заголовок без ссылки, мы сохраняем его текст и href равно None:

```
def get_column_titles(table):
    """ Получает номинации Нобелевской премии
    из заголовка таблицы """
    cols = []
    for th in table.select_one('tr').select('th')[1:]: ❶
        link = th.select_one('a')
        # Сохраним имя номинации и любую имеющуюся ссылку
        # на Википедию
        if link:
            cols.append({'name':link.text,\
                         'href':link.attrs['href']})
        else:
            cols.append({'name':th.text, 'href':None})
    return cols
```

❶ Пройдем циклом по заголовку таблицы, игнорируя первый столбец Year ([1:]). Будут выбраны заголовки столбцов, как показано на рисунке 5.2.

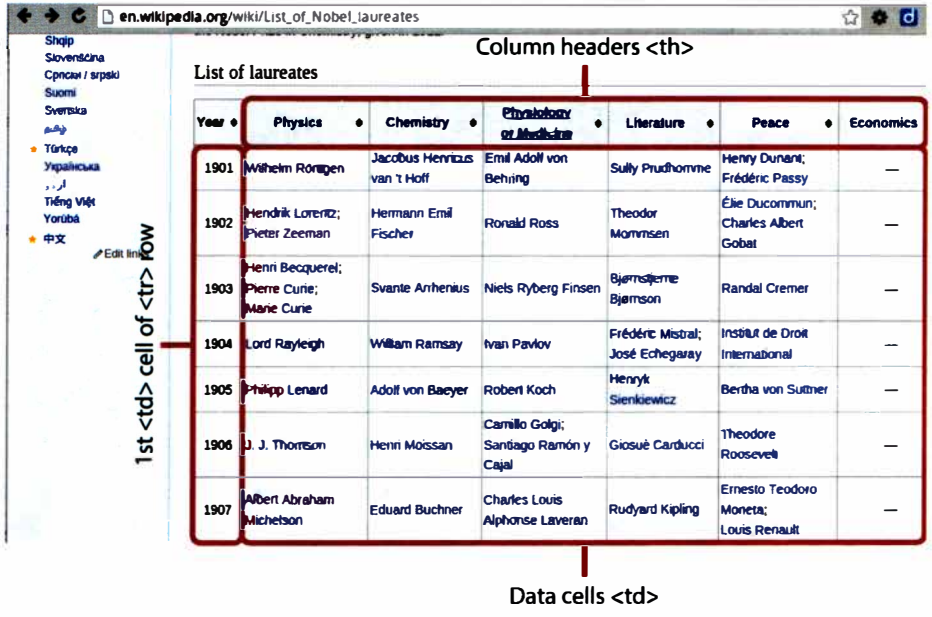


***Рис. 5.2.** Таблица лауреатов Нобелевской премии из Википедии*

Убедимся, что функция `get_column_titles` вернет нужный результат:

```
get_column_titles(table)
Out:
[{'name': 'Physics', \
 'href': '/wiki/List_of_Nobel_laureates_in_Physics'},
{'name': 'Chemistry',\
 'href': '/wiki/List_of_Nobel_laureates_in_Chemistry'},...
]
```

```
def get_Nobel_winners(table):
    cols = get_column_titles(table)
    winners = []
    for row in table.select('tr')[1:-1]: ❶
        year = int(row.select_one('td').text) # Gets 1st <td>
        for i, td in enumerate(row.select('td')[1:]): ❷
            for winner in td.select('a'):
                href = winner.attrs['href']
                if not href.startswith('#endnote'):
```

```
                winners.append({
                    'year':year,
                    'category':cols[i]['name'],
                    'name':winner.text,
                    'link':winner.attrs['href']
                })
    return winners
```

❶ Получаем все строки Year, начиная со второй, соответствующие строкам на рисунке 5.2.

❷ Находим данные из ячеек <td>, как показано на рисунке 5.2.

Проходя по строкам Year, мы получаем первый столбец Year, а затем проходим по оставшимся столбцам, с помощью `enumerate` отслеживая индекс, который будет соответствовать именам столбцов номинаций. Мы знаем, что все имена лауреатов содержатся в теге <a>, но иногда встречаются дополнительные теги <a>, начинающиеся с `#endnote`, которые мы фильтруем. И в завершении добавляем словарь, содержащий год, номинацию, имя и ссылку в конец списка данных. Обратите внимание, что у селектора winner есть словарь атрибутов `attrs`, содержащий, помимо прочего, href из тега <a>.

Убедимся, что `get_Nobel_winners` выводит список словарей лауреатов Нобелевской премии:

```
get_Nobel_winners(table)
[{'year': 1901,
  'category': 'Physics',
  'name': 'Wilhelm Rontgen',
  'link': '/wiki/Wilhelm_R%C3%B6ntgen'},
 {'year': 1901,
  'category': 'Chemistry',
  'name': "Jacobus Henricus van 't Hoff",
  'link': '/wiki/Jacobus_Henricus_van_%27t_Hoff'},
 {'year': 1901,
  'category': 'Physiologyor Medicine',
  'name': 'Emil Adolf von Behring',
  'link': '/wiki/Emil_Adolf_von_Behring'},
 {'year': 1901,
 ...}]
```

Итак, у нас есть полный список лауреатов Нобелевской премии со ссылками на их страницы в Википедии, и мы можем, используя эти ссылки, извлечь

данные из биографий. Для этого потребуется выполнить множество запросов, чего хотелось бы избежать. Разумным и вежливым[1] решением будет кешировать собираемые данные, что позволит проводить различные эксперименты со скрейпингом, не возвращаясь к Википедии.

Кеширование веб-страниц

Хотя достаточно просто написать быстрый кеш на Python, часто проще найти готовое и более качественное решение с открытым исходным кодом, написанное кем-то другим и любезно предоставленное сообществу. В Requests есть замечательный плагин `requests-cache` — всего несколько строк конфигурации позволит решить практически все проблемы с кешированием ответов.

Установим плагин с помощью `pip`:

```
$ pip install –upgrade requests-cache
```

`Requests-cache` использует манкипатчинг (monkey patching), позволяющий динамически заменять части `requests` API во время выполнения программы. То есть он может работать прозрачно. Достаточно просто установить этот кеш, а затем использовать `Requests`, как обычно. Все вопросы с кешированием будут обрабатываться автоматически. Самый простой способ использовать `requests-cache`:

```
import requests
import requests_cache

requests_cache.install_cache()
# используем requests, как обычно...
```

У метода `install_cache` имеется ряд полезных опций, например можно указать `бэкенд` кеша (`sqlite`, `memory`, `mongdb` или `redis`) или установить максимальное время кеширования (`expiry_after`) в секундах. Ниже мы создаем кеш с именем `nobel_pages` с бэкендом `sqlite`, который будет хранить наши станицы в течение двух часов (7200 секунд):

```
requests_cache.install_cache('nobel_pages',\
                             backend='sqlite', expire_after=7200)
```

[1] При скрейпинге вы влияете на скорость доступа к ресурсу других пользователей, что в конечном итоге стоит им денег, поэтому ограничение числа запросов — хороший тон.

SKLADCHIK.ORG

Чрезвычайно простой в использовании `requests-cache` удовлетворит большинство ваших потребностей в кешировании. Подробнее см. официальную документацию, там же есть пример троттлинга запросов, полезного при скрейпинге большого массива данных.

Скрейпинг национальности лауреатов

Разобравшись с кешированием, попробуем для эксперимента получить данные о национальности первых 50 лауреатов. Функция `get_winner_nationality()` использует ранее сохраненные ссылки на персональные страницы лауреатов, а затем получает значение поля `Nationality` из инфобокса (см. рисунок 5.3).

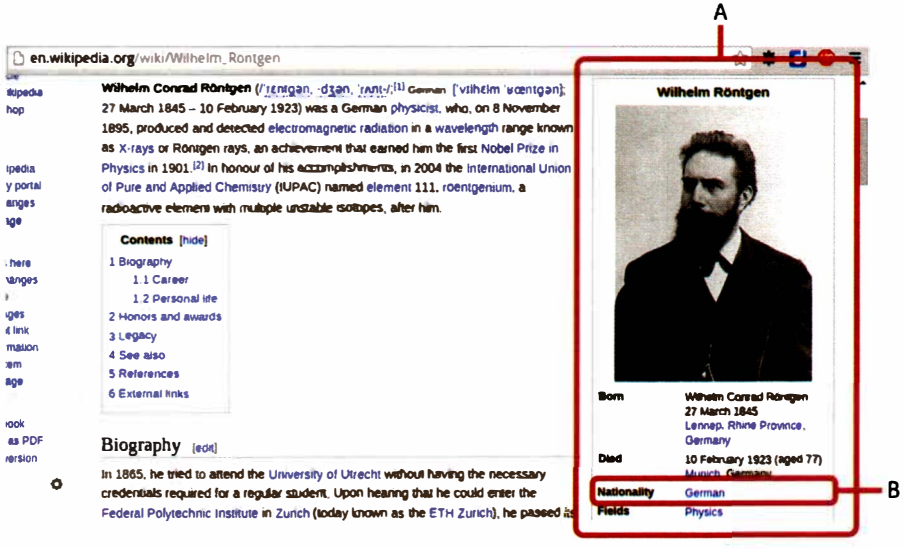


***Рис. 5.3.** Скрейпинг национальности лауреата*



При скрейпинге нужно искать устойчивые шаблоны и повторяющиеся элементы с полезными данными. Мы увидим далее, что инфобоксы Википедии для отдельных персон — не самый надежный источник, хотя беглый просмотр случайных страниц может создать обратное впечатление. В зависимости от объема данных полезно провести несколько пробных проверок корректности. Это можно сделать вручную, но, как упоминалось в начале главы, такой подход не масштабируется и не способствует развитию навыков.

Функция в примере 5.3 принимает словарь с данными лауреата, собранными ранее, и возвращает новый словарь с именем и ключом `Nationality`, если национальность найдена. Запустим ее для первых 50 лауреатов и посмотрим, насколько часто отсутствует значение `Nationality`.

***Пример 5-3.** Скрейпинг национальности лауреата со страницы с его биографией*

```
HEADERS = {'User-Agent': 'Mozilla/5.0'}

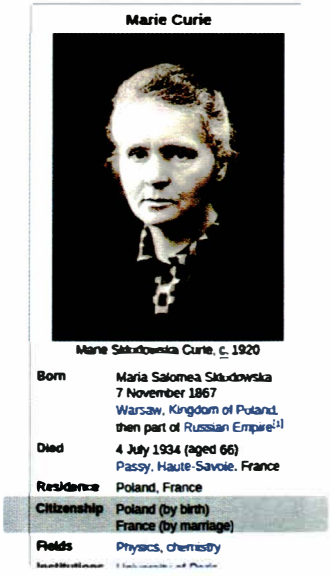
def get_winner_nationality(w):
    """ Скрейпинг биографических данных со страницы лауреата
    в Википедии"""
    response = requests.get('http://en.wikipedia.org' \
                            + w['link'], headers=HEADERS)
    content = response.content.decode('utf-8')
    soup = BeautifulSoup(content)
    person_data = {'name': w['name']}
    attr_rows = soup.select('table.infobox tr')
    for tr in attr_rows:
        try:
            attribute = tr.select_one('th').text ❶
            if attribute == 'Nationality':       ❷
                person_data[attribute] = tr.select_one('td').text
        except AttributeError:
            pass
    return person_data
```

❶ Используем CSS-селектор для поиска всех строк `<tr>` в таблице с классом `infobox`.

❷ Цикл по строкам для поиска поля Nationality.

В примере 5-4 национальность не найдена для 14 из первых 50 лауреатов. Возможно, у Института международного права (фр. Institut de Droit International, IDI) спорная национальная принадлежность, но уж Теодор Рузвельт — стопроцентный американец. Кликнув по некоторым именам, мы увидим проблему (см. рисунок 5.4): отсутствие стандартизированного формата биографий. Для Марии Кюри (Marie Curie) вместо национальности (*Nationality*) указано гражданство (*Citizenship*), для Нильса Финсена (Niels Finsen) нет данных о национальности;

а у Рэндалла Кримера (Randall Cremer) есть только фотография. Инфобоксы нельзя считать надежным источником информации о национальности лауреатов, но раз других структурных источников нет, вернемся к исходной точке. В следующей главе мы реализуем успешный подход с использованием Scrapy и другой стартовой страницы.



'Citizenship'



No nationality



Photo only

Рис. 5.4. *Лауреаты, у которых не указана национальность*

Пример 5-4. *Тестирование собранных данных о национальности*

```
wdata = []
# тестируем на первых 50 лауреатах
for w in winners[:50]:
   wdata.append(get_winner_nationality(w))
missing_nationality = []
for w in wdata:
   # если 'Nationality' отсутствует, добавить в список
   if not w.get('Nationality'):
      missing_nationality.append(w)
# полученный список
missing_nationality
[{'name': 'Theodor Mommsen'},
 {'name': 'Élie Ducommun'},
 {'name': 'Charles Albert Gobat'},
 {'name': 'Pierre Curie'},
```

```
{'name': 'Marie Curie'},
{'name': 'Niels Ryberg Finsen'},
...
{'name': 'Theodore Roosevelt'}, ...]
```

Поскольку Википедия предназначена для людей, а не машин, строгость форматов страдает. Похожие недостатки есть и у других сайтов: чем объемнее наборы данных, тем больше тестов потребуется для выявления ошибок в шаблонах сбора.

Хотя я привел довольно условный пример (для демонстрации инструмента), он, я надеюсь, передал некоторую сумбурность скрейпинга. Безуспешного поиска поля Nationality можно было избежать: провести поиск по веб-страницам и вручную проанализировть HTML. Однако, если набор данных значительно больше, а частота пропусков ниже, то программные методы — по мере освоения модулей скрейпинга — начнут приносить плоды.

Этот небольшой тест для демонстрации возможностей Beautiful Soup показал, что над сбором данных требуется хорошенько поразмыслить; так обычно и бывает при скрейпинге. В следующей главе пустим в ход тяжелую артиллерию — Scrapy — и, используя информацию из этого подраздела, соберем данные для визуализации о Нобелевских лауреатах.

Резюме

В этой главе мы рассмотрели примеры основных способов извлечения веб-данных в структуры данных Python, базы данных или pandas DataFrame. Requests — фундаментальных инструмент Python нашего тулчейна визуализации для работы с HTTP. В простых RESTful API для получения данных через Requests требуется буквально несколько строк кода на Python. При использовании менее удобных API, например, с потенциально сложной авторизацией, библиотека-обертка, такая как Tweepy (для Twitter) избавит от множества хлопот: отследит и, по необходимости, автоматически ограничит частоту отправки запросов. Это важное преимущество, особенно когда есть опасность попасть в список блокировки недобросовестных потребителей.

Мы также сделали первые шаги в скрейпинге, который часто становится необходимым решением, если API отсутствует, а данные предназначены для потребления человеком. В следующей главе мы получим все необходимые для визуализации данные о лауреатах Нобелевской премии, используя Scrapy — Python-библиотеку промышленного класса для скрейпинга.

Эффективный скрейпинг с помощью Scrapy

По мере того, как цели скрейпинга становятся сложнее, использование импровизированных решений с Beautiful Soup и запросов приводит к хаосу. Управление данными усложняется, когда запросы порождают еще больше запросов, а синхронное выполнение запросов резко замедляет процесс. Возникает целый ряд неожиданных проблем, для решения которых (помимо прочих) необходима мощная и надежная библиотека Scrapy.

Если Beautiful Soup — это удобный «перочинный ножик» для быстрого и грязного скрейпинга, то Python-библиотека Scrapy способна с легкостью выполнять скрейпинг большого объема данных. Она включает встроенное кеширование (с настройкой времени жизни кеша), асинхронные запросы (при помощи Python-фреймворка Twisted), рандомизацию юзер-агентов и многое другое. Правда, Scrapy не просто освоить, но данная глава призвана сгладить процесс знакомства с помощью простого примера. Я считаю, что Scrapy — мощное дополнение к любому тулчейну визуализации, открывающее широкие возможности для сбора веб-данных.

В «Скрейпинге данных» мы собрали набор данных, содержащий имена, годы и номинации для всех Нобелевских лауреатов. Эксперимент со скрейпингом биографических страниц выявил сложности при получении национальностей. Теперь мы повышаем планку: наша цель — извлечь объекты из формы, показанной в примере 6-1.

Пример 6-1. *Целевой объект JSON с данными о лауреате Нобелевской премии*

```
{
    "category": "Physiology or Medicine",
    "country": "Argentina",
    "date_of_birth': "8 October 1927",
    "date_of_death': "24 March 2002",
    "gender": "male",
```

```
    "link": "http:\/\/en.wikipedia.org\/wiki\/C%C3%A9sar_Milstein",
    "name": "C\u00e9sar Milstein",
    "place_of_birth": "Bah\u00eda Blanca, Argentina",
    "place_of_death": "Cambridge, England",
    "text": "C\u00e9sar Milstein, Physiology or Medicine, 1984",
    "year": 1984
}
```

К этим данным мы добавим фотографии лауреатов (где возможно) и краткие биографические данные (см. рисунок 6.1). Фотографии и текст добавят индивидуальности нашей визуализации.



Рис. 6.1. *Цели скрейпинга на страницах лауреатов*

Установка Scrapy

Scrapy входит в пакет Anaconda (см. главу 1), так что у вас она может быть уже установлена. Если нет, используйте команду `conda`:

```
$ conda install -c https://conda.anaconda.org/anaconda scrapy
```

Если вы не используете Anaconda, вам поможет `pip`[1]:

```
$ pip install scrapy
```

[1] См. документацию Scrapy об особенностях ее установки на различных платформах https://doc.scrapy.org/en/latest/intro/install.html.

После установки Scrapy станет доступна команда `scrapy`. В отличие от большинства Python-библиотек, Scrapy управляется из командной строки в контексте проекта скрейпинга, определяемого файлами конфигурации, пауками, конвейерами и др. С помощью опции `startproject` создадим новый проект для скрейпинга данных о Нобелевских лауреатах. Будет создан каталог проекта, поэтому убедитесь, что вы запускаете опцию из нужного рабочего каталога:

```
$ scrapy startproject nobel_winners
New Scrapy project 'nobel_winners' created in:
    /home/kyran/workspace/.../scrapy/nobel_winners

You can start your first spider with:
    cd nobel_winners
    scrapy genspider example example.com
```

Как сказано в выводе `startproject`, для дальнейший работы со Scrapy перейдите в каталог *nobel_winners*. Рассмотрим дерево каталогов созданного проекта:

```
nobel_winners
├── nobel_winners
│   ├── __init__.py
│   ├── items.py
│   ├── middlewares.py
│   ├── pipelines.py
│   ├── settings.py
│   └── spiders
│       └── __init__.py
└── scrapy.cfg
```

Каталог проекта включает подкаталог с тем же именем и файл конфигурации *scrapy.cfg*. Подкаталог *nobel_winners* — это пакет Python (содержащий файл *__init__.py)* и несколько файлов-заготовок, а также каталог *spiders*, в котором будут находиться пауки.

Постановка целей

В «Скрейпинге данных» мы пытались извлечь национальности лауреатов из страниц с биографиями, но во многих случаях данные или отсутствовали,

или были помечены не подходящим образом (см. главу 5). Спасением стала страница Википедии, где лауреаты сгруппированы по странам https://oreil.ly/p6pXm. То, что лауреаты представлены не в таблицах, а в виде упорядоченных списков (см. рисунок 6.2), немного усложняет восстановление основных данных по имени, категории и году. Кроме того, данные организованы не идеально (например, заголовки стран и списки лауреатов не разделены на отдельные блоки). Но мы увидим далее, что пара хорошо структурированных запросов Scrapy легко добудет нужные данные.

На рисунке 6.2 показана стартовая страница нашего первого паука, а также ключевые элементы, на которые он будет ориентироваться. За списком названий стран (A) следует упорядоченный список (B) лауреатов от соответствующих стран.

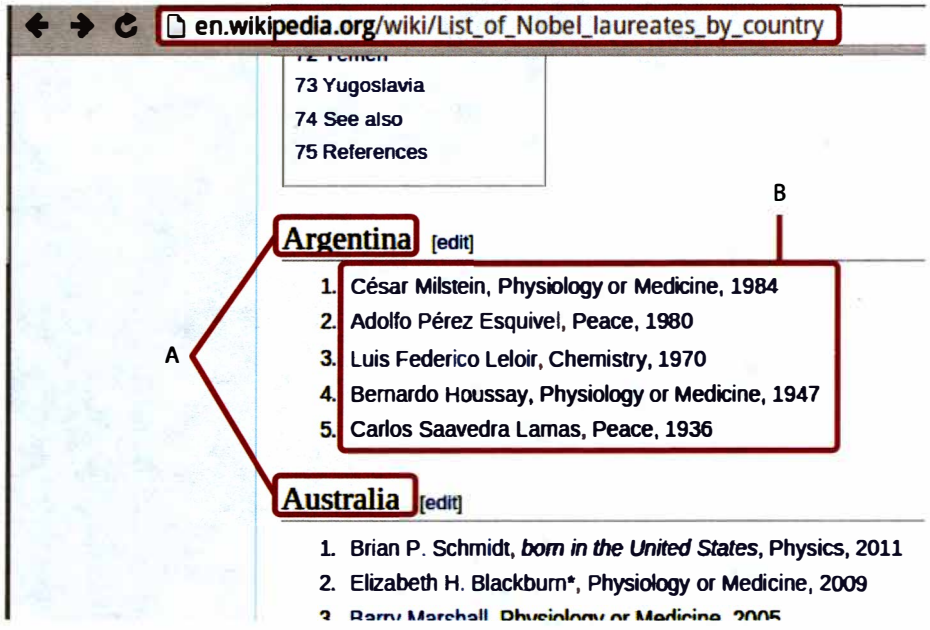


Рис. 6.2. *Скрейпинг данных Википедии по национальности лауреатов Нобелевской премии*

Чтобы извлечь данные из списка, нужно запустить DevTools браузера Chrome (см. «Вкладка Elements» на стр. 140) и проверить целевые элементы с помощью вкладки Elements и ее инспектора (увеличительного стекла). На рисунке 6.3 показаны ключевые HTML-цели для первого паука: заголовки уровня h3 с названиями стран и упорядоченный список (ol) лауреатов (li).

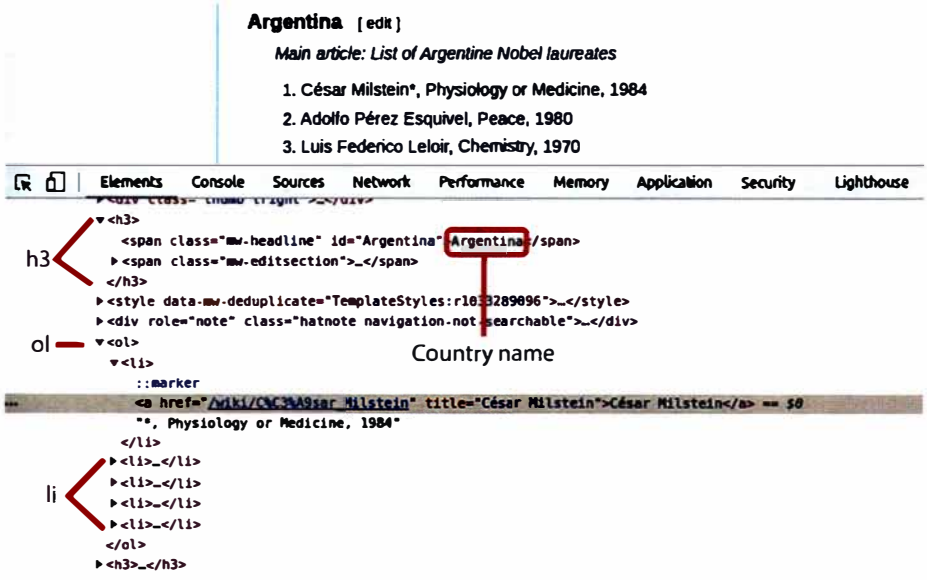


Рис. 6.3. *Целевые HTML-элементы для сбора данных*

Работа с XPath в Scrapy

Scrapy использует выражения xpath для определения целевых HTML-элементов. XPath — синтаксис для выборки частей X[HT]ML-документа, который часто позволяет быстро решить задачу. Хотя он может быть довольно сложным в деталях, его основы просты.

Чтобы получить путь XPath к элементу HTML, откройте вкладку Elements в Chrome, наведите курсор на элемент, затем кликните правой кнопкой и выберите Copy XPath. К примеру, для названий стран в списке лауреатов (h3 на рисунке 6.3), XPath Аргентины (первой в списке) выглядит так:

```
//*[@id="mw-content-text"]/div[1]/h3[1]
```

Расшифруем это с помощью следующих правил XPath:

```
//E
```

Выбирает все элементы <E> в документе (например `//img` вернет все изображения на странице)

```
//E[@id="foo"]
```

Выбирает элемент <E> с ID foo

```
//*[@id="foo"]
```

Выбирает все элементы с ID foo

```
//E/F[1]
```

Выбирает первый дочерний элемент <F>элемента <E>

```
//E/*[1]
```

Выбирает любой первый дочерний элемент элемента <E>

Согласно правилам, заголовок Argentina `//*[@id="mw-content-text"]/div[1]/h3[1]` — первый дочерний заголовок (h3) первого `div` элемента DOM с ID `mw-content-text`. Эквивалентная HTML-структура:

```
<div id="mw-content-text">
   <div>
      <h3>
         ...
      </h3>
   </div>
   ...
</div>
```

Примечание: в отличие от Python индексы xpath начинаются не 0, а с *1*.

Тестирование XPath в Scrapy Shell

Для эффективного скрейпинга критично правильно определить цель XPath, что может потребовать нескольких итераций. Scrapy упрощает этот процесс с помощью командной оболочки, которая принимает URL-адрес и формирует контекст ответа для тестирования:

```
$ scrapy shell
https://en.wikipedia.org/wiki/List_of_Nobel_laureates_by_country

2021-12-09 14:31:06 [scrapy.utils.log] INFO: Scrapy 2.5.1 started
(bot: nobel_winners)
...
```

SKLADCHIK.ORG

```
2021-12-09 14:31:07 [scrapy.core.engine] INFO: Spider opened
2021-12-09 14:31:07 [scrapy.core.engine] DEBUG: Crawled (200)
<GET https://en.wikip...List_of_Nobel_laureates_by_country>
(referer: None)
[s] Available Scrapy objects:

[s] crawler         <scrapy.crawler.Crawler object at 0x3a8f510>
[s] item {}
[s] request         <GET https://...Nobel_laureates_by_country>
[s] response        <200 https://...Nobel_laureates_by_country>
[s] settings        <scrapy.settings.Settings object at 0x34a98d0>
[s] spider          <DefaultSpider 'default' at 0x3f59190>

[s] Useful shortcuts:
[s] shelp()         Shell help (print this help)
[s] fetch(url[, redirect=True]) Fetch URL and update local objects
(by default, redirects are followed)
[s] fetch(req)      Fetch a scrapy.Request and update
[s] view(response)   View response in a browser
In [1]:
```

Теперь, в оболочке (на базе IPython) с автодополнением кода и подсветкой синтаксиса, мы можем тестировать наши xpath. Выберем все заголовки <h3> на странице Википедии:

```
In [1]: h3s = response.xpath('//h3')
```

В результате в `h3s` будет SelectorList, объект Python, представляющий собой специализированный список. Посмотрим, сколько заголовков получено:

```
In [2]: len(h3s)
Out[2]: 91
```

Выберем первый объект `Selector` и исследуем его методы и свойства в оболочке Scrapy, нажав Tab после добавления точки в конце:

```
In [3] h3 = h3s[0]
In [4] h3.
```

```
attrib   get        re                  remove              ...
css      getall     re_first            remove_namespaces   ...
extract  namespaces register_namespace  response            ...
```

Вы часто будете использовать метод `extract`, чтобы получить сырой результат XPath-селектора:

```
In [5]: h3.extract() Out[6]:
u'<h3>
   <span class="mw-headline" id="Argentina">Argentina</span>
   <span class="mw-editsection">
   <span class="mw-editsection-bracket">
   ...
   </h3>'
```

Здесь показано, что заголовки стран начинаются с первого тега `<h3>`, содержащего элемент `span` с классом `mw-headline`. Мы можем использовать наличие класса `mw-headline` в качестве фильтра для заголовков стран, а их содержимое в качестве названия страны. Проверим XPath-селектор с помощью метода `text`, который извлечет текст из span с классом `mw-headline`. Обратите внимание, что используется метод `xpath` селектора `<h3>`, что делает запрос XPath относительным к данному элементу:

```
In [7]: h3_arg = h3
In [8]: country = h3_arg.xpath(\
                 'span[@class="mw-headline"]/text()')\
.extract()
In [9]: country
Out[9]: ['Argentina']
```

Метод `extract` возвращает список возможных совпадений, в нашем случае, единственную строку — `'Argentina'`. Перебирая список `h3s`, мы получим наименования всех стран.

Предположим, у нас есть заголовок страны `<h3>`, теперь нужно получить упорядоченный список `<ol>` лауреатов Нобелевской премии, следующий за этим заголовком (рисунок 6.2 B). Для этого удобно использовать XPath-селектор `following-sibling`. Получим первый нумерованный список после заголовка Argentina:

```
In [10]: ol_arg = h3_arg.xpath('following-sibling::ol[1]')
Out[10]: ol_arg
[<Selector xpath='following-sibling::ol[1]' data=u'<ol><li>
<a href="/wiki/C%C3%A9sar_Milst'>]
```

Обрезанные данные `ol_arg` показывают, что выбран упорядоченный список. Обратите внимание, что хотя `Selector` всего один, `xpath` по-прежнему возвращает `SelectorList` (список селекторов). Обычно первый элемент выбирают напрямую, так удобнее:

```
In [11]: ol_arg = h3_arg.xpath('following-sibling::ol[1]')[0]
```

Теперь, получив нумерованный список, извлечем все его элементы `<li>` (по состоянию на середину 2022 года):

```
In [12]: lis_arg = ol_arg.xpath('li')
In [13]: len(lis_arg)
Out[13]: 5
```

Изучим один из элементов этого списка с помощью `extract`. В качестве первого теста мы попробуем извлечь имя лауреата и текст элемента списка:

```
In [14]: li = lis_arg[0] # select the first list element
In [15]: li.extract()
Out[15]:
'<li><a href="/wiki/C%C3%A9sar_Milstein"
        title="C\xe9sar Milstein">C\xe9sar Milstein</a>,
        Physiology or Medicine, 1984</li>'
```

У извлеченного элемента стандартный шаблон: имя лауреата (со ссылкой на его страницу в Википедии), далее через запятую — номинация премии и год. Надежный способ получить имя лауреата — выбрать текст из первого тега `<a>` элемента списка:

```
In [16]: name = li.xpath('a//text()')[0].extract()
In [17]: name
Out[17]: 'César Milstein'
```

Часто бывает полезно получить весь текст, например, из элемента списка, удалив различные HTML-теги <a>, и др. Чтобы получить список строк из потомков, можно использовать ось XPath `descendant-or-self`:

```
In [18]: list_text = li.xpath('descendant-or-self::text()')\
.extract()
In [19]: list_text
Out[19]: ['César Milstein', '*, Physiology or Medicine, 1984']
```

Объединим элементы в цельный текст:

```
In [20]: ' '.join(list_text)
Out[20]: 'César Milstein *, Physiology or Medicine, 1984'
```

Обратите внимание, что первый элемент `list_text` — имя лауреата. Это позволяет получить его даже при отсутствии гиперссылки.

Теперь, определив XPath-селекторы для целей скрейпинга (имя лауреата и текст ссылки), встроим их в первого паука Scrapy.

Поиск при помощи относительного Xpath

Как мы увидели, `xpath`-селекторы Scrapy возвращают списки CSS-селекторов, имеющих в свою очередь собственные `xpath`-методы. Используя метод `xpath`, важно различать абсолютные и относительные селекторы. Рассмотрим это различие на примере оглавления (Table of Contents) страницы лауреатов Нобелевской премии.

Структура оглавления:

```
<div id='toc'... >
   ...
      <ul ... >
         <li ... >
            <a href='Argentina'> ... </a>
         </li>
         ...
      </ul>
   ...
</div>
```

Чтобы выбрать элемент оглавления на странице Википедии, используем стандартный запрос xpath для div с id toc:

```
In [21]: toc = response.xpath('//div[@id="toc"]')[0]
```

Чтобы получить все теги <li> со странами, используем относительный xpath для выбранного элемента toc. HTML-код на рисунке 6.3 показывает, что неупорядоченный список стран ul является первым элементом второго пункта в основном списке оглавления. Этот список можно выбрать с помощью двух эквивалентных выражений XPath. Оба выбирают дочерние элементы относительно текущего toc:

```
In [22]: lis = toc.xpath('.//ul/li[2]/ul/li')
In [23]: lis = toc.xpath('ul/li[2]/ul/li')
In [24]: len(lis)
Out[24]: 81 # число стран в оглавлении (на июль 2022 г.)
```

Частая ошибка — использование абсолютного xpath селектора для текущей выборки, который выбирает элементы из всего документа, в этом случае — все (<ul>) теги:

```
In [25]: lis = toc.xpath('//ul/li')
In [26]: len(lis)
OUt[26]: 271
```

Путаница между относительными и абсолютными запросами часто обсуждается на форумах. Обращайте внимание на точку (.//) в начале относительных Xpath.



Подобрать правильное Xpath-выражение для целевых элементов порой непросто, а в особых случаях требуются многоуровневые конструкции. К счастью, существует множество хороших шпаргалок по Xpath. Отличную их подборку см. на devhints.io.

Первый паук Scrapy

Освоив базовые принципы XPath, создадим нашего первого скрейпинг-бота для сбора названий стран и информации о лауреатах (рисунок 6.2 A и B).

Скрейпинг-боты в Scrapy называются пауками (*spider*). Каждый паук — это модуль Python в каталоге *spiders* вашего проекта. Назовем первого паука *nwinner_list_spider.py*:

```
.
├── nobel_winners
│   ├── __init__.py
│   ├── items.py
│   ├── middlewares.py
│   ├── pipelines.py
│   ├── settings.py
│   └── spiders
│       ├── __init__.py
│       └── nwinners_list_spider.py <---
└── scrapy.cfg
```

Пауки — это подклассы класса `scrapy.Spider`. Любой из них при сохранении в каталоге *spiders* автоматически обнаруживается Scrapy и его имя используется в качестве аргумента команды `scrapy`.

Пример 6-2 демонстрирует базовый шаблон, который можно использовать для большинства пауков. Сначала вы создаете подкласс `scrapy.Item` для определения полей данных (секция A в примере 6-2). Затем создаете именованного паука, унаследовав `scrapy.Spider` (секция B в примере 6-2). Для запуска паука используйте его имя в командной строке `scrapy`. У пауков есть метод `parse`, обрабатывающий ответы на HTTP-запросы к изначальным URL-адресам из атрибута класса `start_url`. В нашем случае изначальный URL — это страница Википедии List of Nobel laureates by country.

***Пример 6-2.** Первый паук Scrapy*

```
# nwinners_list_spider.py

import scrapy
import re
# A. Определяет какие данные скрейпить
class NWinnerItem(scrapy.Item):
    country = scrapy.Field()
```

```
    name = scrapy.Field()
    link_text = scrapy.Field()

# В Создает паука с именем
class NWinnerSpider(scrapy.Spider):
    """ Скрейпит страну и текстовое поле ссылки лауреатов
    Нобелевской премии. """

    name = 'nwinners_list'
    allowed_domains = ['en.wikipedia.org']
    start_urls = [
        "http://en.wikipedia.org ... of_Nobel_laureates_by_country"
    ]
    # С Метод parse для анализа HTTP ответа
    def parse(self, response):

        h3s = response.xpath('//h3') ❶

        for h3 in h3s:
            country = h3.xpath('span[@class="mw-headline"]'\
            'text()').extract() ❷
            if country:
                winners = h3.xpath('following-sibling::ol[1]') ❸
                for w in winners.xpath('li'):
                    text = w.xpath('descendant-or-self::text()')\
                    .extract()
                    yield NWinnerItem(
                        country=country[0], name=text[0],
                        link_text = ' '.join(text)
                        )
```

❶ Получает все заголовки <h3> на странице, большинство из них — наша цель: названия стран.

❷ Где это возможно, извлекает текст элементов с классом mw-headline, дочерних для <h3>.

❸ Получает список лауреатов для текущей страны.

В примере 6-2 метод parse получает ответ на HTTP-запрос к странице лауреатов Нобелевской премии в Википедии и возвращает элементы Scrapy, которые

затем преобразуются в объекты JSON и добавляются в конец массива объектов JSON-файла.

Запустим нашего первого паука, чтобы убедиться в правильности парсинга и скрейпинга данных о лауреатах Нобелевской премии. Перейдем в корневой каталог скрейпинг-проекта (содержащий файл *scrapy.cfg*) — *nobel_winners*. Проверим список доступных пауков:

```
$ scrapy list
nwinners_list
```

Как и ожидалось, в каталоге *spiders* есть один паук — `nwinners_list`. Начнем скрейпинг по команде `crawl`, сохраняя результат в файл *nwinners.json* По умолчанию мы получим множество логов, сопутствующих обходу паука:

```
$ scrapy crawl nwinners_list -o nobel_winners.json
2021- ...  [scrapy] INFO: Scrapy started (bot: nobel_winners)
...
2021- ...  [nwinners_list] INFO: Closing spider (finished)
2021- ...  [nwinners_list] INFO: Dumping Scrapy stats:
          {'downloader/request_bytes': 1147,
           'downloader/request_count': 4,
           'downloader/request_method_count/GET': 4,
           'downloader/response_bytes': 66459,
           ...
           'item_scraped_count': 1169, ❶
2021- ...  [scrapy.core.engine] INFO: Spider closed (finished)
```

❶ Мы успешно собрали данные 1169 нобелевских лауреатов.

Лог выполнения команды scrapy `crawl` показывает, что успешно собрано 1169 записей. Посмотрим выходной JSON-файл, чтобы убедиться: все прошло по плану:

```
$ head nobel_winners.json
[{"country": "Argentina",
   "link_text": "C\u00e9sar Milstein, Physiology or Medicine,"\ "
   1984",
   "name": "C\u00e9sar Milstein"},
  {"country": "Argentina",
```

```
    "link_text": "Adolfo P\u00e9rez Esquivel, Peace, 1980",
    "name": "Adolfo P\u00e9rez Esquivel"},
    ...
```

Как видим, массив JSON-объектов содержит четыре корректных ключевых поля.

Мы убедились, что паук успешно извлекает данные из списка всех лауреатов Нобелевской премии на странице, теперь начнем его совершенствовать, чтобы получить все необходимые для визуализации данные (см. пример 6-1 и рисунок 6.1).

Сначала добавим поля данных, которые планируем скрейпить, в `scrapy.Item`:

```
...
class NWinnerItem(scrapy.Item):
    name = scrapy.Field()
    link = scrapy.Field()
    year = scrapy.Field()
    category = scrapy.Field()
    country = scrapy.Field()
    gender = scrapy.Field()
    born_in = scrapy.Field()
    date_of_birth = scrapy.Field()
    date_of_death = scrapy.Field()
    place_of_birth = scrapy.Field()
    place_of_death = scrapy.Field()
    text = scrapy.Field()
...
```

Чтобы упростить код, используем специальную функцию `process_winner_li` для обработки текстовых полей ссылок лауреатов. Она принимает селектор ссылок и название страны, а возвращает словарь, содержащий полученные данные:

```
...

def parse(self, response):

    h3s = response.xpath('//h3')
    for h3 in h3s:
```

```
country = h3.xpath('span[@class="mw-headline"]/text()')\
.extract()
if country:
    winners = h3.xpath('following-sibling::ol[1]')
    for w in winners.xpath('li'):
        wdata = process_winner_li(w, country[0])
        ...
```

Работа с регулярными выражениями

> «Некоторые люди, сталкиваясь с одной проблемой, думают: "Знаю, что делать — нужно использовать регулярные выражения". И теперь у них две проблемы».
>
> Джейми Завински (Jamie Zawinskie)

Эта цитата — старая добрая классика, но она точно отражает отношение многих к регулярным выражениям (regex). Регулярные выражения — это последовательность символов, задающих шаблон для поиска совпадений в тексте. И в Python, и в JavaScript поддерживают regex на уровне языка.

В Python методы для работы с regex предоставляет модуль `re`. Например, найдем в документе email-адреса, распознавая их по шаблону *foo@bar.com*. Создадим regex для их поиска, разбив процесс на этапы[1]:

```
In [12]: txt = 'Feel free to contact me at '\ ' pyjdataviz@
kyrandale.com with any feedback.'

In [13]: re.findall(r'[\w\.-]+@[\w\.-]+', txt) Out[13]:
['pyjdataviz@kyrandale.com']
```

Метод `findall` первым аргументом принимает строку регулярного выражения (с префиксом *r*) и вторым — текст для поиска. Правила шаблона для поиска email:

```
\w    Соответствует цифре, латинской букве в любом регистре
      или подчеркиванию (диапазоны [0-9 a-z A-Z_])
\     Экранирует специальный символ
```

[1] Существуют удобные онлайн-инструменты для тестирования регулярных выражений, некоторые из них ориентированы на конкретные языки программирования. PyRegex http://www.pyregex.com/ — хороший инструмент для Python с удобной шпаргалкой.

SKLADCHIK.ORG

`\.`	Соответствует точке
`-`	Соответствует дефису
`+`	Соответствует одному или нескольким повторениям предыдущего элемента в квадратных скобках

Используя комбинацию этих правил, можно сопоставить строки, разделенные символом @ и содержащие буквы, цифры, точки или дефисы. Очевидно, что такой шаблон слишком общий (например `.@.` ему соответствует), поэтому требует уточнения. К примеру, для поиска только Gmail-адресов можно использовать `r'[\w\.-]@gmail.com`.

Хотя синтаксис регулярных выражений поначалу кажется сложным, они приспособлены для решения неожиданно возникающих задач и годятся для веб-скрейпинга, который часто заключается в сопоставлении беспорядочных и недостаточно определенных данных с шаблонами. Вполне вероятно, вы сможете обойтись без регулярных выражений, но если вы их поймете, то ваша жизнь заиграет новыми красками. См. пример 6-3 с несколькими регулярными выражениями.

В примере 6-3 показан метод `process_winner_li`. Словарь `wdata` заполняется данными, извлеченными из тега `li` лауреата с помощью двух регулярных выражений для поиска года и номинации (category).

Пример 6-3. *Обработка элемента списка лауреатов*

```
# ...
import re
BASE_URL = 'http://en.wikipedia.org'
# ...

def process_winner_li(w, country=None):
    """
    Обрабатывает тег <li> с данными о лауреате, добавляет страну
    рождения или национальность (если имеются)
    """
    wdata = {}
    # получить адрес ссылки из href тега <a>
    wdata['link'] = BASE_URL + w.xpath('a/@href').extract()[0] ❶
```

```
        text = ' '.join(w.xpath('descendant-or-self::text()')\
              .extract())
        # получить имя, разделенное запятыми, и удалить whitespace
        # в конце
        wdata['name'] = text.split(',')[0].strip()

        year = re.findall('\d{4}', text) ❷
        if year:
           wdata['year'] = int(year[0])
        else:
           wdata['year'] = 0
           print('Oops, no year in ', text)

        category = re.findall(
           'Physics|Chemistry|Physiology or Medicine|Literature|'\
           'Peace|Economics',
              text) ❸
        if category:
           wdata['category'] = category[0]
        else:
           wdata['category'] = category[0]
           print('Oops, no category in ', text)

        if country:
           if text.find('*')!= -1: ❹
              wdata['country'] = ''
              wdata['born_in'] = country
           else:
              wdata['country'] = country
              wdata['born_in'] = ''
        # сохраняем копию ссылки на случай ручного изменения
        # полученных данных
        wdata['text'] = text
        return wdata
```

❶ Для извлечения атрибута `href` из тега `<a>` элемента списка (`<li><a href=/wiki... >[winner name]</a>...`) используется символ @ в XPath, указывающий на атрибут.

- В этом примере для поиска строк с годом из четырех цифр в текстовом элементе списка используется `re` — встроенная Python-библиотека для работы с регулярными выражениями.
- Библиотека также используется для поиска в тексте номинации Нобелевской премии.
- Звездочка после имени победителя означает, что лауреат родился в указанной стране, но не являлся ее гражданином на момент присуждения премии (например `"William Lawrence Bragg*, Physics, 1915"`, «Уильям Лоуренс Брэгг*, Физика, 1915» в списке для Австралии).

Код из примера 6-3 возвращает все данные лауреатов, доступные на странице Википедии List of Nobel laureates by country («Список лауреатов Нобелевской премии по странам»). Это имя, год, номинация, страна (страна рождения или страна гражданства на момент присуждения премии) и ссылка на страницу с биографией лауреата. Нам потребуются эти ссылки, для скрейпинга недостающих данных (см. пример 6-1 и рисунок 6.1).

Скрейпинг биографических страниц лауреатов

Со страницы List of Nobel laureates by country мы получили большую часть данных, но пол, дату рождения и смерти (если она применима) нужно скрейпить дополнительно. Ожидается, что эта информация (по лауреатам-людям, а не организациям) присутствует на их биографических страницах — явно или неявно. Снова откроем вкладку Elements в Chrome и изучим эти страницы, чтобы понять, как извлечь нужные данные.

В предыдущей главе (глава 5) мы убедились, что явные информационные блоки не являются надежным источником информации, а часто отсутствуют. До недавних пор[1] доступ к данным — месту рождения, дате смерти и тому подобному — предоставляла скрытая таблица `persondata` (см. рисунок 6.4). К сожалению, этот удобный ресурс устарел[2]. Однако предпринята попытка улучшить категоризацию биографических данных, для них выделено отдельное пространство в Wikidata — центральном хранилище структурированных данных Википедии.

[1] Автора огорчило удаление этой возможности.

[2] Подробности см. в Википедии https://en.wikipedia.org/wiki/Wikipedia:Village_pump_(proposals)/Archive_122#RfC:_Should_Persondata_template_be_deprecated_and_methodically_removed_from_articles.3F.

```
</table>
<table id="persondata" class="[redacted] noprint" style="border:1px solid #a
<tr>
<th colspan="2"><a href="/wiki/Wikipedia:Persondata" title="Wikipedia:Person
</tr>
<tr>
<td class="persondata-label" style="color:#aaa;">Name</td>
<td>Röntgen, Wilhelm</td>
</tr>
<tr>
<td class="persondata-label" style="color:#aaa;">Alternative names</td>
<td>Conrad</td>
</tr>
```

Рис. 6.4. *Persondata — скрытая таблица лауреата Нобелевской премии*

При анализе биографических страниц в Википедии через вкладку Chrome Elements отображается ссылка на соответствующий элемент Wikidata (см. рисунок 6.5), которая ведет к биографическим данным на сайте *https://www.wikidata.org*. Перейдя по ссылке, мы можем извлечь все, что там найдем, возможно — основную часть целевых данных: важные даты и места (см. пример 6.1).

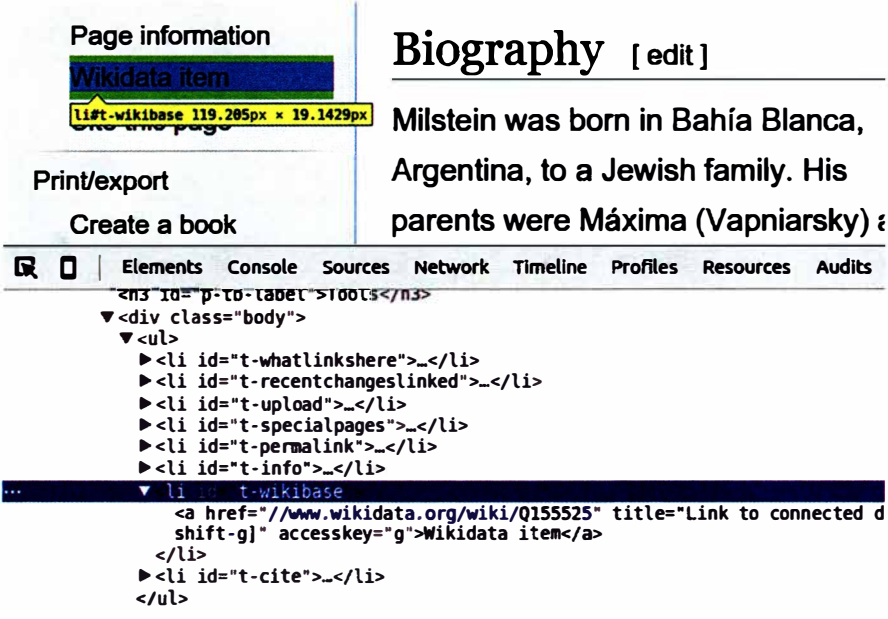


Рис. 6.5. *Гиперссылка на страницу лауреата в Wikidata*

При переходе на Wikidata открывается страница с полями данных, которые мы ищем, например, датой рождения лауреата. Как показано на рисунке 6.6, свойства встроены в автоматически сгенерированный вложенный HTML-код с сопутствующими кодами, которые можно использовать как идентификатор для скрейпинга (например, свойство «дата рождения» имеет код `P569`).

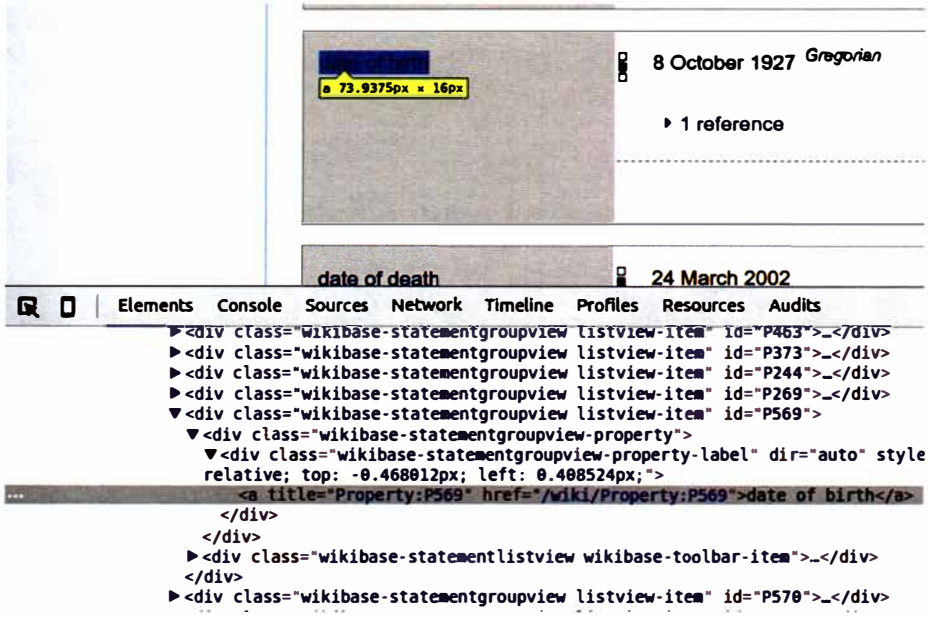


Рис. 6.6. Биографические свойства в Wikidata

На рисунке 6.7 показаны нужные нам данные (в этом случае — строка с датой), содержащиеся внутри соответствующего тега свойств во вложенной ветке HTML. Кликнув правой кнопкой по нужному `div`, можно сохранить XPath элемента и использовать его как указатель для получения данных с помощью Scrapy.



Рис. 6.7. Получение XPath для свойства Wikidata

Теперь, когда у нас есть XPath-пути для скрейпинга целевых данных, объединим их и посмотрим, как Scrapy связывает запросы в цепочку для сложного скрейпинга множества страниц.

Цепочка запросов и извлечение данных

В данном подразделе мы рассмотрим, как создавать цепочку запросов Scrapy, чтобы переходить по гиперссылкам и по пути извлекать данные. Для начала активируем в Scrapy кеширование страниц. Во время экспериментов с XPath-целями хорошим тоном будет сохранять скачанные страницы, чтобы минимизировать запросы к Википедии. В отличие от многих наборов данных, данные о лауреатах Нобелевской премии обновляются лишь раз в год[1].

Кеширование страниц

Как и ожидалось, Scrapy имеет продвинутую систему кеширования, предоставляющую полный контроль над кешированием страниц (например, можно выбирать систему хранения: базы данных или файлы, определять срок хранения страниц и так далее). Кеширование реализовано как middleware (промежуточное, или связующее ПО), настройки которого задаются в модуле `settings.py` нашего проекта. Доступны различные опции, но для скрейпинга данных о лауреатах достаточно установить `HTTPCACHE_ENABLED` в значение `True`:

```
# -*- coding: utf-8 -*-

# Настройки Scrapy для проекта nobel_winners #
# Этот файл содержит только самые важные настройки по # умолчанию.
# Все остальные настройки документируются здесь:
#
#  http://doc.scrapy.org/en/latest/topics/settings.html #

BOT_NAME = 'nobel_winners'

SPIDER_MODULES = ['nobel_winners.spiders']
NEWSPIDER_MODULE = 'nobel_winners.spiders'
```

[1] Хотя правки в Википедии вносятся постоянно, но основные детали остаются неизменными до следующих награждений.

```
# Рекомендуется указывать себя
# (и свой сайт) в user-agent
#USER_AGENT = 'nobel_winners (+http://www.yourdomain.com)'

HTTPCACHE_ENABLED = True
```

См. полный перечень Scrapy middleware в документации Scrapy. После настройки кеша перейдем к цепочкам запросов в Scrapy.

Генерация запросов

Метод `parse` нашего паука перебирает в цикле лауреатов Нобелевской премии, используя метод `process_winner_li`, чтобы скрейпить поля страны, года, имени, номинации и гиперссылок на биографии. Теперь мы используем гиперссылки на биографию для создания запроса Scrapy, который извлечет страницы биографии и отправит их в пользовательский метод для скрейпинга.

Для организации цепочки запросов Scrapy применяет паттерн Python, используя ключевое слово `yield`, которое создает генератор[1], позволяющий Scrapy легко потреблять любые наши добавочные запросы к странице. В примере 6-4 показан этот паттерн в действии.

***Пример 6-4.** Генерация запроса с помощью Scrapy*

```
class NWinnerSpider(scrapy.Spider):
    name = 'nwinners_full'
    allowed_domains = ['en.wikipedia.org']
    start_urls = [
        "https://en.wikipedia.org/wiki/List_of_Nobel_laureates" \
        "_by_country"
    ]

    def parse(self, response):

        h3s = response.xpath('//h3')
        for h3 in h3s:
            country = h3.xpath('span[@class="mw-headline"]/text()')
                    .extract()
```

[1] См. в блоге Джеффа Кнаппа (Jeff Knupp) "Everything I Know About Python" («Все, что я знаю о Python») прекрасный обзор генераторов Python и использования yield.

```
            if country:
                winners = h2.xpath('following-sibling::ol[1]')
                for w in winners.xpath('li'):
                    wdata = process_winner_li(w, country[0])
                    request = scrapy.Request( ❶
                        wdata['link'],
                        callback=self.parse_bio, ❷
                        dont_filter=True)
                    request.meta['item'] = NWinnerItem(**wdata) ❸
                    yield request ❹

    def parse_bio(self, response):
        item = response.meta['item'] ❺
        ...
```

❶ Делает запрос к странице с биографией лауреата, используя ссылку (`wdata[`*`link`*`]`), извлеченную функцией `process_winner_li`.

❷ Устанавливает коллбэк-функцию для обработки ответа.

❸ Создает Scrapy `Item` для хранения данных лауреатов и инициализирует его данными, только что извлеченными из `process_winner_li`. Данные `Item` прикрепляются к метаданным запроса, что позволяет любому ответу получить доступ к ним.

❹ yield request превращает метод `parse` в генератор потребляемых запросов.

❺ Этот метод обрабатывает коллбэк-функцию из нашего запроса биографии по ссылке. Чтобы добавить извлеченные данные в Scrapy `Item`, мы сначала получим их из метаданных `response`.

Анализ страниц Википедии в подразделе «Скрейпинг биографических страниц лауреатов» показал, что требуется извлечь Wikidata-ссылку из биографии лауреата для генерации запроса. Затем мы извлекаем из ответа данные: дату, место и пол.

В примере 6-5 показаны два метода для скрейпинга биографических данных лауреатов — `parse_bio` и `parse_wikidata`. Метод `parse_bio` использует извлеченную ссылку Wikidata, затем так же, как метод `parse`, генерирует `request` для запроса страницы. В конце цепочки запросов метод `parse_wikidata` получает response и заполняет поля объекта `item` данными Wikidata, и возвращает его через `yield`.

Пример 6-5. *Парсинг биографических данных лауреата*

```
# ...

    def parse_bio(self, response):

        item = response.meta['item']
        href = response.xpath("//li[@id='t-wikibase']/a/@href") ❶
              .extract()
        if href:
            url = href[0] ❷
            wiki_code = url.split('/')[-1]
            request = scrapy.Request(href[0],\
                      callback=self.parse_wikidata,\ ❸
                      dont_filter=True)
            request.meta['item'] = item
            yield request

    def parse_wikidata(self, response):

        item = response.meta['item']
        property_codes = [ ❹
            {'name':'date_of_birth', 'code':'P569'},
            {'name':'date_of_death', 'code':'P570'},
            {'name':'place_of_birth', 'code':'P19', 'link': True},
            {'name':'place_of_death', 'code':'P20', 'link': True},
            {'name':'gender', 'code':'P21', 'link': True}
        ]

    for prop in  property_codes:

        link_html = ''
        if prop.get('link'):
            link_html = '/a'
        # Выбрать div с id кода свойства
        code_block = response.xpath('//*[@id="%s"]'%(prop['code']))
        # продолжать, если code_block существует
        if code_block:
        # Мы можем использовать css-селектор вышестоящего класса
        # selection
```

```
            values = code_block.css('.wikibase-snakview-value')
        # первое значение соответствует коду свойства\
        # (например, '10 August 1879')
            value = values[0]
            prop_sel = value.xpath('.%s/text()'%link_html)
            if prop_sel:
                item[prop['name']] = prop_sel[0].extract()

    yield item ❺
```

❶ Извлекает ссылку на Wikidata, показанную на рисунке 6.5.

❷ Извлекает `wiki_code` из URL, например *http://wikidata.org/wiki/Q155525* → Q155525.

❸ Использует ссылку Wikidata для генерации запроса с помощью метода нашего паука `parse_wikidata` в качестве обратного вызова для обработки ответа.

❹ Это коды свойств, которые мы нашли ранее (см. рисунок 6.6), с именами, соответствующими полям нашего элемента Scrapy, `NWinnerItem`. Те, что имеют атрибут `True link` содержатся в тегах <a>.

❺ Наконец, мы возвращаем item, который на данный момент должен содержать все целевые данные, которые доступны в Википедии.

Получив цепочку запросов, проверим, извлекает ли паук нужные нам данные:

```
$ scrapy crawl nwinners_full
2021-...    [scrapy] ... started (bot: nobel_winners)
...
2021-...    [nwinners_full] DEBUG: Scraped from
            <200 https://www.wikidata.org/wiki/Q155525>
    {'born_in': '',
     'category': u'Physiology or Medicine',
     'date_of_birth': u'8 October 1927',
     'date_of_death': u'24 March 2002',
     'gender': u'male',
     'link': u'http://en.wikipedia.org/wiki/C%C3%A9sar_Milstein',
     'name': u'C\xe9sar Milstein',
     'country': u'Argentina',
     'place_of_birth': u'Bah\xeda Blanca',
```

SKLADCHIK.ORG

```
    'place_of_death': u'Cambridge',
    'text': u'C\xe9sar Milstein, Physiology or Medicine, 1984',
    'year': 1984}
2021-...   [nwinners_full] DEBUG: Scraped from
           <200 https://www.wikidata.org/wiki/Q193672>
  {'born_in': '',
   'category': u'Peace',
   'date_of_birth': u'1 November 1878',
   'date_of_death': u'5 May 1959',
   'gender': u'male',
   'link': u'http://en.wikipedia.org/wiki/Carlos_Saavedra_Lamas',
  ...
```

Выглядит отлично. За исключением поля `born_in`, которое зависит от того, отмечено ли звездочкой имя в основном списке лауреатов Нобелевской премии в Википедии, мы получили все целевые данные. Набор данных готов к очистке с помощью pandas, чем мы займемся в следующей главе.

Мы собрали основные биографические данные о лауреатах Нобелевской премии, теперь соберем оставшиеся: тексты биографии и, если есть, фотографии этих выдающихся мужчин и женщин.

Конвейеры Scrapy

Краткие биографические справки и фотографии лауреатов добавят индивидуальности нашей визуализации. В основном на биографических страницах такие данные есть, так что займемся их скрейпингом.

До сих пор мы имели дело со скрейпингом строк текста. Для скрейпинга изображений формата используется *конвейер* Scrapy. Конвейеры выполняют постобработку полученных элементов. Число используемых конвейеров не ограничено. Вы можете написать свой собственный или применить предоставленные Scrapy, например `ImagesPipeline`, который мы и будем использовать.

В самом простом случае для конвейера требуется всего лишь определить метод `process_item`. Он принимает полученные с помощью скрепинга объекты item и объект паука. Напишем небольшой конвейер для исключения лауреатов, для которых не указан пол (чтобы исключить организации), с помощью паука `nwinners_full`. Сначала добавим конвейер `DropNonPersons` в модуль `pipelines.py` нашего проекта.

```
# nobel_winners/nobel_winners/pipelines.py

# Определяйте свои конвейеры элементов здесь
#
# Не забудьте добавить конвейер в настройки ITEM_PIPELINES
# См. http://doc.scrapy.org/en/latest/topics/item-pipeline.html

from scrapy.exceptions import DropItem


class DropNonPersons(object):
    """ Исключение организаций-лауреатов """

    def process_item(self, item, spider):
        if not item['gender']: ❶
            raise DropItem("No gender for %s"%item['name'])
        return item ❷
```

❶ Если в элементе нет поля gender, вероятно, это организация (например, Красный крест). Поскольку визуализация касается лауреатов-людей, то мы исключаем такие элементы через `DropItem`.

❷ Нам нужно вернуть объект item для следующих конвейеров или для сохранения средствами Scrapy.

Как упоминалось в заголовке `pipelines.py`, чтобы добавить конвейер к паукам нашего проекта, необходимо указать его в словаре конвейеров в модуле `settings.py` и активировать (1):

```
# nobel_winners/nobel_winners/settings.py

BOT_NAME = 'nobel_winners'
SPIDER_MODULES = ['nobel_winners.spiders']
NEWSPIDER_MODULE = 'nobel_winners.spiders'

HTTPCACHE_ENABLED = True
ITEM_PIPELINES = {'nobel_winners.pipelines.DropNonPersons':300}
```

Теперь у нас есть базовое воркфлоу для конвейеров, добавим один конвейер в наш проект.

Скрейпинг текста и изображений с помощью конвейера

Извлечем доступные биографии и фотографии лауреатов (см. рисунок 6.1). Для скрейпинга текста биографий мы используем паука, а для фотографий — конвейер изображений.

Мы можем просто написать собственный конвейер, который берет URL-адрес изображения, запрашивает его из Википедии и сохраняет на диске, но чтобы сделать все правильно, требуется определенная осторожность. Например, желательно избегать повторного скачивания изображений, которые недавно скачивали или которые не менялись со времени скачивания. Гибкость в указании места хранения изображений — полезная фича. Также неплохо иметь возможность конвертировать изображения в стандартные форматы (например, JPG или PNG) или создавать миниатюры (thumbnails). К счастью, Scrapy предоставляет объект `ImagesPipeline`, соответствующий всем этим требованиям. Это часть системы конвейеров медиафайлов, куда также входит конвейер файлов `FilesPipeline`

Добавив функции скрейпинга изображений и текста биографии пауку `nwinners_full`, мы слишком его нагрузим. Логичнее отделить персональные данные от более формальных. Поэтому мы создадим нового паука `nwinners_minibio`, который будет использовать части метода `parse` предыдущего паука, чтобы пройти в цикле по лауреатам.

Как обычно, при создании Scrapy-паука для начала нужно получить целевой XPath для скрейпинга. В данном случае — это первая часть текста биографии лауреата и фотография (если имеются). Откроем в Chrome вкладку Elements для анализа HTML-структуры биографий (см. рисунок 6.8).

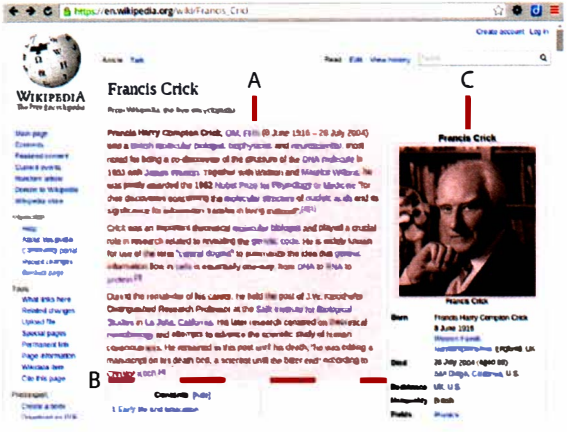


Рис. 6.8. *Целевые элементы для скрейпинга биографий: первая часть биографического текста (A) до точки останова (B) и фотография лауреата (C)*

Пример 6-6. *Скрейпинг биографического текста*

```
<div id="mw-content-text">
   <div class="mw-parser-output">
      ...
      <table class="infobox biography vcard">...</table>
      /* целевые абзацы: */
      <p>...</p>
      <p>...</p>
      <p>...</p>
      <div id="toc">...</div>
   ...
   </div>
</div>
```

Анализ с помощью Chrome Elements (пример 6-6) показывает, что текст биографии (см. рисунок 6.8 A) содержится в дочерних абзацах блока `div` с классом `mw-parser-output`, который, в свою очередь, является дочерним для `div` с ID `mw-content-text`. Абзацы вложены между `table` с классом `infobox` и блоком содержания `div` с id `toc`. Мы можем использовать операторы XPath `following-sibling` и `preceding-sibling`, чтобы создать селекторы для выбора целевых абзацев:

```
ps = response.xpath(\
   '//*[@id="mw-content-text"]/div/table/following-sibling::p' ❶
   '[not(preceding-sibling::div[@id="toc"])]').extract() ❷
```

❶ Все абзацы, следующие за первой таблицей внутри блока `div`, дочернего для `div` с ID `mw-content-text`.

❷ Исключение (not) всех абзацев, следующих за родственным элементом `div` с ID `toc`.

Тестирование в Scrapy Shell подтверждает стабильный сбор мини-биографий.

Дальнейшее изучение страниц лауреатов показывает, что их фотографии (рисунок 6.8 C) содержатся в таблице с классом `infobox` и являются единственными тегами изображений (`<img>`) в таблице:

```
<table class="infobox biography vcard">
   ...
```

```
      <img alt="Francis Crick crop.jpg" src="//upload..." />
    ...
  </table>
```

Указав XPath `'//table[contains(@class,"infobox")]//img/@src`, мы получим исходный URL изображения.

Так же, как для первого паука, сначала нужно объявить Scrapy `Item`, для хранения полученных данных. Мы извлечем ссылку на биографию и имя лауреата для использования к качестве идентификаторов изображения и текста. Нам нужно где-то хранить `image-urls` (хотя мы извлечем только одно изображение, я расскажу о сценарии для нескольких изображений), ссылки на полученные изображения (путь к файлу) и поле `bio_image` для хранения целевого изображения:

```
import scrapy
import re

BASE_URL = 'http://en.wikipedia.org'


class NWinnerItemBio(scrapy.Item):
    link = scrapy.Field()
    name = scrapy.Field()
    mini_bio = scrapy.Field()
    image_urls = scrapy.Field()
    bio_image = scrapy.Field()
    images = scrapy.Field()
...
```

Теперь снова используем цикл скрейпинга данных лауреатов (см. пример 6-4), передавая запрос новому методу `get_mini_bio` для сбора URL изображений и текста биографии:

```
class NWinnerSpiderBio(scrapy.Spider):

    name = 'nwinners_minibio'
    allowed_domains = ['en.wikipedia.org']
    start_urls = [
        "https://en.wikipedia.org/wiki/List_of_Nobel_" \
```

```
        "laureates_by_country"
    ]
    def parse(self, response):

        filename = response.url.split('/')[-1]
        h3s = response.xpath('//h3')

        for h3 in h3s:
            country = h3.xpath('span[@class="mw-headline"]'\
            'text()').extract()
            if country:
                winners = h3.xpath('following-sibling::ol[1]')
                for w in winners.xpath('li'):
                    wdata = {}
                    wdata['link'] = BASE_URL + \
                    w.xpath('a/@href').extract()[0]
                    # Обработка страницы биографии лауреата
                    # методом get_mini_bio
                    request = scrapy.Request(wdata['link'],
                                callback=self.get_mini_bio)
                    request.meta['item'] = NWinnerItemBio(**wdata)
                    yield request
```

Метод `get_mini_bio` добавит URL всех доступных фотографий в список `image_urls` и абзацы биографии до точки останова `<p></p>` в поле элемента `mini_bio`:

```
...
    def get_mini_bio(self, response):
        """ Получение текста биографии и фото лауреата """

        BASE_URL_ESCAPED = 'http:\/\/en.wikipedia.org'
        item = response.meta['item']
        item['image_urls'] = []
        img_src = response.xpath(\
              '//table[contains(@class,"infobox")]//img/@src') ❶
        if img_src:
            item['image_urls'] = ['http:' +\
            img_src[0].extract()]
```

```
        ps = response.xpath(
              '//*[@id="mw-content-text"]/div/table/'
              'following-sibling::p[not(preceding-sibling:: \
              div[@id="toc"])]')\
              .extract() ❷
        # Конкатенация абзацев биографии в строку mini_bio
        mini_bio = ''
        for p in ps:
           mini_bio += p
        # корректировка для полей wiki-link
        mini_bio = mini_bio.replace('href="/wiki', 'href="'
                   + BASE_URL + '/wiki"') ❸
        mini_bio = mini_bio.replace('href="#',\
           'href="' + item['link'] + '#"')
        item['mini_bio'] = mini_bio
        yield item
```

❶ Определяет первое (и единственное) изображение в таблице класса `infobox`, извлекая атрибут его источника (`src`), (например `<img src='//upload.wikimedia.org/.../ Max_Perutz.jpg'...`).

❷ Захватывает абзацы mini-bio между соседними родственными элементами.

❸ Заменяет внутренние ссылки Википедии (например */wiki/...*) на полные URL для визуализации.

После создания паука для скрейпинга биографий нужно создать конвейер, который принимает извлеченные URL изображений и сохраняет изображения. Используем встроенный в Scrapy конвейер для обработки изображений ImagesPipeline.

Класс `ImagesPipeline` (пример 6-7) содержит два основных метода — `get_media_requests` для генерации запросов URL и `item_completed` для обработки результатов.

Пример 6-7. *Скрейпинг изображений с помощью конвейера*

```
import scrapy
from itemadapter import ItemAdapter
from scrapy.pipelines.images import ImagesPipeline
from scrapy.exceptions import DropItem
```

```
class NobelImagesPipeline(ImagesPipeline):

    def get_media_requests(self, item, info): ❶

        for image_url in item['image_urls']:
            yield scrapy.Request(image_url)

    def item_completed(self, results, item, info): ❷

        image_paths = [img['path'] for ok, img in results if ok] ❸

        if not image_paths:
            raise DropItem("Item contains no images")
        adapter = ItemAdapter(item) ❹
        adapter['bio_image'] = image_paths[0]

        return item
```

❶ Принимает URL-адреса изображений, полученные пауком *nwinners_minibio*, и генерирует HTTP-запросы для получения их содержимого.

❷ Передает результат запроса по URL изображений методу `item_completed`.

❸ Этот генератор списков Python фильтрует список кортежей (вида `[(True, Image), (False, Image) ...]`), сохраняя пути к файлам успешно загруженных изображений, относительно каталога, указанного в переменной `IMAGES_STORE` из `settings.py`.

❹ Используем класс Scrapy ItemAdapter, который предоставляет общий интерфейс для доступа к поддерживаемым типам элементов.

После определения паука и конвейера нужно добавить конвейер в модуль `settings.py` и задать в переменной `IMAGES_STORE` каталог, куда будем сохранять изображения:

```
# nobel_winners/nobel_winners/settings.py

...
ITEM_PIPELINES = {'nobel_winners.pipelines'\
                  '.NobelImagesPipeline':300}
IMAGES_STORE = 'images'
```

Запустим паука из корневого каталога *nobel_winners* нашего проекта и проверим результат:

```
$ scrapy crawl nwinners_minibio -o minibios.json
...
2021-12-13 17:18:05 [scrapy.core.scraper] DEBUG: Scraped from
     <200 https://en.wikipedia.org/wiki/C%C3%A9sar_Milstein>

{'bio_image': 'full/65ac9541c305ab4728ed889385d422a2321a117d.jpg',
'image_urls': ['http://upload.wikimedia...
        150px-Milstein_lnp_restauraci%C3%B3n.jpg'],
'link': 'http://en.wikipedia.org/wiki/C%C3%A9sar_Milstein',
'mini_bio': '<p><b>César Milstein</b>, <a ...'
            'href="http://en.wikipedia.org/wiki/Order_of_the_...'
            'title="Order of the Companions of Honour">CH</a>'
            'href="http://en.wikipedia.org/wiki/Royal_Society'
            'Society">FRS</a><sup id="cite_ref-frs_2-1" class'...>
            'href="http://en.wikipedia.org/wiki/C%C3%A9sar_Mi'
            '(8 October 1927-24 March 2002) was an <a ...>'
            'href="http://en.wikipedia.org/wiki/Argentine" '

...
```

Паук правильно собрал мини-биографии лауреатов, а также — с помощью конвейера изображений — их фотографии. Изображение, URL которого сохранен в `image_urls` и успешно обработан, загружается в формате JPG в каталог *images*, указанный в `IMAGE_STORE`, с относительным путем (`full/a5f763b828006e704cb291411b8b643bfb1886c.jpg`). Имя файла — SHA-1 хэш URL-адреса изображения, что позволяет конвейеру изображений, проверяя существующие изображения, избежать повторных загрузок.

В каталоге images мы видим полный набор изображений лауреатов, готовый для веб-визуализации:

```
$ (nobel_winners) tree images images
└── full
    ├── 0512ae11141584da1262661992a1b05dfb20dd52.jpg
    ├── 092a92689118c16b15b1613751af422439df2850.jpg
    ├── 0b6a8ca56e6ff115b7d30087df9c21da09684db1.jpg
    ├── 1197aa95299a1fec983b3dbdeaeb97a1f7e545c9.jpg
    ├── 1f6fb8e9e2241733da47328291b25bd1a78fa588.jpg
    ├── 272cf1b089c7a28ea0109ad8655bc3ef1c03fb52.jpg
    ├── 28dcc7978d9d5710f0c29d6dfcf09caa7e13a1d0.jpg
    ...
```

В главе 16 мы поместим их в каталог *static* нашего веб-приложения для доступа через поле `bio_image`.

Итак, мы выполнили успешный скрейпинг всех элементов, которые выбрали целью в начале главы (см. пример 6-1 и рисунок 6.1), получив изображения и тексты биографий. Прежде чем перейти к очистке заведомо грязных данных с помощью pandas, подведем краткий итог.

Настройка конвейеров для каждого из пауков

Активированные конвейеры из `settings.py` применяются ко всем паукам проекта Scrapy. При наличии в проекте нескольких пауков можно настроить конкретные конвейеры для каждого. Для этого существует несколько способов, но наиболее оптимально использовать атрибут `custom_settings` класса паука — словарь настроек для `ITEM_PIPELINES`, — вместо настройки в `settings.py`. В таком случае для паука `nwinners_minibio` класс `NWinnerSpiderBio` переопределится следующим образом:

```
class NWinnerSpiderBio(scrapy.Spider):
    name = 'nwinners_minibio'
    allowed_domains = ['en.wikipedia.org']
    start_urls = [
        "http://en.wikipedia.org/wiki"\
        "List_of_Nobel_laureates_by_country"
    ]

    custom_settings = {
        'ITEM_PIPELINES':\
        {'nobel_winners.pipelines.NobelImagesPipeline':1}
    }

    # ...
```

Теперь конвейер `NobelImagesPipeline` активируется только при скрейпинге биографий.

SKLADCHIK.ORG

Резюме

В этой главе мы создали двух пауков Scrapy, которые собрали статистические данные о лауреатах Нобелевской премии, а также биографические тексты и фотографии, чтобы немного расцветить сухую статистику. Scrapy — мощная библиотека, которая предоставляет полный набор средств для скрейпинга. Хотя работа с ней требует больших усилий, по сравнению с Beautiful Soup, Scrapy предоставляет гораздо больше возможностей и становится незаменимой при выполнении сложных задач. Все пауки Scrapy следуют показанным здесь стандартным инструкциям, и после того, как вы запрограммируете несколько штук, рабочий процесс сделается привычным.

Надеюсь, эта глава показала итеративную природу скрейпинга, порой требующего нестандартных решений, и то удовлетворение, которое чувствуешь, получая относительно относительно чистые данные из малоперспективной груды веб-данных. Проблема в том, что и сейчас, и в обозримом будущем большинство интересных данных (основы для искусства и науки визуализации данных) заключено в форматы, непригодные для веб-визуализаций, которым посвящена эта книга. Скрейпинг в этом контексте становится инструментом подготовки данных к преобразованию.

Собранные нами данные, в основном отредактированные людьми, неизбежно содержат ошибки: некорректные форматы дат, различные аномалии, пропущенные поля. В следующих главах основное внимание уделяется тому, как с помощью pandas сделать эти данные пригодными для презентации. Но сначала будет небольшое введение в pandas и NumPy.

РАЗДЕЛ III

Очистка и исследование данных с помощью pandas

В этом разделе книги, на втором этапе использования нашего тулчейна (см. рисунок III-1), мы возьмем набор данных о нобелевских лауреатах, полученный с помощью скрейпинга в главе 6, очистим его, а затем исследуем на наличие интересных закономерностей. Основными инструментами будут крупные Python-библиотеки Matplotlib и pandas.



Во второй редакции книги используется тот же набор данных о Нобелевских лауреатах, что и в первой. Автор счел целесообразным сосредоточиться на обновлении материалов и библиотек, сохранив те же подходы к исследованию и анализу, что и в первом издании. Визуализация обычно оперирует давно сформированными наборами данных, и добавление новых лауреатов не влияет на методологию.

В следующих двух главах мы познакомимся с pandas и ее строительным блоком NumPy. В главе 9 мы используем pandas для очистки данных. В главе 11 мы проанализируем эти данные, используя pandas совместно с Python-библиотекой Matplotlib.

В разделе IV мы рассмотрим доставку очищенных данных в браузер с помощью веб-сервера Flask.

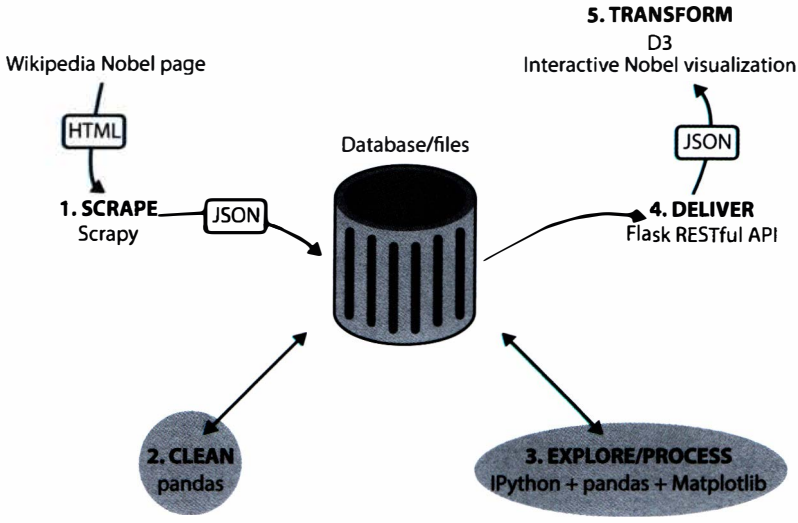


Рис. III-1. *Наш тулчейн для визуализации: очистка и анализ данных*



Исходный код для этого раздела доступен в репозитории книги на GitHub https://github.com/Kyrand/dataviz-with-python-and-js-ed-2.

ГЛАВА 7

Введение в NumPy

Цель этой главы — познакомить с библиотекой NumPy (Numeric Python library) тех, кто с ней не знаком. NumPy — ключевой строительный блок pandas, мощной библиотеки для анализа данных, которую мы будем использовать для очистки и анализа набора данных о нобелевских лауреатах (см. главу 6). Чтобы максимально эффективно использовать pandas, важно иметь базовое представление об основных элементах и принципах NumPy. Поэтому основная цель главы — заложить основу для предстоящего знакомства с pandas.

NumPy — библиотека на Python, которая очень быстро работает с многомерными массивами, благодаря использованию библиотек на низкоуровневых языках C и Fortran[1]. Собственная производительность Python при работе с большими объемами данных относительно низкая, но NumPy позволяет выполнять параллельные операции над большими массивами одновременно, что ощутимо ускоряет процесс. Учитывая, что NumPy является основой большинства Python-библиотек обработки данных, включая pandas, ее статус как фундамента Python-аналитики неоспорим.

Помимо pandas экосистема NumPy включает: SciPy для сложных научных и инженерных расчетов, scikit-learn с алгоритмами машинного обучения (классификация, извлечение признаков) и другие специализированные библиотеки, которые используют многомерные массивы NumPy в качестве основных объектов данных. С этой точки зрения, базовое владение NumPy может значительно расширить ваши познания в области обработки данных на Python.

Ключевой концепцией библиотеки NumPy являются массивы. Если вы понимаете, как они работают и как ими манипулировать, то по поводу остальных вещей в NumPy не стоит беспокоиться[2]. Далее мы рассмотрим основные операции

[1] Писать код на Python просто и быстро, но за это пришлось пожертвовать скоростью выполнения. Оборачивая быстрые библиотеки, написанные на низкоуровневых языках, NumPy нацелена на программирование, лишенное всяческого мусора, и на выдающуюся производительность.

[2] NumPy применяют для очень сложных математических расчетов, поэтому не стоит ожидать, что вы поймете все, что увидите в сети, постарайтесь понять основные строительные блоки.

с массивами на нескольких примерах NumPy в действии — это подготовит почву для знакомства с типами данных pandas в главе 8.

Массив NumPy

Вся функциональность NumPy построена на основе однородных[1] многомерных массивов `ndarray` (N-dimensional array). Операции с этими массивами выполняются быстрыми скомпилированными низкоуровневыми библиотеками, что обеспечивает производительность NumPy на порядки выше нативного Python. Помимо прочего, арифметические операции с массивами выполняются так же, как с `int` или `float` в Python[2]. В примере ниже целый массив прибавляется сам к себе так же легко и быстро, как складываются два целых чисел:

```
import numpy as np ❶

a = np.array([1, 2, 3]) ❷
a + a
# результат: array([2, 4, 6])
```

❶ Типичный способ импорта NumPy предпочтительнее, чем `"from numpy import *"`[3].

❷ Автоматически преобразует список Python с числами.

Под капотом NumPy использует массовые параллельные вычисления, доступные современным CPU, что позволяет обрабатывать, например, большие матрицы (двумерные массивы) за приемлемое время.

Основные свойства NumPy `ndarray` — размерность (`ndim`), форма (`shape`) и тип данных (`dtype`). У того же массива чисел можно преобразовать форму, что иногда влечет за собой изменение размерности массива. Продемонстрируем изменение формы на примере восьмиэлементного массива. Используем метод `print_array_details`, чтобы вывести на экран основные свойства массива:

---

[1] Это значит, что массив NumPy составляют элементы одного типа (dtype) в отличие от Python, где в список могут одновременно входить, например: строки, числа, даты и так далее.

[2] Предполагается, что массивы соответствуют ограничениям по форме и типу данных.

[3] Импорт всех переменных модуля в ваше пространство имен с помощью * почти всегда плохая идея.

```
def print_array_details(a):
    print('Dimensions: %d, shape: %s, dtype: %s'\
        %(a.ndim, a.shape, a.dtype))
```

Для начала создадим одномерный массив. Как показывают выведенное на экран сообщение, по умолчанию используется 64-битный целочисленный тип (`int64`):

```
In [1]: a = np.array([1, 2, 3, 4, 5, 6, 7, 8])

In [2]: a
Out[2]: array([1, 2, 3, 4, 5, 6, 7, 8])

In [3]: print_array_details(a)
Dimensions: 1, shape: (8,), dtype: int64
```

Метод `reshape` изменяет форму и размерность массива `a`. Преобразуем `a` в двумерный массив, состоящий из двух 4-элементных массивов:

```
In [4]: a = a.reshape([2, 4])
In [5]: a
Out[5]:
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])

In [6]: print_array_details(a)
Dimensions: 2, shape: (2, 4), dtype: int64
```

Массив из восьми элементов можно преобразовать в трехмерный массив:

```
In [7]: a = a.reshape([2, 2, 2])

In [8]: a
Out[8]:
array([[[1, 2],
        [3, 4]],

       [[5, 6],
        [7, 8]]])
```

```
In [9]: print_array_details(a)
Dimensions: 3, shape: (2, 2, 2), dtype: int64
```

Форму и числовой тип можно указать при создании массива или позже. Самый простой способ изменить числовой тип массива — использовать метод astype для создания копии массива с новым типом данных[1]:

```
In [0]: x = np.array([[1, 2, 3], [4, 5, 6]], np.int32) ❶
In [1]: x.shape
Out[1]: (2, 3)
In [2]: x.shape = (6,)
In [3]: x
Out[3]: array([1, 2, 3, 4, 5, 6], dtype=int32)
In [4] x = x.astype('int64')
In [5]: x.dtype
Out[5]: dtype('int64')
```

❶ Вложенный список чисел преобразуется в многомерный массив.

Создание массивов

Помимо создания массивов из списков с числами, NumPy предоставляет вспомогательные функции для создания массивов требуемой формы. Чаще всего используются функции `zeros` и `ones`, которые создают массивы, заполненные этими значениями. Рассмотрим пару примеров. Обратите внимание, что тип данных по умолчанию (`dtype`) для методов создания массивов — `float64` (64-битное число с плавающей точкой):

```
In [32]: a = np.zeros([2,3]) In [33]: a
Out[33]:
array([[0., 0., 0.],
       [0., 0., 0.]])

In [34]: a.dtype
Out[34]: dtype('float64')
```

[1] Оптимизация памяти и производительности достигается через работу с представлениями, но такая оптимизация требует дополнительных действий.

```
In [35]: np.ones([2, 3])
Out[35]:
array([[1., 1., 1.],
       [1., 1., 1.]])
```

Метод `empty` работает быстрее, поскольку, создавая массив, не инициализирует элементы сам, а передает эту задачу вам. То есть, в отличие от np.zeros, значения массива остаются неопределенными, что требует осторожности:

```
empty_array = np.empty((2,3)) # создает неинициализированный массив

empty_array
Out[3]:
array([[ 6.93185732e-310, 2.52008024e-316, 4.71690401e-317],
       [ 2.38085057e-316, 6.93185752e-310, 6.93185751e-310]])
```

Еще одна полезная вспомогательная функция `random` находится в модуле NumPy `random` вместе с родственными функциями. Создание массива со случайными элементами:

```
>>> np.random.random((2,3))
>>> Out:
array([[0.97519667, 0.94934859, 0.98379541], ❶
       [0.10407003, 0.35752882, 0.62971186]])
```

❶ Массив 2×3 из случайных чисел в диапазоне 0 <= x < 1.

Функция `linspace` возвращает заданное количество равномерно расположенных чисел в пределах заданного интервала. Функция `arange` действует похоже, но использует аргумент step, задающий шаг внутри интервала.

```
np.linspace(2, 10, 5) # 5 чисел в интервале 2-10
Out: array([2., 4.,6., 8., 10.]) ❶

np.arange(2, 10, 2) # от 2 до 10 (не включая) с шагом 2.
Out: array([2, 4, 6, 8])
```

Обратите внимание, что у `linspace`, в отличие `arange`, тип данных по умолчанию `float64`, и верхняя граница включена в интервал.

Индексация и срезы массива

Индексация и срезы одномерных массивов во многом похожи на операции со списками Python:

```
a = np.array([1, 2, 3, 4, 5, 6]) a[2] # Вывод: 3
a[3:5] # Вывод: массив([4, 5])
# каждый второй элемент от 0 до 4 установить в 0
a[:4:2] = 0 # Вывод: массив([0, 2, 0, 4, 5, 6])
a[::-1] # Вывод: массив([6, 5, 4, 0, 2, 0]), перевернутый
```

Индексация многомерных массивов аналогична *одномерным*. Для каждого измерения задается отдельная операция индексации/срезов, параметры которых указываются в кортеже, разделенном запятыми[1] (см. рисунок 7.1).

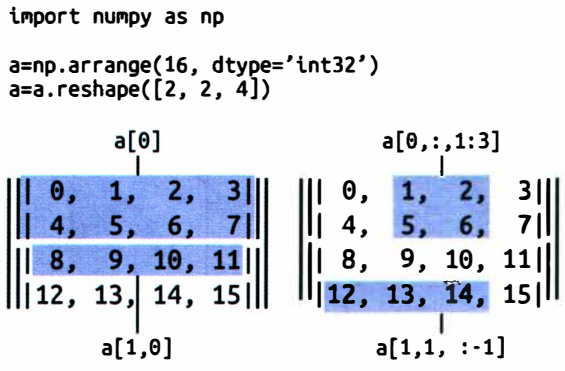


Рис. 7.1. *Индексация многомерных массивов в NumPy*

Обратите внимание, если число элементов в кортеже меньше числа измерений, оставшиеся оси считаются полностью выбранными (:). Для сокращения записи среза также можно использовать многоточие (ellipsis) вместо полного набора индексов — оно обозначает наличие стольких объектов ':', сколько необходимо. Для демонстрации используем трехмерный массив:

```
a = np.arange(8) a.shape = (2, 2, 2)
a Out:
array([[[0, 1],
        [2, 3]],
```

[1] Существует короткая точечная нотация для выбора всех индексов, например `[..1:3]`

SKLADCHIK.ORG

```
       [[4, 5],
        [6, 7]]])
```

В NumPy есть удобный метод `array_equal` для сравнения массивов по форме и по элементам. Используем его, чтобы показать эквивалентность выборок массива при извлечении второго подмассива вдоль оси 0 (по вертикали):

```
a1 = a[1]
a1 Out:
array([[4, 5],
       [6, 7]])
```

Проверка эквивалентности:

```
np.array_equal(a1, a[1,:])
Out: True

np.array_equal(a1, a[1,:,:])
Out: True
# Взять первый элемент подмассива
# array([[0, 2], [4, 6]])
np.array_equal(a[…,0], a[:,:,0])
Out: True
```

Несколько базовых операций

Одна из ключевых возможностей NumPy — выполнение как базовых, так и сложных математических операций с массивами с тем же синтаксисом, что и для обычных числовых переменных. На рисунке 7.2 показано применение перегруженных арифметических операторов к двумерному массиву. Простые математические операции применяются ко всем членам массива. Обратите внимание, что при делении массива на значение с плавающей точкой (2.0), результат автоматически преобразуется к типу `float64`. Возможность работать с массивами так же легко, как и отдельными числами — огромное преимущество NumPy и значительная часть ее выразительных возможностей.

Логические операторы работают так же, как и арифметические. Как мы увидим в следующей главе, это полезный способ создания логических масок (Boolean masks), которую часто использует pandas. Пример:

```
a = np.array([45, 65, 76, 32, 99, 22])
a < 50
Out[69]: array([True, False, False, True, False, True]
               , dtype=bool)
```

```
a=np.arrange([1, 2, 3, 4, 5, 6])
a=a.reshape([2, 3])
```

```
          a + 2          a - 2          a / 2.0
|1, 2, 3|  |2, 4, 6|  |-1, 0, 1|  |0.5, 1., 1.5|
|4, 5 ,6|  |8, 8, 20|  | 2, 3, 4|  |2., 2.5, 3. |
                                    dtype= float64
```

Рис. 7.2. *Несколько базовых арифметических операций с двумерным массивом NumPy*

Пример 7-1 демонстрирует некоторые методы массивов. См. их исчерпывающий перечень в официальной документации NumPy.

Пример 7-1. *Некоторые методы массивов*

```
a = np.arange(8).reshape((2,4))
# array([[0, 1, 2, 3],
#  [4, 5, 6, 7]])
a.min(axis=1)
# array([0, 4])
a.sum(axis=0)
# array([4, 6, 8, 10])
a.mean(axis=1) ❶
# array([1.5, 5.5])
a.std(axis=1) ❷
# array([1.11803399, 1.11803399])
```

❶ Среднее значение по второй оси.

❷ Стандартное отклонение [0, 1, 2, 3],…

NumPy содержит обширный набор встроенных функций для работы с массивами. Часть из них продемонстрирована в примере 7-2. Полный список встроенных математических функций NumPy см. на официальном сайте NumPy.

Пример 7-2. *Некоторые математические функции для массивов NumPy*

```
# Тригонометрические функции
pi = np.pi
a = np.array([pi, pi/2, pi/4, pi/6])

np.degrees(a) # преобразование радианов в градусы
# Вывод: array([180., 90., 45., 30.,])

sin_a = np.sin(a)
# Вывод: array([ 1.22464680e-16,     1.00000000e+00, ❶
#                7.07106781e-01,     5.00000000e-01])
# Округление
np.round(sin_a, 7) # округление до 7 знаков после запятой
# Вывод: array([0., 1., 0.7071068, 0.5])
# Сумма, произведение, разность
a = np.arange(8).reshape((2,4))
# array([[0, 1, 2, 3],
#        [4, 5, 6, 7]])

np.cumsum(a, axis=1) # накопленная сумма вдоль горизонтальной оси
# массив([[0, 1, 3, 6],
#        [4, 9, 15, 22]])

np.cumsum(a) # без аргумента axis массив преобразуется в одномерный
# array([0, 1, 3, 6, 10, 15, 21, 28])
```

❶ Обратите внимание на ошибку округления с плавающей точкой для sin(pi).

Создание функций для работы с массивами

Массив NumPy — основная структура данных для pandas, SciPy, scikit-learn или PyTorch. Следовательно, умение создавать функции для обработки массивов расширяет возможности набора инструментов обработки данных и тулчейна визуализации данных. В сообществе, как правило, найдется готовое решение, но можно получить массу удовольствия от создания собственного, и, помимо прочего, это отличный способ попрактиковаться. Давайте рассмотрим, как использовать массив NumPy для расчета скользящей средней. Скользящая средняя (moving

average) — это последовательность средних значений, в скользящем окне (сдвигающемся интервале) из *n* элементов.

Расчет скользящей средней

В примере 7-3 показан короткий код для расчета скользящей средней одномерного массива NumPy[1]. Как видите, код красивый и лаконичный, но его логика требует пояснений. Попробуем разобраться.

Пример 7-3. *Реализация скользящей средней с помощью NumPy*

```
def moving_average(a, n=3):
    ret = np.cumsum(a, dtype=float)
    ret[n:] = ret[n:] - ret[:-n]
    return ret[n - 1:] / n
```

Функция принимает массив *a* и число *n*, задающее размер скользящего окна.

Сначала с помощью встроенного метода NumPy мы рассчитываем накопленную сумму массива:

```
a = np.arange(6)
# массив([0, 1, 2, 3, 4, 5])
csum = np.cumsum(a)
csum
# Вывод: array([0, 1, 3, 6, 10, 15])
```

Начиная с *n-го* индекса массива накопленной суммы, мы вычитаем значение *csum[i-n]*, так что что теперь *i*-й элемент содержит сумму последних *n* элементов из массива *a*. В примере ниже размер окна 3:

```
# a = array([0, 1, 2, 3, 4, 5])
# csum = array([0, 1, 3, 6, 10, 15])
csum[3:] = csum[3:] - csum[:-3]
# csum = array([0, 1, 3, 6, 9, 12])
```

[1] Метод convolves NumPy упрощает расчет простой скользящей средней, но он менее нагляден. В pandas также есть несколько реализаций поиска скользящей средней.

Сравнивая массив `a` с окончательным массивом `csum`, мы видим, что элемент csum [5] теперь отражает сумму окна [3, 4, 5] массива `a`.

Поскольку вычислять скользящую среднюю имеет смысл, начиная с индекса (n-1), остается только возвратить эти значения, разделенные для получения среднего на размер окна n.

Функция `moving_average` требует некоторого времени для осмысления, но она демонстрирует краткость и выразительность операций с массивами NumPy и их срезами. Такую функцию легко написать и на чистом Python, но она будет сложнее и, что особенно важно, с большими массивами будет работать гораздо медленнее.

Вызовем нашу функцию:

```
a = np.arange(10)
moving_average(a, 4)
# Вывод[98]: array([1.5,  2.5,  3.5,  4.5,  5.5,  6.5,  7.5])
```

Резюме

В этой главе мы рассмотрели основы NumPy, уделив особое внимание структурному блоку NumPy — многомерному массиву (`ndarray`). Навык использования NumPy необходим любому питонисту, работающему с данными. Эта библиотека является ядром стека Python для обработки данных, уже по этой причине надо знать, как работать с ее массивами.

Навыки работы с NumPy упрощают работу с pandas, к тому же для рабочего процесса в pandas можно использовать богатую экосистему NumPy: алгоритмы машинного обучения, научные, инженерные и статистические. Хотя pandas прячет массивы NumPy за контейнерами данных, такими как DataFrame и Series, предназначенными для работы с неоднородными данными, на деле эти контейнеры по большей части ведут себя как массивы NumPy и выполняют необходимое. Когда вы формулируете задачи для pandas, полезно помнить, что в ее основе лежат массивы `ndarray`, — в конечном итоге операции должны быть совместимы с NumPy. Итак, мы познакомились со структурными блоками NumPy, теперь рассмотрим, как pandas расширяет однородный массив в область неоднородных данных, где и происходит большая часть работы по визуализации.

Знакомство с библиотекой pandas

Библиотека pandas — ключевой элемент тулчейна визуализации. Мы будем использовать ее для очистки и для анализа полученного нами набора данных (см. главу 6). В предыдущей главе мы познакомились с NumPy — Python-библиотекой для обработки массивов, на базе которой работает pandas. Прежде чем перейти к применению pandas, ознакомимся в этой главе с ее основными концепциями и рассмотрим, как она взаимодействует с существующими файлами данных и таблицами базы данных. В следующих двух главах мы будем изучать pandas.

Почему pandas оптимальна для визуализации данных

Данные любой визуализации, как браузерной, так и печатной обычно хранятся в табличных форматах: Excel, CSV или HDF5. Конечно, есть визуализации, например графы сетей, для которых табличный формат данных не очень подходит, но таких меньшинство. Pandas специально создана для работы с табличными данными. Ее основной тип данных DataFrame лучше всего воспринимать, как быстродействующую программную таблицу.

Зачем разработали pandas

Впервые секрет раскрыл Уэс МакКинни (Wes McKinney) в 2008 г.: pandas была создана для конкретной цели, как более мощный инструмент анализа данных и моделирования. В этой области Python, в целом прекрасно подходивший для работы с данными, был не силен, особенно по сравнению с языком R.

Библиотека pandas предназначена для работы с неоднородными[1] данными, которые часто встречаются в электронных таблицах, но при этом ей удается действовать быстрее благодаря однородным числовым массивам NumPy, которые

[1] Столбцы в типичной электронной таблице, как правило, имеют различные типы данных, например, числа с плавающей точкой, дата и время, целые числа и др.

используют математики, физики, специалисты по компьютерной графике и так далее. Pandas — первоклассный интерактивный инструмент анализа и обработки данных, особенно в сочетании с Jupyter Notebook и библиотекой для построения графиков Matplotlib (с ее расширением Seaborn). Поскольку pandas входит в экосистему NumPy, то для улучшения моделирования данных она легко может использовать SciPy, statsmodels, scikit-learn и массу других библиотек.

Классификация данных и измерения

В следующих подразделах я рассмотрю основные концепции pandas и уделю особое внимание структуре DataFrame и методам импорта/экспорта данных из распространенных хранилищ: CSV-файлов, SQL-баз данных и других источников. Но сначала рассмотрим, что подразумевается под гетерогенными наборами данных — основным объектом работы pandas и ключевым материалом для визуализаций.

Типичные визуализации — столбчатые диаграммы или линейные графики в статьях и современных веб-дашбордах — обычно отображают результаты измерений: динамику цен на товары, годовое количество осадков, распределение электоральных предпочтений по этническим группам и др. Эти измерения можно условно разделить на количественные (числовые) и качественные (категориальные). Числовые величины разделяют на интервальные шкалы и шкалы отношений, а категориальные — на номинальные и порядковые измерения. Это дает нам четыре обобщенные категории наблюдений.

Возьмем в качестве примера набор твитов, чтобы выделить эти категории. В каждом твите есть различные поля данных:

```
{
   "text": "#Python and #JavaScript sitting in a tree...", ❶
   "id": 2103303030333004303, ❶
   "favorited": true, ❷
   "filter_level":"medium", ❸
   "created_at": "Wed Mar 23 14:07:43 +00002015", ❹
   "retweet_count":23, ❺
   "coordinates":[-97.5, 45.3] ❻
   ...
}
```

❶ Поля `text` и `id` — уникальные идентификаторы. Первое может содержать категориальные данные (например, группу твитов с хэштегом #Python),

а второе — использоваться для формирования категорий (например, множества пользователей, ретвитнувших запись), но сами по себе эти поля непригодны для визуализации.

❷ Поле `favorited` содержит логическую категориальную информацию, разделяющую твиты на два набора. Ее можно считать *номинальной* категорией, поскольку ее можно подсчитать, но нельзя упорядочить.

❸ Поле `filter_level` содержит категориальную информацию порядкового типа. Уровни фильтров располагаются в следующем порядке: низкий→средний→высокий.

❹ Поле `created_at` содержит метку времени — числовое значение на интервальной шкале. Возможно, нам потребуется упорядочить твиты по этой шкале, что pandas делает автоматически, а затем сгруппировать их в более широкие интервалы: по дням или неделям. Опять же, для pandas это тривиально.

❺ Поле `retweet_count` использует числовую шкалу отношений. В отличие от интервальной шкалы, нуль шкалы отношений имеет смысл, в данном случае — отсутствие ретвитов. С другой стороны, метка времени `created_at` может иметь произвольную базовую линию (например, Unix-время или 0 год григорианского календаря), аналогично температурным шкалам, где 0 °C соответствует 273,15 K.

❻ Если есть поле `coordinates`, то оно содержит две шкалы: географической широты и долготы. Хотя это интервальные шкалы, так как говорить о соотношении градусов не имеет смысла.

Итак, небольшое подмножество полей нашего скромного твита содержит неоднородную информацию, охватывающую все общепринятые типы измерений. В то время, как массив NumPy используется для обработки больших объемов однородных (обычно числовых) данных, pandas предназначена для работы с категориальными данными, временными рядами и элементами, отражающими неоднородную природу данных реального мира и поэтому замечательно подходит для работы с визуализациями.

Узнав, с какими типами данных имеет дело pandas, рассмотрим ее структуры данных.

DataFrame

Сессия pandas обычно начинается с загрузки данных в DataFrame. В следующем подразделе мы рассмотрим различные способы загрузки. А сейчас прочитаем

данные из JSON-файла *nobel_winners.json*, передав путь к нему в `read_json`. Метод возвращает DataFrame с данными, полученными при парсинге файла. По соглашению, имена переменных DataFrame начинаются с `df`:

```
import pandas as pd

df = pd.read_json('data/nobel_winners.json')
```

Получив DataFrame, изучим его содержимое. Быстрый способ просмотреть структуру DataFrame (строки и столбцы) — использовать метод `head`, по умолчанию отображающий первые пять элементов. На рисунке 8.1 показан вывод из Jupyter Notebook, с подсвеченными ключевыми элементами DataFrame.

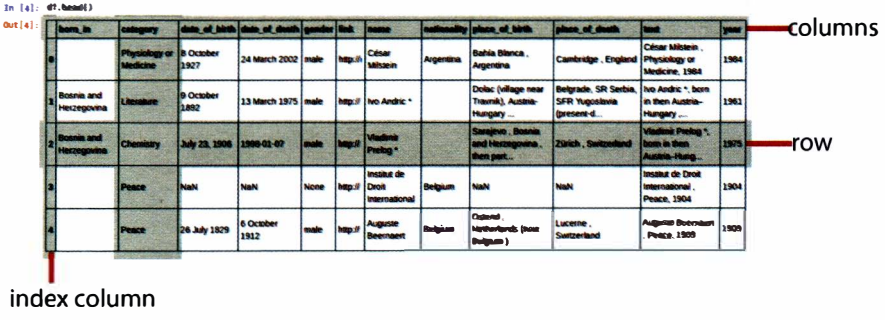


***Рис. 8.1.** Ключевые элементы pandas DataFrame*

Индексы

Столбцы DataFrame индексируются через свойство `columns`, которое является экземпляром `index` объекта pandas. Выберем столбцы на рисунке 8.1:

```
In [0]: df.columns
Out[0]: Index(['born_in', 'category', ...], dtype='object')
```

На начальном этапе строки pandas имеют единственный числовой индекс (при необходимости pandas может обрабатывать несколько индексов), доступ к которому можно получить с помощью свойства `index`. По умолчанию это экономящий память `RangeIndex`.

```
In [1]: df.index
Out[1]: RangeIndex(start=0, stop=1052, step=1)
```

Индексы строк могут быть как целыми числами, так и строками, а также объектами `DatetimeIndex` или `PeriodIndex` для временных данных и так далее. Для упрощения выборок столбец в DataFrame часто устанавливают в качестве индекса с помощью метода `set_index`. В следующем примере мы сначала устанавливаем индекс DataFrame нобелевских лауреатов на столбец name методом `set_index`, а затем выбираем строку по метке индекса (в данном случае `name`) через метод `loc`:

```
In [2] df = df.set_index('name') ❶
In [3] df.loc['Albert Einstein'] ❷
Out[3]:
      born_in category    country date_of_birth    date_of_death \
name
Albert Einstein  Physics  Switzerland   1879-03-14    1955-04-18
Albert Einstein  Physics  Germany       1879-03-14    1955-04-18
[...]

df = df.reset_index() ❸
```

❶ Устанавливает индекс на столбец name.
❷ Теперь можно выбрать строку по метке `name`.
❸ Возвращает индекс в исходное целочисленное значение.

Строки и столбцы

Строки и столбцы DataFrame хранятся в структуре pandas Series, гетерогенном аналоге массива NumPy. По сути, это одномерный массив данных с метками. Данные могут быть любого типа: от целых чисел, строк и чисел с плавающей точкой до объектов и списков Python.

Есть два способа получить строку из DataFrame. Метод `loc`, использующий метку, мы уже рассмотрели. Есть также метод `iloc`, который позволяет использовать позицию. Извлечем строку с номером 2, показанную на рисунке 8.1:

```
In [4] df.iloc[2]
Out[4]:
name                    Vladimir Prelog *
born_in                 Bosnia and Herzegovina
category                Chemistry
country
```

SKLADCHIK.ORG

```
date_of_birth              July 23, 1906
...
year                       1975
Name: 2, dtype: object
```

Можно извлечь ряд из DataFrame, используя точечную нотацию[1] или обычный доступ к массиву по ключевой строке. Мы получим pandas Series со всеми полями столбца и сохраненными индексами DataFrame:

```
In [9] gender_col = df.gender # or df['gender']
In [10] type(gender_col)
Out[10] pandas.core.series.Series
In [11] gender_col.head() # grab the Series' first five items
Out[11]:
0     male #index, object
1     male
2     male
3     None
4     male
Name: gender, dtype: object
```

Выбор групп

Есть различные способы выборки группы (то есть подмножества строк) из DataFrame. Часто требуется отобрать все строки с конкретным значением столбца (например, все строки с категорией Physics). Один из способов — использование метода DataFrame `groupby` для группирования столбца (или списка столбцов), а затем использовать метод `get_group` для получения нужной группы. Применим оба эти метода, чтобы выбрать всех лауреатов-физиков:

```
cat_groups = df.groupby('category')
cat_groups
#Out[-] <pandas.core.groupby.generic.DataFrameGroupBy object …>

cat_groups.groups.keys()
#Out[-]:  dict_keys(['', 'Chemistry', 'Economics', 'Literature',\
```

[1] Только если имя столбца — строка без пробелов.

```
#           'Peace', 'Physics', 'Physiology or Medicine'])
...

In [14] phy_group = cat_groups.get_group('Physics')
In [15] phy_group.head()
Out[15]:
      name          born_in category    country     date_of_birth \
13    François Englert    Physics     Belgium     6 November 1932
19    Niels Bohr          Physics     Denmark     7 October 1885
23    Ben Roy Mottelson   Physics     Denmark     July 9, 1926
24    Aage Bohr           Physics     Denmark     19 June 1922
47    Alfred Kastler      Physics     France      3 May 1902
...
```

Второй способ выбрать подмножество строк — использовать логическую маску для создания нового DataFrame. Логические операторы можно применять ко всем строкам DataFrame практически так же, как это делается для элементов массива NumPy.

```
In [16] df.category == 'Physics'
Out[16]:
0     False
1     False
...
1047  True
...
```

Полученную в результате логическую маску затем можно применить к исходному DataFrame для фильтрации строк:

```
In [17]: df[df.category == 'Physics']
Out[17]:
         name                born_in category       country \
13       Francois Englert    Physics                Belgium
19       Niels Bohr          Physics                Denmark
23       Ben Roy Mottelson   Physics                Denmark
24       Aage Bohr           Physics                Denmark
...
1047     Brian P. Schmidt    Physics                Australia ...
```

В следующих главах мы рассмотрим гораздо больше примеров отбора данных. А пока сосредоточимся на создании DataFrame из существующих данных и сохранении результатов манипуляций.

Создание и сохранение структур DataFrame

Проще всего создать DataFrame на основе словаря Python. Однако этот способ используется редко, поскольку данные обычно загружаются из файлов или баз данных. Тем не менее, он тоже может пригодиться.

По умолчанию столбцы задают по отдельности. В этом примере создадим три строки со столбцами name и category:

```
df = pd.DataFrame({
      'name': ['Albert Einstein', 'Marie Curie',\
      'William Faulkner'],
      'category': ['Physics', 'Chemistry', 'Literature']
      })
```

А теперь применим метод `from_dict`. С помощью его необязательного аргумента `orient` можно явно указать ориентацию данных по строкам, но pandas обычно корректно определяет структуру автоматически:

```
df = pd.DataFrame.from_dict([ ❶
      {'name': 'Albert Einstein', 'category':'Physics'},
      {'name': 'Marie Curie', 'category':'Chemistry'},
      {'name': 'William Faulkner', 'category':'Literature'}
     ])
```

❶ Здесь мы передаем список словарей, где каждый элемент соответствует строке в DataFrame.

Оба показанных метода создают идентичный DataFrame:

```
df.head()
Out:
      name                category
0     Albert Einstein     Physics
1     Marie Curie         Chemistry
2     William Faulkner    Literature
```

Как говорилось выше, создание DataFrame напрямую из Python-коллекций используют редко, зато часто применяют методы pandas для чтения данных из внешних источников.

Набор методов типа `read_[format]/to_[format]` охватывает всевозможные варианты загрузки данных: от CSV и бинарных HDF5 до баз данных SQL. Мы рассмотрим те методы, которые больше всего подходят для работы с визуализацией. См. полный перечень в документации pandas.

Важно отметить, что pandas по умолчанию применяет интеллектуальное преобразование загружаемых данных. Параметры read-методов `convert_axes` (автоматическое приведение осей к корректным типам), `dtype` (автоматическое определение типов) и `convert_dates` по умолчанию установлены в `True`. Пример доступных опций см. в документации pandas (в данном случае для чтения файлов JSON в DataFrame).

Сначала рассмотрим работу с файловыми DataFrame, затем перейдем к взаимодействию с базами данных SQL/NoSQL.

JSON

Загрузка данных из формата JSON в pandas выполняется элементарно:

```
df = pd.read_json('file.json')
```

Существуют различные формы данных, считываемых из файла JSON, которые задают необязательный аргумент `orient`, имеющий одно из следующий значений: `split`, `records`, `index`, `columns`, `values`]. Массив записей (стандартный формат) будет распознан автоматически:

```
[{"name":"Albert Einstein", "category":"Physics", ...},
{"name":"Marie Curie", "category":"Chemistry", ...} ...]
```

Но по умолчанию используется ориентация на столбцы (`columns`), где данные представлены в форме:

```
{"name":{"0":"Albert Einstein","1":"Marie Curie" ...},
"category":{"1","Physics","2":"Chemistry" ...}}
```

Как уже говорилось, для веб-визуализации, особенно с помощью D3, наиболее распространенным форматом передачи табличных данных являются JSON-массивы записей.



Обратите внимание, что для работы с pandas необходимы корректные файлы JSON, поскольку требования метода `read_json` и Python-парсеров JSON в целом довольно строги, а сообщения об ошибках часто недостаточно информативны[1]. Распространенные ошибки — отсутствие двойных кавычек у ключей или использование одинарных кавычек вместо двойных. Последнее особенно характерно для разработчиков, перешедших с языков, где одинарные и двойные кавычки взаимозаменяемы. Это одна из причин, по которой не следует создавать JSON-документы вручную — всегда используйте официальные или проверенные библиотеки.

Есть разные способы хранения DataFrame в JSON, но для задач визуализации оптимальным форматом является массив записей. Это наиболее распространенный формат данных для D3, который я рекомендую использовать при экспорте из pandas[2]. Для сохранения DataFrame в виде записей достаточно указать параметру `orient` метода `to_json` значение 'records':

```
df = pd.read_json('data.json')
# ... Выполнение операций по очистке данных
json = df.to_json('data_cleaned.json', orient='records') ❶
Out:
[{"name":"Albert Einstein", "category":"Physics", ...},
{"name":"Marie Curie", "category":"Chemistry", ...} ...]
```

❶ При сохранении данных в JSON переопределяет их форму по умолчанию на удобные для визуализации записи.

Мы может использовать параметры `date_format` (метка времени *epoch*, *iso* для ISO8601 и др.), `double_precision` (точность чисел) и `default_handler` для обработки объектов, которые парсер pandas не может конвертировать в JSON. См. подробнее в документации pandas.

[1] При возникновении ошибок проверьте подмножество данных в валидаторе JSONLint, чтобы получить детализированную диагностику.

[2] D3 поддерживает различные форматы данных, в том числе иерархические данные (деревья), или графы с узлами и связями. См. пример иерархических данных в JSON.

CSV

Поскольку pandas прекрасно работает с табличной формой представления, при обработке CSV-файлов она справляется практически со всеми возможными данными. Большинство стандартных CSV-файлов загружаются без указания дополнительных параметров:

```
# data.csv:
# name,category
# "Albert Einstein",Physics
# "Marie Curie",Chemistry

df = pd.read_csv('data.csv')
df
Out:
   name                category
0  Albert Einstein     Physics
1  Marie Curie         Chemistry
```

Хотя файлы CSV (Comma-Separated Values) подразумевают разделение запятыми, часто встречаются другие разделители — точка с запятой или вертикальная черта (|). Также для строк, содержащих пробелы или специальные символы могут использоваться специфичные кавычки. В таком случае при чтении файла можно указать любые нестандартные параметры. Воспользуемся удобным модулем Python `StringIO` для эмуляции чтения из файла[1]:

```
from io import StringIO

data = " `Albert Einstein`| Physics \n`Marie Curie`| Chemistry"

df = pd.read_csv(StringIO(data),
   sep='|', ❶
   names=['name', 'category'], ❷
   skipinitialspace=True, quotechar="`")

df
Out:
```

[1] Я рекомендую использовать этот подход, если вы хотите оценить работу парсеров CSV или JSON. Это гораздо удобнее, чем работать с локальными файлами.

```
      name     category
0     Albert   Einstein Physics
1     Marie    Curie Chemistry
```

❶ Поля разделяются вертикальной чертой (|), а не стандартными запятыми.
❷ Здесь мы добавляем отсутствующие заголовки столбцов.

Такая же гибкость доступна при сохранении в CSV-файл. Мы установим кодировку Unicode `UTF-8`:

```
df.to_csv('data.csv', encoding='utf-8')
```

Полный список параметров CSV см. в документации pandas.

Файлы Excel

В pandas для чтения файлов Excel 2003 (*.xls*) используется модуль Python `xlrd`, для Excel 2007+ (*.xlsx*) — модуль `openpyxl`. Последний является опциональной зависимостью, требующей отдельной установки.

```
$ pip install openpyxl
```

Документы Excel содержат именованные листы, каждый из которых можно передать в DataFrame. Прочитать их можно двумя способами. Первый — создать объект `ExcelFile`, а затем выполнить его парсинг.

```
dfs = {}
xls = pd.ExcelFile('data/nobel_winners.xlsx') # загрузить файл Excel
dfs['WinnersSheet1'] = xls.parse('WinnersSheet1', na_values=['NA']) ❶
dfs['WinnersSheet2'] = xls.parse('WinnersSheet2',
      index_col=1, ❷
      na_values=['-'], ❸
      skiprows=3 ❹
      )
```

❶ Выбрать лист по имени и сохранить в словарь.
❷ Задать столбец по позиции для использования в качестве меток строк DataFrame.
❸ Список дополнительных строковых значений, распознаваемых как `NaN`.

❹ Число строк (метаданные), которые нужно проигнорировать перед обработкой.

Альтернативный вариант — использование метода `read_excel`, который упрощает загрузку нескольких таблиц:

```
dfs = pd.read_excel('data/nobel_winners.xlsx',
                    ['WinnersSheet1','WinnersSheet2'],
                    index_col=None, na_values=['NA'])
```

Проверим содержимое второго листа Excel, используя полученный DataFrame:

```
In: dfs['WinnersSheet2'].head()
Out:
   category    nationality              year  name                gender
0  Peace       American                 1906  Theodore Roosevelt  male
1  Literature  South African            1991  Nadine  Gordimer    female
2  Chemistry   Bosnia and Herzegovina   1975 Vladamir Prelog      male
```

Если для чтения каждого листа Excel нужны разные аргументы, единственная причина отказа от использования `read_excel` — необходимость их индивидуальной настройки.

Можно задавать листы по индексу или имени, используя второй параметр `sheetname`, который может быть одной строкой имени или индексом (отсчет с 0) или смешанным списком. По умолчанию `sheetname` имеет значение 0 — возвращается первый лист. В примере 8-1 показано несколько вариантов. При установке `sheetname` в значение `None` возвращается словарь объектов DataFrame с ключами, соответствующими именам листов.

Пример 8-1. *Загрузка листов sheets*

```
# возвратить первый лист
df = pd.read_excel('nobel_winners.xls')

# возвратить лист по имени
df = pd.read_excel('nobel_winners.xls', 'WinnersSheet3')

# первый лист и лист с именем  'WinnersSheet3'
df = pd.read_excel('nobel_winners.xls', [0, 'WinnersSheet3'])
```

```
# все листы выгружаются в словарь, где ключи — имена листов.
dfs = pd.read_excel('nobel_winners.xls', sheetname=None)
```

Параметр `parse_cols` указывает столбец листа для обработки. Если в `parse_cols` передать целочисленное значение, то будут отобраны все столбцы до этого порядкового номера. Передача списка целых чисел в `parse_cols` позволяет выбирать конкретные столбцы:

```
# парсинг до пятого столбца
pd.read_excel('nobel_winners.xls', 'WinnersSheet1', parse_cols=4)

# парсинг второго и четвертого столбца
pd.read_excel('nobel_winners.xls', 'WinnersSheet1', parse_cols=[1, 3])
```

Больше информации о `read_excel` см. в документации pandas.

DataFrame можно сохранить в лист Excel с помощью метода `to_excel`, указав имя файла `'nobel_winners'` и имя листа `'WinnersSheet1'`:

```
df.to_excel('nobel_winners.xlsx', sheet_name='WinnersSheet1')
```

Для Excel есть опции, схожие с `to_csv`, см. о них в документации pandas. Поскольку объекты `Panel` библиотеки pandas и файлы Excel могут хранить несколько DataFrame, метод `Panel to_excel` позволяет записать все содержащиеся в Panel объекты DataFrame в файл Excel.

Для записи нескольких DataFrame в один файл Excel, можно использовать объект `ExcelWriter`:

```
with pd.ExcelWriter('nobel_winners.xlsx') as writer:
    df1.to_excel(writer, sheet_name='WinnersSheet1')
    df2.to_excel(writer, sheet_name='WinnersSheet2')
```

SQL

Для работы с различными СУБД pandas использует модуль Python `SQLAlchemy`, обеспечивающий уровень абстракции баз данных. При использовании `SQLAlchemy` потребуется установить драйвер соответствующей базы данных.

Загрузить базу данных или результаты SQL-запроса проще всего с помощью метода `read_sql`. Из выбранной нами SQLite прочитаем таблицу nobel_winners в DataFrame:

```
import sqlalchemy

engine = sqlalchemy.create_engine(
'sqlite:///data/nobel_winners.db') ❶
df = pd.read_sql('winners', engine) ❷
df
Out:
   предметный указатель  category country  date_of_birth
0  4                     Peace Belgium     1829-07-26
...
   name                          place_of_birth
0  Auguste Beernaert     Ostend, Netherlands (now Belgium)
...
```

❶ Мы используем однофайловую базу данных SQLite. SQLAlchemy может создавать движки для всех популярных баз данных, например *mysql://USER: PASSWORD@localhost/db*.

❷ Чтение содержимого SQL-таблицы `'nobel_winners'` в датафрейм. Функция `read_sql` — удобная обертка для методов `read_sql_table` и `read_sql_query`, которая автоматически выбирает необходимое в зависимости от первого аргумента.

Запись DataFrame в базу данных SQL выполняется достаточно просто. Используя созданное нами подключение, мы можем добавить копию таблицы nobel_winners в базу данных SQLite:

```
# сохранить DataFrame df в SQL-таблицу nobel_winners
df.to_sql('winners_copy', engine, if_exists='replace')
```

В случае возникновения ошибок из-за ограничений на размер пакета параметр `chunksize` позволяет задать количество строк для записи за одну операцию:

```
# за один раз записать 500 строк
df.to_sql('winners_copy', engine, chunksize=500)
```

Сделав вывод о типе данных объектов, pandas пытается сопоставить ваши данные с подходящим типом SQL. При необходимости тип по умолчанию можно переопределить при вызове метода:

SKLADCHIK.ORG

```
from sqlalchemy.types import String
df.to_sql('winners_copy', engine, dtype={'year': String}) ❶
```

❶ Переопределяем тип данных pandas и указываем год как столбец `String`. Подробнее о взаимодействии pandas и SQL см. в документации pandas.

MongoDB

Документноориентированные базы данных NoSQL довольно удобно использовать при создании визуализаций. В случае MongoDB преимуществ еще больше: для своего хранилища данных она поддерживает бинарный JSON, а именно BSON (binary JSON). Поскольку JSON — основной формат данных, передаваемых между браузером с нашей веб-визуализацией и бэкендом, хранение наборов данных в MongoDB становится логичным выбором. К тому же MongoDB хорошо работает в связке с pandas.

Как мы выяснили, датафреймы библиотеки pandas прекрасно конвертируются в формат JSON и обратно, поэтому перенести коллекцию документов Mongo в pandas DataFrame довольно просто:

```
import pandas as pd
from pymongo import MongoClient client = MongoClient() ❶
db = client.nobel_prize ❷
cursor = db.winners.find() ❸
df = pd.DataFrame(list(cursor)) ❹
df ❺
# _
```

❶ Создать клиент MongoDB, используя хост и порт по умолчанию.
❷ Получить базу данных `nobel_prize`.
❸ Найти все документы в коллекции `winners`.
❹ Загрузить все документы из курсора (объекта cursor) в список и использовать для создания DataFrame.
❺ На данный момент коллекция winners пуста — заполним ее данными из DataFrame.

Так же просто можно вставить записи DataFrame в базу данных MongoD B. Чтобы получить базу данных `nobel_prize` и сохранить DataFrame в ее коллекции winners, применим метод `get_mongo_database`, который мы определили в примере 3-5:

```
db = get_mongo_database('nobel_prize')

records = df.to_dict('records') ❶
db[collection].insert_many(records) ❷
```

❶ Преобразуем DataFrame в `dict`, используя аргумент `records` для преобразования строк в отдельные объекты.
❷ Для PyMongo версии 2 нужно использовать метод `insert`.

В pandas нет таких удобных методов для MongoDB, как `to_csv` и `read_csv`, но достаточно будет написать пару вспомогательных функций для преобразования данных из MongoDB в DataFrame и обратно:

```
def mongo_to_dataframe(db_name, collection, query={},\
                       host='localhost', port=27017,\
                       username=None, password=None,\
                       no_id=True):
    """ создать DataFrame из коллекции mongodb """

    db = get_mongo_database(db_name, host, port, username,\
        password)
    cursor = db[collection].find(query)
    df = pd.DataFrame(list(cursor))

    if no_id: ❶
        del df['_id']

    return df

def dataframe_to_mongo(df, db_name, collection,\
                       host='localhost', port=27017,\
                       username=None, password=None):
    """ сохранить DataFrame в коллекцию mongodb """
    db = get_mongo_database(db_name, host, port, username,\
        password)

    records = df.to_dict('records')
    db[collection].insert_many(records)
```

❶ Поле Mongo `_id` будет включено в DataFrame. По умолчанию этот столбец удаляется.

Убедимся, что записи, вставленные из DataFrame в Mongo, успешно сохранены:

```
db = get_mongo_database('nobel_prize')
list(db.winners.find()) ❶
[{'_id': ObjectId('62fcf2fb0e7fe50ac4393912'),
  'id': 1,
  'category': 'Physics',
  'name': 'Albert Einstein',
  'nationality': 'Swiss',
  'year': 1921,
  'gender': 'male'},
 {'_id': ObjectId('62fcf2fb0e7fe50ac4393913'),
  'id': 2,
  'category': 'Physics',
  'name': 'Paul Dirac',
  'nationality': 'British',
  'year': 1933,
  'gender': 'male'},
 {'_id': ObjectId('62fcf2fb0e7fe50ac4393914'),
  'id': 3,
  'category': 'Chemistry',
  'name': 'Marie Curie',
  'nationality': 'Polish',
  'year': 1911,
  'gender': 'female'}]
```

❶ Метод `find` коллекции возвращает курсор, который мы преобразуем в список Python для просмотра содержимого.

Другой способ создать DataFrame — построить его из коллекции Series. Рассмотрим этот способ и заодно более детально познакомимся со структурой данных Series.

Создание DataFrame из Series

Series — строительные блоки для DataFrame pandas. Структуры данных Series можно обрабатывать отдельно с помощью методов, аналогичных методам DataFrame, или комбинировать для формирования DataFrame, как будет показано далее.

Ключевая особенность pandas Series — система индексов. Индексы служат метками для разнотипных данных в строке. При обработке нескольких объектов данных pandas использует индексы для согласования полей.

Series можно создать тремя способами. Первый способ — из списка Python или массива NumPy:

```
s = pd.Series([1, 2, 3, 4]) # Series(np.arange(4))
Out:
0  1 # индекс, значение
1  2
2  3
3  4
dtype: int64
```

Обратите внимание, что для Series автоматически создаются целочисленные индексы. При добавлении строки данных в DataFrame требуется явное указание индексов столбцов через список целых чисел или меток:

```
s = pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])
s
Out:
a  1

c  3
d  4
dtype: int64
```

Примечание: длина массива индексов должна соответствовать длине массива данных. С помощью словаря Python можно определить как данные, так и индексы:

```
s = pd.Series({'a':1, 'b':2, 'c':3})
Out:
```

```
a  1
b  2
c  3
dtype: int64
```

Если мы передадим массив индексов вместе со словарем, то pandas сопоставит индексы с массивом данных. Несовпадающие индексы получат значение NaN, а лишние данные будут удалены. Важное следствие несоответствия количества элементов и индексов — автоматическое приведение типа Series к `float64`:

```
s = pd.Series({'a':1, 'b':2}, index=['a', 'b', 'c'])
Out:
a  1.0
b  2.0
c  NaN dtype: float64

s = pd.Series({'a':1, 'b':2, 'c':3}, index=['a', 'b'])
Out:
a  1
b  2
dtype: int64
```

И наконец, мы можем передать в Series в качестве данных одно скалярное значение, при условии, что также укажем индекс. Затем скалярное значение применяется ко всем индексам:

```
pd.Series(9, {'a', 'b', 'c'})
Out:
a  9
b  9
c  9
dtype: int64
```

Series похожи на массивы NumPy (`ndarray`), поэтому их можно передавать большинству функций NumPy:

```
s = pd.Series([1, 2, 3, 4], ['a', 'b', 'c', 'd'])
np.sqrt(s)
Out:
```

```
a  1.000000
b  1.414214
c  1.732051
d  2.000000
dtype: float64
```

Обратите внимание: операции среза работают также, как со списками Python или массивами `ndarray`, но метки индексов сохраняются:

```
s[1:3]
Out:
b  2
c  3
dtype: int64
```

Series в pandas, в отличие от массивов NumPy, могут содержать данные различного типа. Это полезное свойство показано при сложении двух Series, при этом числа суммируются, а строки объединяются:

```
pd.Series([1, 2.1, 'foo']) + pd.Series([2, 3, 'bar'])
Out:
0       3 # 1 + 2
1     5.1 # 2.1 + 3
2  foobar # корректная конкатенация строк
dtype: object
```

Возможность создавать отдельные Series и управлять ими особенно важна при взаимодействии с экосистемой NumPy, обработке данных из DataFrame или создании визуализаций вне оберток Matplotlib в pandas.

Поскольку Series являются базовыми элементами объектов DataFrame, их можно объединить в DataFrame с помощью метода pandas `concat`:

```
names = pd.Series(['Albert Einstein', 'Marie Curie'],\
name='name') ❶
categories = pd.Series(['Physics', 'Chemistry'],\
name='category')

df = pd.concat([names, categories], axis=1) ❷
df.head()
```

```
Out:
      name                category
0     Albert Einstein     Physics
1     Marie Curie         Chemistry
```

❶ Мы используем две Series — `names` и `categories` — для предоставления данных и имен столбцов (свойство `name`) для DataFrame.

❷ Объединяем две Series с помощью аргумента `axis` со значением `1`, чтобы указать, что Series будут столбцами.

Познакомившись с созданием DataFrame из Series, а также из файлов и баз данных (как обсуждалось ранее), вы получили прочную основу для импорта и экспорта данных в структурах DataFrame.

Резюме

Эта глава заложила основу для двух следующих, посвященных pandas. Мы обсудили основные концепции pandas — индексы, DataFrame и Series. Узнали, почему pandas хорошо подходит для визуализации типов данных, связанных с реальным миром, и о том, что расширяя возможности многомерных массивов NumPy, она позволяет хранить данные различных типов и добавляет мощную систему индексации.

Из следующих глав вы узнаете, как использовать основные структуры данных pandas при очистке и обработке набора данных о лауреатах Нобелевской премии, а также расширите знания об инструментах pandas и научитесь применять их для визуализации данных.

Теперь, зная, как помещать данные в DataFrame и извлекать их, рассмотрим, что pandas может с ними делать. Сначала узнаем, как обеспечить чистоту данных, обнаружив и исправив аномалии: дублирование строк, пропуск полей и повреждение данных.

Очистка данных с помощью pandas

В двух предыдущих главах вы познакомились с библиотекой NumPy (Numeric Python) и расширяющей ее возможности библиотекой pandas. Теперь, вооружившись базовыми знаниями о pandas, мы готовы приступить к этапу очистки данных с помощью нашего тулчейна: постараемся найти и удалить грязные данные в полученном с помощью скрейпинга наборе данных (см. главу 6). Эта глава также углубит ваши знания о pandas, познакомив с новыми методами в процессе работы.

В главе 8 мы рассмотрели основные компоненты pandas: DataFrame — программную таблицу, работающую с множеством различных типов данных, которые встречаются в реальном мире, и с Series — гетерогенным расширением однородного многомерного массива NumPy (`ndarray`). Мы также выяснили, как выполнять чтение и запись для различных хранилищ данных: файлов JSON и CSV, баз данных SQLite и MongD B. Теперь начнем проверять на практике возможности pandas по очистке грязных данных. Я познакомлю вас с ключевыми элементами очистки данных на примере нашего набора данных о лауреатах Нобелевской премии.

Я буду постепенно демонстрировать вам ключевые концепции pandas в рабочей среде. Для начала выясним, почему очистка данных настолько важный этап создания визуализаций.

Чистая правда о грязных данных

Думаю, можно смело утверждать, что большинство начинающих специалистов в области визуализации данных зачастую сильно недооценивают время, которое потребуется на подготовку данных к презентации. На самом деле на получение наборов чистых данных, которые можно преобразовать во впечатляющие визуализации, порой требуется более половины общего времени работы. Сырые данные редко бывают безупречными: в них встречаются ошибки ручного ввода данных, поля, не заполненные из-за невнимательности или ошибок парсинга, и несогласованные форматы даты и времени.

Чтобы создать по-настоящему содержательную задачу, я через скрейпинг взял для этой книги набор данных о лауреатах Нобелевской премии из Википедии — ресурса с ручным редактированием и неформальными стандартами структурирования данных. Так что добытые данные неизбежно будут грязными, ведь людям свойственно ошибаться. Но даже данные из официальных API, например крупных соцсетей, часто содержат отсутствующие или неполные поля, рассогласование вследствие многократных изменений схем данных, преднамеренные ошибки ввода и т. п.

Таким образом, очистка данных — это необходимый, хотя и трудоемкий этап подготовки данных, отнимающий время от творческих аспектов визуализации. Это веская причина научиться делать очистку действительно хорошо и быстро. Успех во многом зависит от правильного выбора набора инструментов для очистки, и библиотека pandas — это то, что нужно. Библиотека отлично справляется со сложным анализом даже объемных[1] наборов данных и может сэкономить массу времени, когда хорошо в ней разберешься. Этому и посвящена текущая глава.

Подведем итог: скрейпинг данных о нобелевских лауреатах из Википедии с использованием библиотеки Scrapy (см. главу 6) привел к созданию массива объектов JSON следующего вида:

```
{
   "category": "Physics",
   "name": "Albert Einstein",
   "gender": "male",
   "place_of_birth": "Ulm, Baden-W\u00fcrttemberg,
      German Empire",
   "date_of_death": "1955-04-18",
...
}
```

Задача этой главы — показать, как преобразовать полученный массив в максимально очищенный источник данных, которые мы будем анализировать с помощью pandas в следующей главе.

Наиболее распространенные виды грязных данных:

— дублирование записей/строк;

— пропущенные значения в полях;

[1] *Объемных* — расплывчатое определение, но здесь возможности pandas ограничены только размером оперативной памяти вашего компьютера, где и находятся DataFrame.

— несогласованные строки;
— повреждение полей;
— смешанные типы данных в одном столбце.

Проанализируем наши данные о лауреатах на наличие таких аномалий.

Сначала нужно загрузить данные JSON в DataFrame, как показано в предыдущей главе (см. «Создание и сохранение структур DataFrame» на стр. 261). Можно напрямую открыть JSON-файл:

```
import pandas as pd

df = pd.read_json(open('data/nobel_winners_dirty.json'))
```

Мы поместили грязные данные в DataFrame, теперь посмотрим, что там есть.

Проверка качества данных

В pandas есть несколько методов и свойств DataFrame, позволяющих быстро получить представление о содержащихся данных. Метод `info` дает общую информацию — структурированную сводку числа записей по столбцам:

```
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1052 entries, 0 to 1051
Data columns (total 12 columns):
 #      Column            Non-Null Count      Dtype
---     -------           --------------      -----
 0       born_in          1052 non-null       object
 1       category         1052 non-null       object
 2       country          1052 non-null       object
 3       date_of_birth    1044 non-null       object
 4       date_of_death    1044 non-null       object
 5       gender           1040 non-null       object
 6       link             1052 non-null       object
 7       name             1052 non-null       object
 8       place_of_birth   1044 non-null       object
 9       place_of_death   1044 non-null       object
```

SKLADCHIK.ORG

```
 10      text              1052 non-null       object
 11      year              1052 non-null       int64
dtypes: int64(1), object(11)
memory usage: 98.8+ KB
```

Мы видим, что в некоторых полях отсутствуют значения. Например, в нашем датафрейме 1052 строки, а столбец gender содержит только 1040 записей. Также обратите внимание на параметр `memory_usage`, показывающий объем оперативной памяти, занятый датафреймами pandas. При увеличении объема данных этот показатель позволяет оценить приближение к аппаратным ограничениям RAM.

Метод `describe` формирует сводную статистику для столбцов DataFrame:

```
df.describe()
Out:
                year
count    1052.000000
mean     1968.729087
std      33.155829
min      1809.000000
25%      1947.000000
50%      1975.000000
75%      1996.000000
max      2014.000000
```

Как мы видим, по умолчанию описываются только столбцы с числовыми данными. Здесь сразу обнаруживается аномалия: минимальный год равен 1809, что невозможно, поскольку первое вручение Нобелевской премии состоялось в 1901 году.

Параметр `include` метода `describe` позволяет указать типы данных (`dtype`) в столбцах, которые нужно проверить. За исключением столбца year, все столбцы в нашем наборе данных имеют тип object — базовый полиморфный `dtype` в pandas, способный хранить числа, строки, временные данные и другие форматы. В примере 9-1 показано, как получить для них статистику.

Пример 9-1. *Описание DataFrame*

```
In [140]: df.describe(include=['object']) ❶
Out[140]:
        born_in category   date_of_birth   date_of_death   gender \
count      1052     1052            1044            1044     1040
unique       40        7             853             563        2
top             Physio..  9 May 1947                       male
freq        910      250               4             362      983
                          link            name \
count                     1052            1052
unique                    893             998
top   http://eg/wiki/...                  Daniel Kahneman
freq                      4               2
        country    place_of_birth     place_of_death \
count      1052            1044               1044
unique       59             735                410
top      United States
freq       350              29                409
...
```

❶ Аргумент `include` — это список (или отдельный элемент) для подведения итогов по `dtype` столбцов.

Из результата в примере 9-1 можно почерпнуть много полезной информации, например, что в наборе данных представлено 59 уникальных национальностей, и что самая большая группа лауреатов — из США (350).

Из 1044 зарегистрированных записей дат рождения только 853 являются уникальными — это интересный момент, который может возникнуть по разным причинам. Варианты могут быть разные: в какие-то дни родилось более одного лауреата, или (помня об очистке данных) есть лауреаты, имена которых повторяются, или некоторые даты неверны, или в них указан только год. Гипотеза о дублировании лауреатов подтверждается тем, что из 1052 имен только 998 уникальные. Хотя существуют двукратные лауреаты, их слишком мало, чтобы объяснить 54 дубликата.

Методы `head` и `tail` класса DataFrame предоставляют простой способ предварительного просмотра данных. По умолчанию они выводят первые/последние пять строк, но это количество можно изменить, передав целое число в качестве первого аргумента. В примере 9-2 показан результат применения `head` к Nobel DataFrame.

Пример 9-2. *Выборка первых пяти строк DataFrame*

```
df.head()
Out:
      born_in                category                    date_of_bi..
0                            Physiology or Medicine     8 October 1..
1  Bosnia and Herzegovina    Literature                 9 October 1..❶
2  Bosnia and Herzegovina    Chemistry                   July 23, 1..
3                            Peace                                ..
4                            Peace                       26 July 1..
   date_of_death    gender ..
0  24 March 2002    male   http://en.wikipedia.org/wiki/C%C3%A..
1  13 March 1975    male   http://en.wikipedia.org/wi..
2  1998-01-07       male   http://en.wikipedia.org/wiki/Vl.. ❷
3  NaN              None   http://en.wikipedia.org/wiki/Institu..
4  6 October 1912   male   http://en.wikipedia.org/wiki/Auguste..
      name                                country \
0     Cesar Milstein                      Argentina
1     Ivo Andric *
2     Vladimir Prelog *
3     Institut de Droit International     Belgium
4      Auguste Beernaert                  Belgium
```

❶ В этих строках есть запись в поле `born_in` и звездочка рядом с именем.

❷ В поле `date_of_death` представлен формат даты, отличный от формата в других строках.

В первых пяти строках данных о лауреатах в примере 9-2 можно заметить пару полезных вещей. Во-первых, имена в строках 1 и 2 помечены звездочкой и имеют запись в поле `born_in` ❶. Во-вторых, в строке 2 в поле `date_of_death` используется альтернативный формат даты, а в `date_of_birth` ❷ присутствуют смешанные форматы «месяц-день» и «день-месяц». Такого рода несоответствие — вечная проблема данных, отредактированных человеком, особенно дат и времени. Далее мы покажем, как устранить эти проблемы с помощью библиотеки pandas.

В примере 9-1 подсчитано, что в поле `born_in` 1052 объекта, что формально исключает пустые поля, однако `head` показывает, что контент есть только в строках 1 и 2. Это говорит о том, что пропущенные значения представлены пустой строкой или пробелом, которые pandas интерпретирует как валидные данные. Заменим эти значения на `NaN`, чтобы исключить их из подсчета. Но сначала будет маленькое введение в механизмы отбора данных с помощью pandas.

Индексы и отбор данных с помощью pandas

Прежде чем приступить к очистке данных, я сделаю краткий обзор базовых методов отбора данных с использованием pandas на примере набора данных о нобелевских лауреатах.

Обычно индексы столбцов определяются источником данных (файлом, SQL-таблицей и т. д.), однако, как было показано в предыдущей главе, их можно явно задать при создании DataFrame, передав в аргументе `names` список имен столбцов. Индекс столбцов доступен как свойство DataFrame:

```
# Столбцы набора данных о лауреатах Нобелевской премии
df.columns
Out: Index(['born_in', 'category', 'date_of_birth',
...
            'place_of_death', 'text', 'year'], dtype='object')
```

По умолчанию pandas задает для строк целочисленные индексы, начиная от нуля, но можно их переопределить, передав список значений в параметр `index` при создании DataFrame или напрямую изменив свойство `index`. Чаще всего в качестве индекса используют один или несколько[1] столбцов DataFrame. Для этого предназначен метод `set_index`. Если вы хотите вернуться к стандартному целочисленному индексу, используйте метод `reset_index`, как показано в примере 9-3.

Пример 9-3. *Настройка индекса DataFrame*

```
# устанавливает поле name в качестве индекса
df = df.set_index('name') ❶
df.head(2)
Out:
                born_in                    category \
name ❷
Cesar Milstein                             Physiology or Medicine
Ivo Andric *    Bosnia and Herzegovina     Literature
...
df.reset_index(inplace=True) ❸
df.head(2)
```

[1] pandas поддерживает многоуровневую индексацию с помощью объекта MultiIndex. Он предоставляет весьма эффективный способ работы с многомерными данными. Подробнее см. в документации pandas.

```
Out:
   name              born_in                 category \
0  Cesar Milstein                            Physiology or Medicine ❹
1  Ivo Andric *      Bosnia and Herzegovina  Literature
```

❶ Устанавливаем столбец name в качестве индекса датафрейма. Возвращаем результат в `df`.

❷ Теперь строки индексируются по `name`.

❸ Индекс возвращается к целочисленному значению. Обратите внимание: изменение выполняется «на месте», в исходном DataFrame.

❹ Теперь индексы целочисленные, в соответствии с позицией строки.



Изменить pandas DataFrame или Series можно двумя способами: модификация данных на месте (in-place) или в созданной копии. Нет гарантии, что первый способ быстрее, к тому же, если вызывается цепочка методов, то требуется, чтобы операция возвращала измененный объект. Обычно я использую форму `df = df.foo(...)`, но большинство модифицирующих методов поддерживают параметр inplace с синтаксисом `df.foo(..., inplace=True)`.

Рассмотрев систему индексации строк и столбцов, начнем отбирать срезы DataFrame.

Столбец DataFrame можно получить с помощью точечной нотации (если в имени нет пробелов и специальных символов) или с помощью квадратных скобок. Рассмотрим столбец `born_in`:

```
bi_col = df.born_in # или bi = df['born_in']
bi_col
Out:
0
1     Bosnia and Herzegovina
2     Bosnia and Herzegovina
3
...
1051
Name: born_in, Length: 1052, dtype: object
type(bi_col)
Out: pandas.core.series.Series
```

Обратите внимание, что при выборе столбца возвращается объект pandas Series с сохраненной индексацией DataFrame.

DataFrame и Series используют одни и те же методы доступа к строкам/элементам. `iloc` ищет по целочисленному номеру позиций строк и столбцов, а `loc` — по метке. Применим `iloc` для извлечения первой строки нашего DataFrame:

```
# доступ к первой строке
df.iloc[0]
Out:
name                        Cesar Milstein
born_in
category                    Physiology or Medicine
...
# установка индекса в 'name' и доступ по метке имени
df.set_index('name', inplace=True)
df.loc['Albert Einstein']
Out:
           born_in category         country ...
name
Albert Einstein     Physics          Switzerland ...
Albert Einstein     Physics          Germany ...
...
```

Отбор нескольких строк

При отборе нескольких строк из DataFrame можно использовать стандартные срезы для списков Python:

```
# отбор первых 10 строк
df[0:10]
Out:
        born_in              category                  date_of_b..
0                            Physiology or Medicine    8 October ..
1  Bosnia and Herzegovina    Literature                9 October ..
...
9                            Peace                     1910-0..
# выбрать последние четыре строки
df[-4:]
```

```
Out:
      born_in        category                 date_of_birth date_..
1048                 Peace                    November 1, 1878 May..
1049          Physiology or Medicine          1887-04-10 19..
1050                 Chemistry                1906-9-6 1..
1051                 Peace                    November 26, 1931 ..
```

Стандартный способ выбора нескольких строк, удовлетворяющих условию (например, значение столбца `value` больше `x`) — создать логическую маску и использовать ее для фильтрации. Давайте найдем всех лауреатов, удостоенных Нобелевской премии после 2000 г. Сначала создадим маску, выполнив логическое выражение для каждой строки:

```
mask = df.year > 2000 ❶
mask
Out:
0     False
1     False
...
13    True
...
1047  True
1048  False
...
Name: year, Length: 1052, dtype: bool
```

❶ Получаем `True` для строк, где значение поля `year` больше 2000.

Результирующая логическая маска имеет тот же индекс, что и DataFramee, и используется для отбора всех строк со значением `True`:

```
mask = df.year > 2000
winners_since_2000 = df[mask] ❶
winners_since_2000.count()
Out:
...
year 202       # число лауреатов после 2000 г.
dtype: int64
winners_since_2000.head()
```

```
Out:
...
                                                    text  year
13                    Francois Englert ,  Physics,  2013  2013
32       Christopher A. Pissarides ,     Economics,  2010  2010
66                           Kofi Annan ,    Peace,  2001  2001
87                 Riccardo Giacconi *,  Physics,  2002  2002
88       Mario Capecchi *,   Physiology or Medicine,  2007  2007
```

❶ Возвращает DataFrame, содержащий только строки, для которых логическая маска `mask` имеет значение `True`.

Фильтрация с помощью логических масок — мощный способ выбора произвольных подмножеств данных. Я бы рекомендовал задать целевые условия и попрактиковаться в построении корректных логических выражений. Как правило, мы можем обойтись без создания промежуточной маски:

```
winners_since_2000 = df[df.year > 2000]
```

Теперь, освоив отбор строк с помощью срезов и логической маски, перейдем к изменению DataFrame, параллельно очищая его от «грязных» данных.

Очистка данных

Зная, как получить доступ к данным, давайте рассмотрим способы их улучшения. Начнем с анализа полей `born_in`, которые отображались пустыми в примере 9-2. Анализ столбца `born_in` методом count не выявляет пропущенных строк, которые обычно обозначаются как отсутствующие значения или `NaN` (Not a Number):

```
In [0]: df.born_in.describe() Out[0]:
count    1052
unique   40
top
freq     910
Name: born_in, dtype: object
```

Обнаружение смешанных типов данных

Обратите внимание, что pandas хранит все строковые данные с использованием типа object. Беглый осмотр показывает, что в столбце смешаны пустые строки и строки с названиями стран. Мы можем быстро проверить, действительно ли тип у всех элементов столбца str: применив метод `apply` с функцией Python `type` ко всем элементам, а затем преобразовав полученный список типов в множество:

```
In [1]: set(df.born_in.apply(type)) Out[1]: {str}
```

Мы убедились, что все элементы `born_in` — строки с типом `str`. Теперь заменим все пустые строки на значения `NaN`.

Замена строк

Необходимо заменить пустые строковые значения на `NaN`, чтобы исключить их из расчетов[1]. Метод `replace` в pandas оптимально подходит для данной задачи и может применяться как ко всему DataFrame, так и к отдельным Series:

```
import numpy as np
bi_col.replace('', np.nan, inplace=True)
bi_col

Out:
0                          NaN ❶
1       Bosnia and Herzegovina
2       Bosnia and Herzegovina
3                          NaN
...
bi_col.count()
Out: 142 ❷
```

❶ Пустая строка `''` заменяется на NumPy `NaN`.

❷ В отличие от пустых строк поля с `NaN` не учитываются при подсчете.

[1] По умолчанию pandas для обозначения отсутствующих значений использует `NaN` — константу из NumPy с типом float.

После замены пустых строк на значения NaN мы получим точное количество записей (142) для поля born_in.

Теперь заменим все пустые строки в DataFrame на значения NaN:

```
df.replace('', np.nan, inplace=True)
```

Библиотека pandas предоставляет расширенные возможности замены строковых данных и других объектов в столбцах, например с помощью регулярных выражений, которые можно применить к Series или к столбцам DataFrame. Рассмотрим пример, используя помеченные звездочкой имена из DataFrame нобелевских лауреатов.

В примере 9-2 показаны имена некоторых лауреатов, которые помечены звездочкой, указывающей на использование страны рождения вместо страны гражданства на момент присуждения премии:

```
df.head()
Out:
...
          name                                country \
0         Cesar Milstein                      Argentina
1         Ivo Andric *
2         Vladimir Prelog *
3         Institut de Droit International     Belgium
4         Auguste Beernaert                   Belgium
```

Наша задача — очистить имена, удалив звездочки (*) и лишние пробельные символы.

Объект Series.str в pandas предоставляет набор строковых методов для обработки элементов массива. С помощью одного из них подсчитаем, сколько имен помечены звездочкой:

```
df[df.name.str.contains(r'\*')]['name'] ❶
Out:
1         Ivo Andric *
2         Vladimir Prelog *
...
1041      John Warcup Cornforth *
1046      Elizabeth H. Blackburn *
Name: name, Length: 142, dtype: object ❷
```

SKLADCHIK.ORG

❶ Используем метод `str.contains` для столбца `name`. Примечание: необходимо экранировать звездочку (`'\*'`), как в синтаксисе регулярных выражений. Логическая маска применяется к DataFrame с данными лауреатов и выводятся полученные имена.

❷ 142 строки из 1052 содержат символ звездочки `*`.

Чтобы очистить имена, заменим звездочки пустой строкой и удалим все пробельные символы (whitespace):

```
df.name = df.name.str.replace('*', '', regex=True) ❶
# удалить whitespace из имен
df.name = df.name.str.strip()
```

❶ Удаляет все звездочки в полях name и возвращает результат в DataFrame. Обратите внимание, что необходимо установить флаг `regex` в значение `True`.

Быстрая проверка подтверждает корректность имен после очистки:

```
df[df.name.str.contains('\*')]
Out:
Empty DataFrame
```

В pandas Series есть много методов для обработки строк, позволяющих выполнять поиск и модифицировать строки в столбцах. Полный их список см. в этой API документации.

Удаление строк

Следует отметить, что 142 записи о лауреатах с полями `born_in` являются дубликатами — они представлены на биографической странице Википедии как страной рождения, так и страной гражданства на момент присуждения премии. Хотя эти сведения могут служить основой интересной визуализации[1], для нашей задачи необходимо уникальное представление каждого присуждения, поэтому нам нужно удалить из DataFrame дубли.

Создадим новый DataFrame, используя только строки, в которых поле `born_in` имеет значение `NaN`. Казалось бы, можно использовать условное выражение

[1] Один из возможных вариантов визуализации — диаграмма миграции нобелевских лауреатов из стран рождения.

сравнения `born_in` с `NaN`, но по спецификации[1] любое логическое сравнение с `NaN` возвращает `False`:

```
np.nan == np.nan
Out: False
```

В связи с этим pandas предоставляет специальный метод `isnull` для проверки отсутствия значения в полях:

```
df = df[df.born_in.isnull()] ❶
df.count()
Out:
born_in 0 # все записи теперь пустые
category 910
...
dtype: int64
```

❶ Метод `isnull` возвращает логическую маску, где `True` соответствует строкам с пустым полем `born_in`.

Поскольку столбец `born_in` больше не требуется, пока что удалим его[2]:

```
df = df.drop('born_in', axis=1) ❶
```

❶ В качестве первого аргумента `drop` принимает метку или индекс (или список из них), а в качестве второго — `axis`, который указывает, что это индекс строки (0 или по умолчанию) или столбца (`1`).

Поиск дубликатов

Быстрый поиск в интернете показывает, что по состоянию на 2015 год Нобелевскую премию получили 889 человек и организаций. У нас осталось 910 строк, значит еще есть дубли или аномалии.

[1] См. IEEE 754.

[2] Как вы увидите в следующей главе, поля born_in содержат интересную информацию о миграции лауреатов Нобелевской премии. В конце главы мы рассмотрим, как добавить эту информацию в очищенный набор данных.

Метод `duplicated` в pandas возвращает логическую маску, отмечающую повторяющиеся строки. Он делает сравнение по имени столбца или по списку имен столбцов. Получим список всех дубликатов имен:

```
dupes_by_name = df[df.duplicated('name')] ❶
dupes_by_name.count()
Out:
...
year              46
dtype: int64
```

❶ Метод `duplicated` возвращает массив логических значений, где `True` отмечает строки с повторяющимся значением поля `name` после первого вхождения.

Хотя некоторые люди становились лауреатами Нобелевской премии дважды, их количество не достигает 46, что указывает на дублирование около 40 записей. Учитывая, что страница Википедии, которую мы скрейпили, содержит список лауреатов по странам, наиболее вероятно, что причина дублирования — «присвоение» одних и тех же лауреатов несколькими странами.

Рассмотрим некоторые способы поиска дубликатов по полю name в нашем DataFrame. Пусть не все они эффективные, но это хорошая возможность продемонстрировать несколько функций pandas.

По умолчанию метод `duplicated` помечает значением `True` все дубликаты, кроме первого вхождения, но если выставлен параметр `keep='last'`, то значение False устанавливается для последнего вхождения, а для всех остальных — `True`. Объединив эти два вызова с помощью логического *ИЛИ* (|), мы получим полный список дубликатов:

```
all_dupes = df[df.duplicated('name')\
              | df.duplicated('name', keep='last')]
all_dupes.count()
Out:
...
year              92
dtype: int64
```

Все дубликаты также можно получить с помощью метода `isin`, проверяя совпадают ли имена строк DataFrame с именами из списка дубликатов:

```
all_dupes = df[df.name.isin(dupes_by_name.name)] ❶
all_dupes.count()
Out:
...
year             92
dtype: int64
```

❶ `dupes_by_name.name` — это Series столбца, содержащая все дублирующиеся имена.

Для поиска дубликатов также подходит мощный метод `groupby` из библиотеки pandas, который группирует строки DataFrame по столбцу или списку столбцов. Он возвращает список пар «ключ-значение», где значение столбца (или комбинация столбцов) является ключом, а список строк — значениями:

```
for name, rows in df.groupby('name'): ❶
    print('name: %s, number of rows: %d'%(name, len(rows)))

name: A. Michael Spence, number of rows: 1
name: Aage Bohr, number of rows: 1
name: Aaron Ciechanover, number of rows: 1
...
```

❶ Метод `groupby` возвращает итератор кортежей (имя группы, группа).

Чтобы извлечь все дублирующиеся строки, достаточно проверить длину списка строк, возвращаемого ключом. Любой список длиной больше единицы содержит дубликаты имен. Используем метод `concat` из библиотеки pandas, который принимает список списков строк и формирует DataFrame со всеми строками-дубликатами. Конструктор списков Python используется для фильтрации групп, содержащих более одной строки:

```
pd.concat([g for _, g in df.groupby('name')\ ❶
                        if len(g) > 1])['name']

Out:
121    Aaron Klug
131    Aaron Klug
615    Albert Einstein
```

```
844   Albert Einstein
...
489   Yoichiro Nambu
773   Yoichiro Nambu
Name: name, Length: 92, dtype: object
```

❶ Создает список Python, отфильтровав группы строк `name`, где количество строк превышает единицу.



Различными путями к одной цели

В крупных библиотеках, таких как pandas, обычно существует несколько способов добиться одного и того же результата. Для небольших наборов данных (например, о нобелевских лауреатах) подойдет любой метод, но при работе с объемными данными могут возникнуть серьезные проблемы с производительностью. То, что pandas реализует выбранный вами способ, еще не значит, что он эффективный. Учитывая сложные внутренние механизмы обработки данных в библиотеке, важно проявить гибкость при выборе подходов и отслеживать потенциально неэффективные подходы.

Сортировка данных

Итак, после применения `all_dupes` у нас есть DataFrame со всеми строками с дублированными именами. Продемонстрируем на нем метод `sort` из библиотеки pandas.

Библиотека pandas предоставляет метод `sort_values` для классов DataFrame и Series, поддерживающий сортировку по нескольким столбцам:

```
df2 = pd.DataFrame(\
      {'name':['zak', 'alice', 'bob', 'mike', 'bob', 'bob'],\
       'score':[4, 3, 5, 2, 3, 7]})
df2.sort_values(['name', 'score'],\ ❶
        ascending=[1,0]) ❷
Out:
   name     score
1  alice    3
5  bob      7
2  bob      5
```

```
4  bob      3
3  mike     2
0  zak      4
```

❶ Сначала сортирует DataFrame по столбцу name, затем по столбцу score внутри полученных подгрупп. В более ранних версиях pandas использовался метод `sort`, который сейчас устарел и не рекомендуется к применению.

❷ Сортирует имена в алфавитном порядке (по возрастанию), а значения score — в порядке убывания.

Давайте отсортируем DataFrame `all_dupes` по столбцу name и выведем столбцы name, country и year:

```
In [306]: all_dupes.sort_values('name')\
   [['name', 'country', 'year']]
Out[306]:
      name                    country          year
121   Aaron Klug              South Africa     1982
131   Aaron Klug              United Kingdom   1982
844   Albert Einstein         Germany          1921
615   Albert Einstein         Switzerland      1921
...
910   Marie Curie             France           1903
919   Marie Curie             France           1911
706   Marie Skłodowska-Curie  Poland           1903
709   Marie Skłodowska-Curie  Poland           1911
...
650   Ragnar Granit           Sweden           1967
960   Ragnar Granit           Finland          1809
...
396   Sidney Altman           United States    1990
995   Sidney Altman           Canada           1989
...
[92 rows x 3 columns]
```

Данный вывод подтверждает, что, как и ожидалось, некоторые лауреаты указаны дважды для одного года, но с разными странами. Кроме того, обнаружены другие аномалии. Мария Кюри дважды получала Нобелевскую премию, и оба раза она включена в этот список как с французским, так и с польским

гражданством[1]. Наиболее корректным решением будет разделить заслуги между Польшей и Францией, и остановиться на двойной фамилии Склодовская-Кюри. В строке 960 также обнаружен аномальный год — 1809. Сидней Олтмен (Sidney Altman) дублируется, причем в одной из записей указан ошибочный 1990 год.

Удаление дубликатов

Приступим к удалению выявленных дубликатов и начнем создавать компактную функцию очистки.

Представления и копии

При работе с pandas важно четко понимать, изменяете ли вы само представление (view) DataFrame или Series или их копию (copy). Кажется естественным изменить поле `country` в строке (Marie Curie) приведенным ниже способом, но это вызывает предупреждение, которое может ввести в заблуждение:

```
df['country'][709] = 'France'
— c:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a
 DataFrame

See the caveats in the documentation:
   http://pandas.pydata.org/pandas-docs/stable/...
```

Меняем страну в 709 строке с Poland на France.

Предупреждение кажется еще непонятнее, когда обнаруживается, что все работает как ожидалось:

```
df['country'][709]
Out: 'France'
```

Разработчики pandas не рекомендуют использовать *цепочки операций*, так как они могут привести к непреднамеренному изменению данных в копии вместо исходного представления (view)[2].

[1] Хотя Франция стала второй родиной для Кюри, она сохранила польское гражданство. Первый открытый ею радиоактивный изотоп она назвала полонием в честь родной страны.

[2] Некоторые пользователи расценивают подобные предупреждения как излишнюю осторожность. См. обсуждение на Stack Overflow https://stackoverflow.com/questions/20625582/how-to-deal-with-settingwithcopywarning-in-pandas.

Эти предупреждения направлены на соблюдение лучших практик, таких как использование методов `loc` (по метке) и `iloc` (по номеру позиции):

```
df.loc[709, 'country'] = 'France'
```



Изменять строки по числовому индексу допустимо, если ваш набор данных остается неизменным и вы не планируете повторный запуск скриптов очистки. Однако в случаях, подобных данным о нобелевских лауреатах, где требуется применение скриптов очистки к обновленным наборам данных, рекомендуется использовать уникальные идентификаторы (например, строку с именем Marie Curie и 1911 годом, а не с индексом 919).

Более надежный способ изменить страну в конкретной строке — использовать для поиска строки не ее индекс, а неизменяемые значения столбцов. В таком случае, даже если индекс изменится, скрипт очистки будет работать. Следовательно, чтобы изменить страну, указанную при получении премии Марией Кюри в 1911 году, на Францию, можно применить логическую маску с методом `loc` для выбора строки и установить значение France в столбце country. Обратите внимание, что для польского символа ł указан Unicode:

```
df.loc[(df.name == 'Marie Sk\u0142odowska-Curie') &\
       (df.year == 1911), 'country'] = 'France'
```

После изменения страны для Марии Кюри следует удалить или исключить некоторые строки из DataFrame на основе значений столбцов Это можно сделать двумя способами: 1. Использовать метод DataFrame `drop`, который принимает список меток индекса. 2. Создать новый DataFrame с применением логической маски для фильтрации строк, требующих удаления. В случае `drop` мы можем использовать аргумент `inplace`, чтобы вносить изменения в существующий DataFrame.

В коде ниже мы удаляем дубликат строки Сиднея Олтмана, создавая DataFrame с одной требуемой строкой (помните, индексные метки сохраняются) и передаем этот индекс методу `drop`, а также изменяем DataFrame «на месте»:

```
df.drop(df[(df.name == 'Sidney Altman') &\
    (df.year == 1990)].index,
       inplace=True)
```

Другой способ удалить строку — использовать ту же логическую маску с логическим *отрицанием* (~), чтобы создать новый DataFrame со всеми строками, кроме заданной:

```
df = df[~((df.name == 'Sidney Altman') & (df.year == 1990))]
```

Добавим это изменение и все текущие модификации в метод `clean_data`:

```
def clean_data(df):
    df = df.replace('', np.nan)
    df = df[df.born_in.isnull()]
    df = df.drop('born_in', axis=1)
    df.drop(df[df.year == 1809].index, inplace=True)
    df = df[~(df.name == 'Marie Curie')]
    df.loc[(df.name == 'Marie Sk\u0142odowska-Curie') &\
          (df.year == 1911), 'country'] = 'France'
    df = df[~((df.name == 'Sidney Altman') &\
       (df.year == 1990))]
    return df
```

Теперь остались только валидные дубликаты (записи о дважды лауреатах Нобелевской премии) и записи с двумя странами. Для нашей визуализации требуется учитывать каждое присуждение премии только один раз, поэтому требуется удалить половину записей с двумя странами. Проще всего применить метод `duplicated`, но первоначальная сортировка по алфавиту стран приводит к систематическому смещению в пользу стран с первыми буквами алфавита. При отсутствии экспертной оценки наиболее справедливым подходом является случайный выбор одной записи из пары с последующим удалением. Для этого есть несколько способов, но самый простой — рандомизировать порядок строк перед использованием метода pandas `drop_duplicates`, который удаляет все последующие дубликаты (по умолчанию) или предыдущие (при `take_last=True`).

В модуле `random` библиотеки NumPy имеется ряд полезных методов, среди которых для рандомизации индекса строки нам идеально подходит `permutation`. Этот метод принимает массив значений (или индексы) и выполняет их перемешивание (shuffle). Затем применим метод `reindex` для переиндексации

DataFrame в соответствии с перетасованными индексами. Важно: удаляются строки с совпадением как имени так и года, что сохранит корректные случаи дважды лауреатов с разными годами награждения:

```
df = df.reindex(np.random.permutation(df.index))  ❶
df = df.drop_duplicates(['name', 'year'])         ❷
df = df.sort_index() df.count()                   ❸
Out:
...
year              865
dtype: int64
```

❶ Создание версии перетасованного `df.index` и переиндексация `df` с ее помощью.

❷ Удаление всех дубликатов с одинаковым именем и годом.

❸ Возвращение индексов в исходное целочисленное значение.

Если обработка данных выполнена корректно, должны остаться только валидные дубликаты — дважды лауреаты Нобелевской премии. Давайте выведем список оставшихся дубликатов для проверки:

```
In : df[df.duplicated('name') |
        df.duplicated('name', keep='last')]\ ❶
        sort_values(by='name')\
        [['name', 'country', 'year', 'category']]
Out:
     name                    country         year  category
548  Frederick Sanger        United Kingdom  1958  Chemistry
580  Frederick Sanger        United Kingdom  1980  Chemistry
292  John Bardeen            United States   1956  Physics
326  John Bardeen            United States   1972  Physics
285  Linus C. Pauling        United States   1954  Chemistry
309  Linus C. Pauling        United States   1962  Peace
706  Marie Skłodowska-Curie  Poland          1903  Physics
709  Marie Skłodowska-Curie  France          1911  Chemistry
```

❶ Мы скомбинировали методы duplicated с параметрами first и last, чтобы получить все дублирующиеся записи. Если у вас более старая версия pandas, возможно потребуется использовать аргумент `take_last=True`.

SKLADCHIK.ORG

Все правильно, поиск в интернете подтверждает, что двукратных лауреатов было четверо.

Предположим, что мы отыскали все нежелательные дубликаты[1], и теперь можем перейти к другим видам некорректных данных.

Решение проблемы недостающих полей

Проверим, как обстоят дела с пустыми полями (*null*), подсчитав количество строк в столбцах нашего DataFrame:

```
df.count()
Out:
category         864 ❶
country          865
date_of_birth    857
date_of_death    566
gender           857 ❷
link             865
name             865
place_of_birth   831
place_of_death   524
text             865
year             865
dtype: int64
```

❶ Отсутствует одно поле в столбце category.
❷ Нет восьми полей в столбце gender.

По-видимому, отсутствует поле `category`, что указывает на ошибку при вводе данных. Еще при сборе данных о нобелевских лауреатах мы сверяли номинации с валидным списком (см. пример 6-3). Похоже, одна из записей не прошла эту проверку. Для идентификации проблемы извлечем строку с пустым полем category и отобразим ее столбцы name и text:

```
df[df.category.isnull()][['name', 'text']] Out:
        name                                          текст
922     Alexis Carrel Alexis Carrel ,  Medicine,  1912
```

[1] На этом этапе очистки вряд ли удастся отловить всех «нарушителей», хотя это зависит от набора данных.

Мы сохранили ссылку на исходный текст о лауреатах и видим, что Алексис Каррель (Alexis Carrel) назван лауреатом Нобелевской премии по медицине, тогда как корректное название номинации — `Physiology or Medicine` («Физиология и медицина»). Исправим ошибку:

```
...
df.loc[df.name == 'Alexis Carrel', 'category'] =\
   'Physiology or Medicine'
```

Также обнаружены пропуски данных о гендерной принадлежности восьми лауреатов. Выведем их:

```
df[df.gender.isnull()]['name']
Out:
3        Institut de Droit International
156      Friends Service Council
267      American Friends Service Committee (The Quakers)
574      Amnesty International
650      Ragnar Granit
947      Médecins Sans Frontières
1000     Pugwash Conferences on Science and World Affairs
1033     International Atomic Energy Agency
Name: name, dtype: object
```

За исключением Рагнара Гранита (Ragnar Granit), все остальные записи имеют пустые поля гендерных данных (учреждения). Поскольку наша визуализация фокусируется на индивидуальных лауреатах, мы удалим учреждения и укажем гендерную принадлежность для Рагнара Гранита[1]:

```
...
def clean_data(df):
...
df.loc[df.name == 'Ragnar Granit', 'gender'] = 'male'
df = df[df.gender.notnull()] # удалить записи без пола
```

Посмотрим, к чему привели эти изменения, выполнив еще один подсчет для DataFrame:

[1] Хотя гендерная принадлежность Рагнара Гранита не указана в персональны данных, в его биографии в Википедии указано male.

```
df.count() Out:
category          858
date_of_birth     857 # пропущено поле
...
year              858
dtype: int64
```

Оставшиеся после удаления всех учреждений данные лауреатов должны содержать хотя бы дату рождения. Найдем недостающую запись и исправим ее:

```
df[df.date_of_birth.isnull()]['name']
Out:
782   Hiroshi Amano
Name: name, dtype: object
```

Возможно из-за того, что Хироси Амано (Hiroshi Amano) стал лауреатом относительно недавно (2014 г.), его дата рождения была недоступна для скрейпинга. Находим в интернете дату рождения Амано и добавляем ее в DataFrame вручную:

```
...
   df.loc[df.name == 'Hiroshi Amano', 'date_of_birth'] =\
   '11 September 1960'
```

Итак, у нас теперь 858 лауреатов. Для итоговой оценки полноты данных выполним подсчет:

```
df.count()
Out:
category          858
country           858
date_of_birth     858
date_of_death     566
gender            858
link              858
name              858
place_of_birth    831
place_of_death    524
text              858
year              858
dtype: int64
```

Ключевые поля `category`, `date_of_birth`, `gender`, `country` и `year` заполнены. Остальные статистические показатели демонстрируют достаточную репрезентативность. В совокупности очищенные данные обеспечивают надежную основу для создания комплексной визуализации.

А теперь заключительный аккорд: сделаем поля времени более удобными для использования.

Работа с датами и временем

На данный момент поля `date_of_birth` и `date_of_death` представляют собой строки. Как мы видели, отсутствие формализованных правил редактирования Википедии привело к появлению там множества форм записи дат и времени. Уже 10 первых записей нашего исходного DataFrame демонстрируют разнообразнейшие варианты:

```
df[['name', 'date_of_birth']]
Out[14]:
      name                          date_of_birth
4     Auguste Beernaert             26 July 1829
                                       ...
8       Corneille Heymans           28 March 1892
...     ...                         ...
1047  Brian P. Schmidt              February 24, 1967
1048  Carlos Saavedra Lamas         November 1, 1878
1049  Bernardo Houssay              1887-04-10
1050  Luis Federico Leloir          1906-9-6
1051  Adolfo Perez Esquivel         November 26, 1931
[858 rows x 2 columns]
```

Чтобы сравнивать поля дат (например, вычесть год присуждения премии из даты рождения для получения возраста лауреата), необходимо привести их к формату, допускающему такие операции. Неудивительно, что pandas хорошо справляется с анализом неструктурированных временных данных, автоматически конвертируя их в `datetime64` — объект библиотеки NumPy с расширенным набором методов и операторов.

Для преобразования столбца времени в `datetime64` мы используем метод pandas `to_datetime`:

```
pd.to_datetime(df.date_of_birth, errors='raise') ❶
Out:
4        1829-07-26
5        1862-08-29
         ...
1050     1906-09-06
1051     1931-11-26
Name: date_of_birth, Length: 858, dtype: datetime64[ns]
```

❶ Параметр `errors` по умолчанию имеет значение `ignore`, но нам нужно, чтобы отображалось исключение.

По умолчанию `to_datetime` игнорирует ошибки, но нам требуется знать, удалось ли pandas проанализировать `date_of_birth`, чтобы при необходимости исправить дату вручную. К счастью, преобразование прошло без ошибок.

Прежде, чем двигаться дальше, исправим столбец `date_of_birth`:

```
In: df.date_of_birth = pd.to_datetime(df.date_of_birth, errors='coerce')
```

Выполнение `to_datetime` для поля `date_of_birth` вызовет ошибку `ValueError`, причем сообщение об ошибке не указывает на конкретную запись, вызвавшую проблему:

```
In [143]: pd.to_datetime(df.date_of_death, errors='raise')
---------------------------------------------------------------
ValueError       Traceback (most recent call last)
...
301      if arg is None:

ValueError: month must be in 1..12
```

Наивный способ поиска некорректных дат — итерирование по строкам данных с перехватом и выводом ошибок. В pandas для этого удобно использовать `iterrows`, который предоставляет итератор по строкам. В сочетании с блоком Python `try-except` он позволит найти проблемные поля даты:

```
for i, row in df.iterrows():
   try:
      pd.to_datetime(row.date_of_death, errors='raise') ❶
```

```
        except:
            print(f"{row.date_of_death.ljust(30)}({row['name']}, {i})") ❷
```

❶ Запуск `to_datetime` для отдельной строки и перехват любой ошибки.

❷ Для удобства чтения выравниваем текст столбца date_of_death по левому краю (ширина 30). Строки pandas можно маскировать по имени столбца, поэтому мы используем доступ к строковому ключу с помощью `['name']`.

Вывод проблемных строк:

```
1968-23-07 (Henry Hallett Dale, 150)
May 30, 2011 (aged 89)    (Rosalyn Yalow, 349)
living                    (David Trimble, 581)
Diederik Korteweg         (Johannes Diderik van der Waals, 746)
living                    (Shirin Ebadi, 809)
living                    (Rigoberta Menchú, 833)
1 February 1976, age 74   (Werner Karl Heisenberg, 858)
```

Этот перечень наглядно демонстрирует, какие ошибки данных возникают при коллективном редактировании без четких правил.

Хотя выбранный способ работает, прежде, чем перебирать строки DataFrame, следует поискать способ получше, использующий обработку многострочных массивов — главного секрета эффективности pandas.

Лучший способ найти некорректные даты основан на использовании параметра `coerce` метода `to_datetime` библиотеки pandas. При значении `True` этот параметр преобразует все ошибочные даты в `NaT` (Not a Time) — временной аналог `NaN`. Затем можно создать логическую маску для строк с `NaT` в результирующем DataFrame, что визуализировано на рисунке 9.1:

```
with_death_dates = df[df.date_of_death.notnull()] ❶
bad_dates = pd.isnull(pd.to_datetime(\
            with_death_dates.date_of_death, errors='coerce')) ❷
with_death_dates[bad_dates][['category', 'date_of_death','name']]
```

❶ Получаем все строки с непустыми полями даты.

❷ Создаем логическую маску для некорректных дат в `with_death_dates`, проверяя на наличие `NaT` после приведения неудачных преобразований к `NaT`. Для более старых версий pandas может понадобиться использование `coerce=True`.

	category	date_of_death	name
150	Physiology or Medicine	1968-23-07	Henry Hallett Dale
349	Physiology or Medicine	May 30, 2011 (aged 89)	Rosalyn Yalow
581	Peace	living	David Trimble
746	Physics	Diederik Korteweg	Johannes Diderik van der Waals
809	Peace	living	Shirin Ebadi
833	Peace	living	Rigoberta Menchú
858	Physics	1 February 1976, age 74	Werner Karl Heisenberg

Рис. 9.1. *Поля даты, не поддающиеся синтаксическому анализу*

В зависимости от требований к точности, такие поля можно исправить вручную или преобразовать в `NaT` — временной эквивалент `NaN` в NumPy. Наличие более 500 корректных дат смерти позволяет провести временной анализ, поэтому повторно применим `to_datetime` с обработкой ошибок:

```
df.date_of_death = pd.to_datetime(df.date_of_death,\
errors='coerce')
```

Теперь, когда поля времени приведены в пригодный для обработки формат, добавим поле возраста лауреата на момент получения Нобелевской премии. Чтобы получить значение года для новых дат, нужно с помощью метода `DatetimeIndex` сообщить pandas, что она имеет дело со столбцом даты. Примечание: эта грубая оценка возраста может иметь погрешность до одного года. Для целей визуализации данных в следующей главе такой точности достаточно:

```
df['award_age'] = df.year - pd.DatetimeIndex(df.date_of_birth)\
.year ❶
```

❶ Преобразует столбец в объект `DatetimeIndex` — многомерный массив (`ndarray`) данных типа `datetime64` — и использует свойство `year`.

Используем новое поле `award_age` для определения самых молодых лауреатов Нобелевской премии:

```
# используйте +sort+ для старых версий pandas
df.sort_values('award_age').iloc[:10]\
        [['name', 'award_age', 'category', 'year']]
```

```
Out:
      name                award_age  category         year
725   Malala Yousafzai         17.0  Peace            2014 ❶
525   William Lawrence Bragg   25.0  Physics          1915
626   Georges J. F. Kohler     30.0  Phys...Medicine  1976
294   Tsung-Dao Lee            31.0  Physics          1957
858   Werner Karl Heisenberg   31.0  Physics          1932
247   Carl Anderson            31.0  Physics          1936
146   Paul Dirac               31.0  Physics          1933
877   Rudolf Mossbauer         32.0  Physics          1961
226   Tawakkol Karman          32.0  Peace            2011
804   Mairead Corrigan         32.0  Peace            1976
```

❶ См. статью о Малале Юсуфзай (Malala Yousafzai), если хотите узнать подробнее о ее борьбе за доступность образования для женщин.

Теперь, когда поля дат приведены в машиночитаемый формат, рассмотрим полную реализацию функции `clean_data`, обобщающей усилия этой главы по очистке данных.

Полная функция для очистки данных

Маловероятно, что в отредактированных вручную данных, таких как полученный нами набор данных из Википедии, все ошибки будут обнаружены с первого раза. Поэтому будьте готовы к тому, что на этапе исследования данных мы обнаружим несколько новых. Тем не менее наш набор данных о лауреатах Нобелевской премии уже можно использовать. Будем считать, что данные достаточно чистые, и задача главы выполнена. В примере 9-4 показаны все шаги, которые мы предприняли для достижения этого уровня очистки.

Пример 9-4. *Полная функция очистки набора данных о нобелевских лауреатах*

```
def clean_data(df):
df = df.replace('', np.nan)
df_born_in = df[df.born_in.notnull()] ❶
df = df[df.born_in.isnull()]
df = df.drop('born_in', axis=1)
df.drop(df[df.year == 1809].index, inplace=True)
```

```
    df = df[~(df.name == 'Marie Curie')]
    df.loc[(df.name == 'Marie Sk\u0142odowska-Curie') &\
           (df.year == 1911), 'country'] = 'France'
    df = df[~((df.name == 'Sidney Altman') & (df.year == 1990))]
    df = df.reindex(np.random.permutation(df.index))
    df = df.drop_duplicates(['name', 'year']) ❷
    df = df.sort_index()
    df.loc[df.name == 'Alexis Carrel', 'category'] =\
          'Physiology or Medicine'
    df.loc[df.name == 'Ragnar Granit', 'gender'] = 'male'
    df = df[df.gender.notnull()] # удалить учреждения-лауреаты
    df.loc[df.name == 'Hiroshi Amano', 'date_of_birth'] =\
          '11 September 1960'
    df.date_of_birth = pd.to_datetime(df.date_of_birth) ❸
    df.date_of_death = pd.to_datetime(df.date_of_death,\
    errors='coerce')
    df['award_age'] = df.year - pd.DatetimeIndex(df.date_of_birth)\
    .year
    return df, df_born_in ❹
```

❶ Создает DataFrame, содержащий строки с непустыми полями `born_in`.
❷ Удаляет дубликаты после рандомизации порядка строк в DataFrame.
❸ Приводит значения из столбцов с датами к типу `datetime64`.
❹ Возвращает DataFrame с удаленным полем `born_in`. Эти данные пригодятся для интересной визуализации в следующей главе.

Добавление столбца born_in

При очистке DataFrame лауреатов мы удалили столбец `born_in` (см. «Удаление строк» на стр. 289). Как будет показано в следующей главе, данный столбец содержит ценные сведения, которые можно сопоставить со страной награждения для анализа биографических особенностей лауреатов. Функция `clean_data` возвращает данные столбца `born_in` в виде отдельного DataFrame. Рассмотрим, как добавить эти данные в очищенный DataFrame. Сначала загружаем исходные необработанные данные и применяем к ним функцию очистки:

```
df = pd.read_json(open('data/nobel_winners_dirty.json'))
df_clean, df_born_in = clean_data(df)
```

Очищаем поле name в DataFrame df_born_in, удалив звездочки и пробельные символы, а затем удаляем строки-дубликаты по имени. Наконец, установим индекс DataFrame на столбец name:

```
# очистка столбца name: '* Aaron Klug' -> 'Aaron Klug'
df_born_in.name = dfbi.name.str.replace('*', '', regex=False) ❶
df_born_in.name = dfbi.name.str.strip()
df_born_in.drop_duplicates(subset=['name'], inplace=True)
df_born_in.set_index('name', inplace=True)
```

Мы получили DataFrame df_born_in, из которого можно запросить данные по имени лауреата:

```
In: df_born_in['Eugene Wigner']
Out:
born_in                                Hungary
category                               Physics
...
year                                   1963
Name: Eugene Wigner, dtype: object
```

Напишем функцию Python для извлечения поля born_in по имени лауреата из df_born_in нашего датафрейма. Функция возвращает значение поля при наличии соответствующей записи, в противном случае — значение np.nan:

```
def get_born_in(name):
    try:
        born_in = df_born_in.loc[name]['born_in']
        # Выведем строки для проверки корректности данных
        print('name: %s, born in: %s'%(name, born_in))
    except:
        born_in = np.nan
    return born_in
```

Добавим в основной датафрейм столбец born_in, применив функцию get_born_in к полю name каждой строки:

```
In: df_wbi = df_clean.copy()
In: df_wbi['born_in'] = df_wbi['name'].apply(get_born_in)
```

SKLADCHIK.ORG

```
Out:
...
name: Christian de Duve, born in: United Kingdom
name: Ilya Prigogine, born in: Russia
...
name: Niels Kaj Jerne, born in: United Kingdom
name: Albert Schweitzer, born in: Germany
...
```

Убедимся, что мы успешно добавили столбец `born_in` в DataFrame:

```
In: df_wbi.info() Out:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 858 entries, 4 to 1051 Data columns (total 13 columns):
#     Column           Non-Null Count   Dtype
--    ------           --------------   ------
0     category         858 non-null     object
...
12    born_in          102 non-null     object
dtypes: datetime64[ns](2), int64(2),   object(9)
memory usage: 93.8+ KB
```

Обратите внимание: если бы имена лауреатов Нобелевской премии не дублировались, мы могли бы создать столбец `born_in`, просто установив *имя столбца* в качестве индекса `df` и `df_born_in` и создав столбец напрямую:

```
# этот код не будет работать при наличии дублирующихся имен
# в индексе:
In: df_wbi['born_in'] = df_born_in.born_in
Out:
...
ValueError: cannot reindex from a duplicate axis
```

Применение метода `apply` для больших наборов данных может быть неэффективным с точки зрения производительности, однако он обеспечивает гибкий механизм формирования новых столбцов на основе существующих.

Слияние нескольких DataFrame

На данном этапе мы можем создать объединенную базу чистых данных о лауреатах и набора их изображений и биографий, которые добыли, используя скрейпинг текста и изображений с помощью конвейера. При этом мы демонстрируем возможности pandas для объединения двух или более датафреймов. В следующем фрагменте кода показано, как объединить `df_clean` и набора биографических данных:

```
# Считаем биографические данные Scrapy в DataFrame
df_winners_bios = pd.read_json(\
open('data/scrapy_nwinners_minibio.json'))

df_clean_bios = pd.merge(df_wbi, df_winners_bios,\
how='outer', on='link') ❶
```

❶ Метод `merge` библиотеки pandas принимает два DataFrame и объединяет их на основе общего имени/имен столбца (в данном случае, `link`). Аргумент `how` определяет, какие ключи включаются в итоговую таблицу, и работает аналогично операторам JOIN в SQL. В данном случае `outer` задает полное внешнее соединение `FULL_OUTER_JOIN`.

Объединение двух DataFrame приводит к избыточности данных — в результирующем наборе строк становится больше, чем 858 исходных записей о лауреатах:

```
df_clean_bios.count()
Out:
award_age      1023
category       1023
...
bio_image      978
mini_bio       1086
```

Эти избыточные записи можно легко удалить, предварительно отфильтровав строки с отсутствующим полем `name`, а затем устранив дубликаты по комбинации полей `link` и `year` с помощью метода `drop_duplicates`:

```
df_clean_bios = df_clean_bios[~df_clean_bios.name.isnull()]\
.drop_duplicates(subset=['link', 'year'])
```

Быстрый подсчет показывает, что текущий набор данных содержит корректное количество записей — 770 из них имеют изображения, и во всех, кроме одного, присутствует краткая биография (`mini_bio`):

```
df_clean_bios.count()
award_age 858
category 858
...
born_in 102
bio_image 770
mini_bio 857
dtype: int64
```

Очищая наш набор данных, проверим, у кого из лауреатов нет поля `mini_bio`:

```
df_clean_bios[df_clean_bios.mini_bio.isnull()]
Out:
...
      link                                              name \
229   http://en.wikipedia.org/wiki/L%C3%AA_%C3…    Lê Đức Thọ
...
```

Причина заключается в ошибке кодировки Unicode при формировании ссылки на статью Ле Дык Тхо (Lê Đức Thọ), вьетнамского лауреата Нобелевской премии мира. Исправим ее вручную.

DataFrame `df_clean_bios` включает массив URL-адресов изображений, извлеченных из Википедии. Эти данные не будут использоваться в текущем анализе и требуют сериализации в JSON для сохранения в SQL-базу. Минимизируем избыточность, удалив столбец `images_url`:

```
df_clean_bios.drop('image_urls', axis=1, inplace=True)
```

После завершения очистки и оптимизации экспортируем набор данных в пару удобных форматов.

Сохранение очищенных наборов данных

Получив наборы данных для нашего предстоящего исследования с помощью pandas, сохраним их в двух форматах, массово применяющихся при визуализации данных: SQL и JSON.

Прежде всего, с помощью метода pandas `to_json` сохраним очищенный DataFrame с полем `born_in` и объединенными биографиями в виде файла JSON:

```
df_clean_bios.to_json('data/nobel_winners_cleaned.json',\
              orient='records', date_format='iso') ❶
```

❶ Устанавливаем параметр `orient` в значение `records` для сохранения массива объектов-строк и явно указываем кодировку дат как `'iso'`.

Сохраним копию очищенного DataFrame в базе данных SQLite `nobel_prize` в локальном каталоге *data*. Воспользуемся ей в главе 13, чтобы продемонстрировать REST web API на основе Flask. Три строки кода Python и метод DataFrame `to_sql` позволяют лаконично решить задачу:

```
import sqlalchemy

engine = sqlalchemy.create_engine(\
     'sqlite:///data/nobel_winners_clean.db')
df_clean_bios.to_sql('winners', engine, if_exists='replace')
```

Считаем содержимое из базы данных обратно в DataFrame, чтобы убедиться, что создали ее успешно:

```
df_read_sql = pd.read_sql('winners', engine)
df_read_sql.info()
Out:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 858 entries, 0 to 857
Data columns (total 16 columns):
#     Column           Non-Null Count   Dtype
--    --------         --------------   -----
0     index            858 non-null     int64
1     category         858 non-null     object
2     country          858 non-null     object
```

```
3     date_of_birth    858 non-null     datetime64[ns]
[...]
14       bio_image     770 non-null     object
15       mini_bio      857 non-null     object
dtypes: datetime64[ns](2), float64(2), int64(1), object(11)
memory usage: 107.4+ KB
```

Поместив чистые данные в базу данных, в следующей главе приступим к анализу данных.

Резюме

В этой главе вы научились очистке достаточно сложного набора данных, получив информацию, более пригодную для анализа и дальнейшей работы. В процессе работы были представлены новые методы и приемы работы с pandas, расширяющие наше первое знакомство из предыдущей главы до знания основ.

В следующей главе мы используем новый очищенный набор данных для анализа характеристик нобелевских лауреатов: стран, гендерного распределения, возраста и возможных корреляций.

Визуализация данных с помощью Matplotlib

При визуализации один из лучших способов вникнуть в суть данных — отобразить их интерактивно, используя полный спектр диаграмм и графиков, разработанных для упорядочения и анализа наборов данных. Традиционно результаты этапа исследований представлялись в виде статичных изображений, однако сейчас полученные результаты отображаются в виде интерактивных веб-диаграмм, таких как популярные визуализации D3 (подобную мы создадим в разделе V).

Python-библиотека Matplotlib и расширения (например, библиотека Seaborn, предназначенная для статистической визуализации) формируют развитую и гибко настраиваемую экосистему для построения графиков. Интерактивное использование графиков Matplotlib с помощью IPython (версии Qt и Jupyter Notebook) — эффективный и интуитивно понятный метод выявления ценных закономерностей в данных. В этой главе мы познакомимся с Matplotlib и одним из его замечательных расширений — Seaborn.

Pyplot и объектно-ориентированная библиотека Matplotlib

Matplotlib порой сбивает с толку, особенно, если судить по случайно выбранным из интернета примерам. Дело в том, что для создания графиков используются два основных способа, разных по сути, но при этом достаточно похожих, чтобы их можно было перепутать и получить в результате досадные ошибки. При первом способе используется глобальный конечный автомат для прямого взаимодействия с модулем `pyplot` библиотеки Matplotlib. Второй способ — объектно-ориентированный подход с более привычными понятиями классов для программного воплощения фигур и осей. Различие между способами я проясню в следующих подразделах, но, как правило, при работе с отдельными графиками в интерактивном режиме использование глобального состояния `pyplot` ускоряет процесс. В остальных случаях имеет смысл явно определять фигуры и оси и использовать объектно-ориентированный подход.

Запуск интерактивной сессии

Для интерактивной визуализации мы будем использовать Jupyter Notebook. Начнем сессию:

```
$ jupyter notebook
```

Во время сеанса IPython можно использовать для активации интерактивного режима специальные функции — магические команды Matplotlib. Для создания окна построения графиков `%matplotlib` по умолчанию будет использовать GUI-бэкенд, но вы можете указать бэкенд явно. Следующая команда должна работать как со стандартным IPython, так и с QtConsole[1]:

```
%matplotlib [qt | osx | wx ...]
```

Для отображения графиков непосредственно в блокноте Jupyter или Qt Console используйте директиву `inline`. Обратите внимание, что inline-графики нельзя изменять после создания, в отличие от открытых в отдельном окне Matplotlib:

```
%matplotlib inline
```

Независимо от того, используете ли вы Matplotlib интерактивно или в программах Python, команды импорта будет похожи:

```
import numpy as np import pandas as pd
import matplotlib.pyplot as plt
```



Вы можете встретить много примеров кода Matplotlib с использованием модупя `pylab`, который осуществляет массовый импорт `matplotlib.pyplot` (для построения графиков) и NumPy в одно пространство имен. Сейчас `pylab` считается устаревшим, но даже в противном случае я рекомендовал бы избегать данного пространства имен, отдавая предпочтение явному импорту и совмещению `pyplot` и `numpy`.

[1] Если при попытке запустить сессию GUI возникают ошибки, попробуйте изменить настройки бэкенда (например, если при использовании macOS `%matplotlib qt` не работает, попробуйте `%matplotlib osx`).

Хотя использование NumPy и pandas не является обязательным, Matplotlib специально разработана для интеграции с этими библиотеками и эффективно работает с массивами NumPy как и с pandas Series.

Возможность создания встроенных графиков — это ключ для удобной работы с Matplotlib. В IPython это реализуется с помощью «магической»[1] команды:

```
In [0]: %matplotlib inline
```

Теперь ваши Matplotlib-графики будут встроены прямо в вывод IPython. Это сработает как для Notebook, так и для Qt версий. В Jupyter Notebook графики отобразятся в активной ячейке.



Изменение графиков

В режиме inline после запуска/перезапуска ячейки Jupyter Notebook или всего ввода потребуется заполнение всего с самого начала. То есть нельзя изменить график из предыдущей ячейки или ввода, используя метод `gcf` (Get Current Figure) — необходимо повторить все команды построения графика с изменениями в новой ячейке/вводе.

Создание интерактивных графиков с помощью глобального состояния pyplot

Модуль `pyplot` предоставляет глобальное состояние, которым можно управлять интерактивно[2]. Этот подход предназначен для интерактивного анализа данных и оптимален при создании простых графиков, часто содержащих единственную фигуру. Есть много примеров применения удобного `pyplot`, но для сложных сценариев предпочтительнее объектно-ориентированный Matplotlib API (который мы рассмотрим далее). Прежде чем продемонстрировать использование глобального графика, давайте отобразим несколько случайных данных с помощью метода `period_range` из библиотеки pandas:

```
from datetime import datetime

x = pd.period_range(datetime.now(), periods=200, freq='d') ❶
```

[1] IPython включает множество подобных функций, расширяющих функционал стандартного интерпретатора Python полезными дополнениями. См. их перечень на сайте IPython.

[2] Библиотека создавалась как альтернатива MATLAB.

```
x = x.to_timestamp().to_pydatetime() ❷
y = np.random.randn(200, 3).cumsum(0) ❸
```

❶ Создаем индекс pandas `datetime` с элементами 200 дней (`d`), начиная с текущего времени (`datetime.now()`).
❷ Конвертируем индекс `datetime` в нативные Python-объекты `datetime`.
❸ Создаем три случайных массива из 200 элементов, просуммированных вдоль оси 0.

Теперь у нас есть ось Y с 200 временных интервалов и три случайных массива для дополнительных значений x. Для построения линейных диаграмм массивы передаются методу `plot` как отдельные аргументы:

```
plt.plot(x, y)
```

Получилась маловыразительная диаграмма, показанная на рисунке 10.1. Обратите внимание, как естественно Matplotlib работает с многомерным массивом NumPy.

Библиотека создавалась как альтернатива MATLAB.

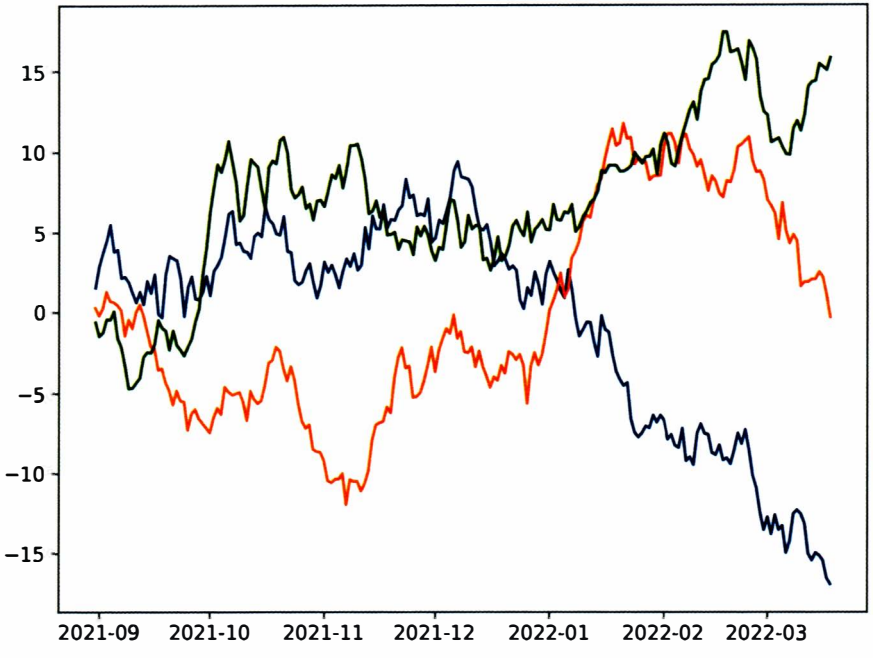


Рис. 10.1. *Линейный график с настройками по умолчанию*

Хотя, по общему мнению, настройки библиотеки Matplotlib по умолчанию неидеальны, она предоставляет богатые возможности для индивидуальных настроек. Существует экосистема библиотек, которые обогащают графики Matplotlib улучшенными настройками по умолчанию, привлекательными палитрами и другими функциями. Рассмотрим примеры такой кастомизации, используя стандартную Matplotlib для настройки полученного нами графика.

Конфигурирование Matplotlib

Matplotlib предоставляет широкий набор настроек, которые можно указать в файле `matplotlibrc` или динамически, через переменную `rcParams`, которая представляет собой словарь. Изменим полученные по умолчанию толщину и цвет линий графиков:

```
import matplotlib as mpl
mpl.rcParams['lines.linewidth'] = 2
mpl.rcParams['lines.color'] = 'r' # красный
```

Пример файла `matplotlibrc` можно найти на основном сайте.

Кроме переменной `rcParams`, можно использовать метод `gcf`, чтобы получить текущую активную фигуру и изменять ее напрямую.

Рассмотрим небольшой пример настройки: изменим размер текущей фигуры.

Настройка размера фигуры

Если график с настройками по умолчанию нечитабельный или соотношение ширины и высоты неоптимальное, следует изменить его размер. По умолчанию Matplotlib задает размеры графиков в дюймах. Это имеет смысл, если учесть множество бэкендов (часто работающих с векторной графикой), с помощью которых Matplotlib может хранить данные. Рассмотрим два способа использования `pyplot` для указания размера фигуры 8 × 4 дюйма с использованием `rcParams` и `gcf`:

```
# Два способа задать размер фигуры 8 на 4 дюйма
plt.rcParams['figure.figsize'] = (8,4)
plt.gcf().set_size_inches(8, 4)
```

SKLADCHIK.ORG

Пункты, а не пиксели

Для измерения фигур в Matplotlib используются не пиксели, а пункты. Эта единица измерения принята в печатных изданиях высокого качества, а в Matplotlib применяется для создания изображений издательского качества.

По умолчанию пункт приблизительно равен 1/72 дюйма, но Matplotlib позволяет настраивать количество точек на дюйм (dpi) для любых генерируемых фигур. Чем больше их число, тем выше качество изображения. Для встроенных фигур, отображаемых интерактивно во время сессий IPython, разрешение обычно определяется бэкендом, используемым для построения графиков (например: Qt, WxAgg, tkinter). Подробное описание бэкендов см. в документации Matplotlib.

Метки и легенды

Рисунок 10.1 позволяет, помимо прочего, рассказать о значении линий с помощью легенды. В Matplotlib предоставляет удобное окно легенды для описания элементов диаграммы. Как и у многого в Matplotlib, у легенд широкие возможности настройки. Чтобы промаркировать три линии, нужен небольшой трюк, так как метод `plot` принимает только один аргумент, который применяет ко всем сгенерированным линиям. Однако команда `plot` возвращает все созданные объекты `Line2D`, у которых есть метод `legend` для установки уникальных обозначений линий.

Поскольку этот график может быть представлен в черно-белом варианте (если вы читаете печатную версию этой книги), чтобы различать линии, требуется альтернативный способ помимо цвета по умолчанию. Оптимальный подход в Matplotlib — последовательное создание линий с указанием координат *x*, *y* и стиля[1]: сплошная (-), пунктирная (--), штрихпунктирная (-.). Обратите внимание на использование индексации столбцов NumPy (см. рисунок 7.1):

```
#plots = plt.plot(x, y)
plots = plt.plot(x, y[:,0], '-', x, y[:,1], '--', x, y[:,2], '-.')
plots
Out:
[<matplotlib.lines.Line2D at 0x9b31a90>,
 <matplotlib.lines.Line2D at 0x9b4da90>,
 <matplotlib.lines.Line2D at 0x9b4dcd0>]
```

[1] Подробнее о стилях линий Matplotlib см. в документации.

С помощью метода `legend` можно задать метки легенды, предложить ее расположение и настроить другие параметры:

```
plt.legend(plots, ('foo', 'bar', 'baz'), ❶
           loc='best', ❷
           framealpha=0.5, ❸
           prop={'size':'small', 'family':'monospace'}) ❹
```

❶ Задаем метки для трех графиков.
❷ Параметр `loc='best'` автоматически выбирает позицию легенды, чтобы она не перекрывала линии.
❸ Задаем прозрачность фона легенды.
❹ Настраиваем свойства шрифта легенды[1].

Заголовки и метки осей

Добавить название и метку для осей очень просто:

```
plt.title('Random trends')
plt.xlabel('Date')
plt.ylabel('Cum. sum')
```

С помощью метода `figtext` можно добавить произвольный текст[2]:

```
plt.figtext(0.995, 0.01, ❶
            '© Acme designs 2022',
            ha='right', va='bottom') ❷
```

❶ Располагаем текст пропорционально размеру фигуры.
❷ Выравниваем по горизонтали (`ha`) и по вертикали (`va`).

Полный код показан в примере 10-1, а полученная диаграмма — на рисунке 10.2.

[1] Подробнее см. документацию.

[2] Подробнее см. сайт Matplotlib.

Пример 10-1. *Настройка линейной диаграммы*

```
plots = plt.plot(x, y[:,0], '-', x, y[:,1], '--', x, y[:,2], '-.')
plt.legend(plots, ('foo', 'bar', 'baz'), loc='best',
                  framealpha=0.25,
                  prop={'size':'small', 'family':'monospace'})
plt.gcf().set_size_inches(8, 4)
plt.title('Random trends')
plt.xlabel('Date')
plt.ylabel('Cum. sum')
plt.grid(True) ❶
plt.figtext(0.995, 0.01, '© Acme Designs 2021',
ha='right', va='bottom')
plt.tight_layout() ❷
```

❶ Добавим точечную сетку, отмечающую деления осей.

❷ Применим метод `tight_layout`, чтобы все элементы графика находились в пределах границы фигуры. В противном случае метки или легенды могут быть обрезаны.

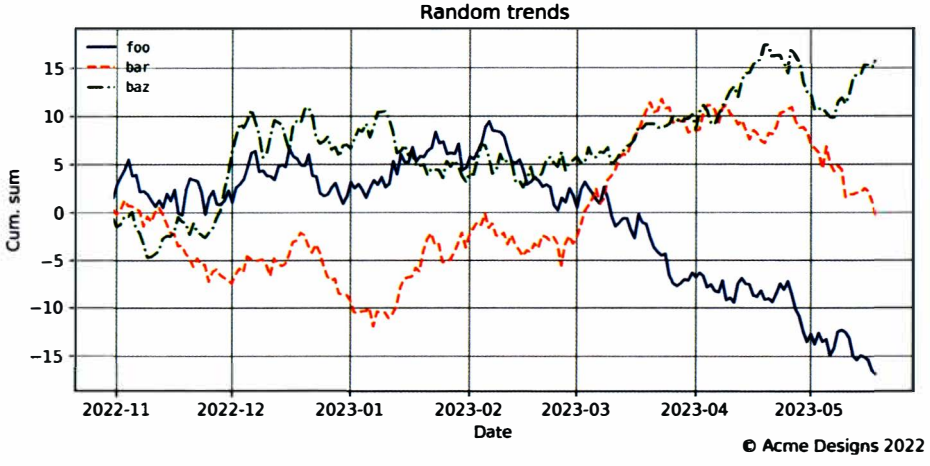


Рис. 10.2. *Настройка линейной диаграммы*

Мы использовали метод `tight_layout` в примере 10.1, чтобы элементы графика не оказались перекрытыми или обрезанными. Однако `tight_layout` может вызывать проблемы в некоторых системах, особенно в macOS. Если у вас

возникли проблемы из-за нее, вам может помочь этот тред. На данный момент я бы посоветовал для текущей фигуры использовать метод `set_tight_layout`:

```
plt.gcf().set_tight_layout(True)
```

Сохранение диаграмм

Библиотека Matplotlib особенно эффективна при сохранении графиков, предлагая множество форматов вывода[1]. Доступность форматов зависит от доступных бэкендов, но PNG, PDF, PS, EPS и SVG почти всегда поддерживаются. PNG (Portable Network Graphics) — наиболее популярный формат для распространения веб-изображений. Остальные перечисленные форматы основаны на векторной графике и поддерживают плавное масштабирование без артефактов пикселизации. Для высококачественной печатной продукции это, вероятно, оптимальный выбор.

Сохранение — это просто:

```
plt.tight_layout() # подгонка графика к размерам фигуры
plt.savefig('mpl_3lines_custom.svg')
```

Можно задать формат явно с помощью `format="svg"`, но Matplotlib распознает расширение *.svg* автоматически. Метод `tight_layout` позволяет избежать обрезания меток[2].

Фигуры и объектно-ориентированная Matplotlib

Как мы видели, интерактивное управление глобальным состоянием `pyplot` прекрасно подходит для быстрого создания набросков визуализаций данных и работы с одиночными графиками. Однако для более точного управления графиками рекомендуется использовать объектно-ориентированный (OO) подход Matplotlib, основанный на работе с фигурами и осями. Большинство продвинутых примеров визуализации, встречающихся в интернете, выполнены именно таким способом.

[1] Помимо множества форматов данных, Matplotlib также поддерживает математический режим LaTeX, позволяющий использовать математические символы в заголовках, легендах и др. Одна из причин популярности Matplotlib в научных кругах — этот режим позволяет создавать графики, соответствующие требованиям научных публикаций.

[2] Подробнее см. на сайте Matplotlib.

Когда мы работаем с ОО Matplotlib, фигуру можно воспринимать как холст для рисования, со встроенными в него одной или несколькими осями (или графиками). И у фигур, и у осей есть свойства, которые настраиваются независимо. Таким образом, интерактивный подход через `pyplot`, о котором говорили выше, предполагает построение графика на единственной оси глобальной фигуры.

Мы можем создать фигуру с помощью метода `figure` из модуля `pyplot`:

```
fig = plt.figure(
      figsize=(8, 4), # размер фигуры в дюймах
      dpi=200, # в точках на дюйм
      tight_layout=True, # подгонка к холсту меток, осей и др.
      linewidth=1, edgecolor='r' # красная линия толщиной 1 пиксель
      )
```

Как видите, фигуры, созданные с помощью глобального модуля `pyplot`, имеют общее подмножество свойств. Свойства можно задать при создании фигуры или с помощью аналогичных методов (например, `fig.text()` вместо `plt.fig_text()`). У каждой фигуры несколько осей, каждая из осей — аналог единого глобального состояния графика, но имеет то преимущество, что в одной фигуре может существовать несколько осей с независимыми свойствами.

Оси и подграфики

Метод `figure.add_axes` позволяет точно контролировать положение осей внутри фигуры (например, встраивать меньший график в основной). Для позиционирования элементов графика используется система координат от 0 до 1, где 1 — максимальная ширина или высота фигуры. Можно указать позицию с помощью списка из четырех элементов или кортежа, установив нижнюю левую и верхнюю правую границы:

```
# h = высота, w = ширина
fig.add_axes([0.2, 0.2, #[снизу(h*0.2), слева(w*0.2),
              0.8, 0.8])# сверху(h*0.8), справа(w*0.8)]
```

В примере 10-2 показан код для вставки меньших осей в большие, с использованием случайных тестовых данных. Результат см. на рисунке 10.3.

Пример 10-2. *График, вставленный с помощью figure.add_axes*

```
fig = plt.figure(figsize=(8,4))
# --- Главная ось
ax = fig.add_axes([0.1, 0.1, 0.8, 0.8])
ax.set_title('Main Axes with Insert Child Axes')
ax.plot(x, y[:,0]) ❶
ax.set_xlabel('Date')
ax.set_ylabel('Cum. sum')
# --- Inserted Axes
ax = fig.add_axes([0.15, 0.15, 0.3, 0.3])
ax.plot(x, y[:,1], color='g') # 'g' — зеленый
ax.set_xticks([]); ❷
```

❶ Выбираем первый столбец случайного массива данных NumPy по оси Y.

❷ Убираем засечки (ticks) и метки с оси X вставленного графика.

Хотя `add_axes` позволяет делать тонкую настройку внешнего вида диаграмм, в большинстве случаев все значительно упрощает встроенная система сеток Matplotlib[1]. Самый простой вариант — использовать функцию `figure.subplots`, которая позволяет задавать разметку строк и столбцов для графиков одинакового размера. Если нужна сетка с графиками разных размеров, используйте модуль `gridspec`.

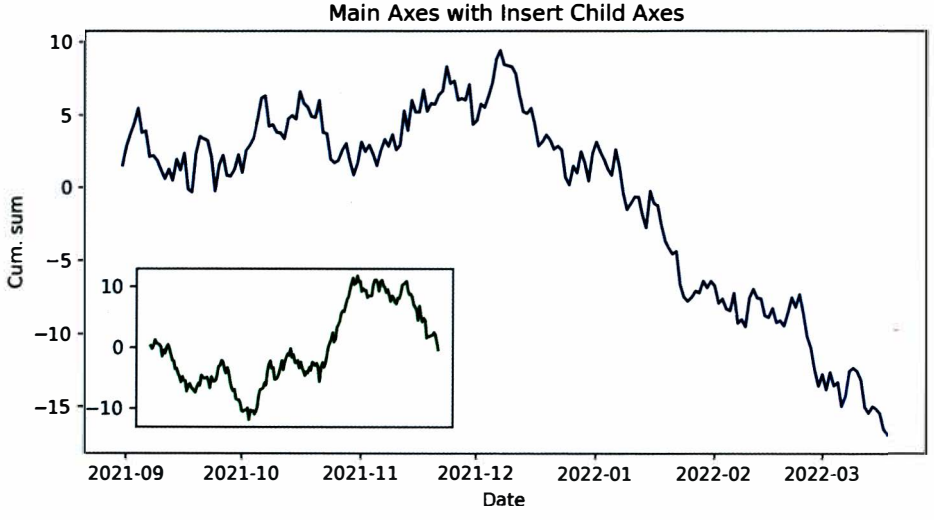


Рис. 10.3. *Вставка графика с помощью figure.add_axes*

[1] Метод `tight_layout` предназначен для оптимального размещения нескольких подграфиков.

При вызове `subplots` без аргументов будет возвращаться пустая фигура только с осями. Этот подход максимально близок к использованию конечного автомата `pyplot` (см. «Создание интерактивных графиков с помощью глобального состояния pyplot» на стр. 316). В примере 10-3 показана фигура и оси, эквивалентные тем, что мы делали с помощью `pyplot` в примере 10-1, получив результат, показанный на рисунке 10.2. Обратите внимание на использование методов-сеттеров (setter) для фигуры и осей.

Пример 10-3. *Построение графика с одной фигурой и осями*

```
figure, ax = plt.subplots()
plots = ax.plot(x, y, label='')
figure.set_size_inches(8, 4)
ax.legend(plots, ('foo', 'bar', 'baz'), loc='best', framealpha=0.25,
          prop={'size':'small', 'family':'monospace'})
ax.set_title('Random trends')
ax.set_xlabel('Date')
ax.set_ylabel('Cum. sum')
ax.grid(True)
figure.text(0.995, 0.01, '© Acme Designs 2022',
            ha='right', va='bottom')
figure.tight_layout()
```

Вызов `subplots` с аргументами `nrows` (число строк) и `ncols` (число столбцов), как в примере 10-4, позволяет разместить множество графиков на сетке. Результат см. на рисунке 10.4. При таком вызове `subplots` возвращает фигуру и массив осей, упорядоченных по строкам и колонкам. В этом примере мы задаем один столбец, так что `axes` представляет собой одномерный массив их трех осей, расположенных друг над другом.

Пример 10-4. *Использование подграфиков*

```
fig, axes = plt.subplots(
                nrows=3, ncols=1, ❶
                sharex=True, sharey=True, ❷
                figsize=(8, 8))
labelled_data = zip(y.transpose(), ❸
                ('foo', 'bar', 'baz'), ('b', 'g', 'r'))
fig.suptitle('Three Random Trends', fontsize=16)
```

```
for i, ld in enumerate(labelled_data):
    ax = axes[i]
    ax.plot(x, ld[0], label=ld[1], color=ld[2])
    ax.set_ylabel('Cum. sum')
    ax.legend(loc='upper left', framealpha=0.5,
              prop={'size':'small'})
axes[-1].set_xlabel('Date') ❹
```

❶ Задаем сетку подграфиков с одним столбцом и тремя строками.
❷ Делаем общими оси X и Y и настраиваем размеры для удобства сравнения.
❸ Транспонируем Y и объединяем данные линий, метки и цвета линий.
❹ Ставим метку по последней (нижней) из общих осей X.

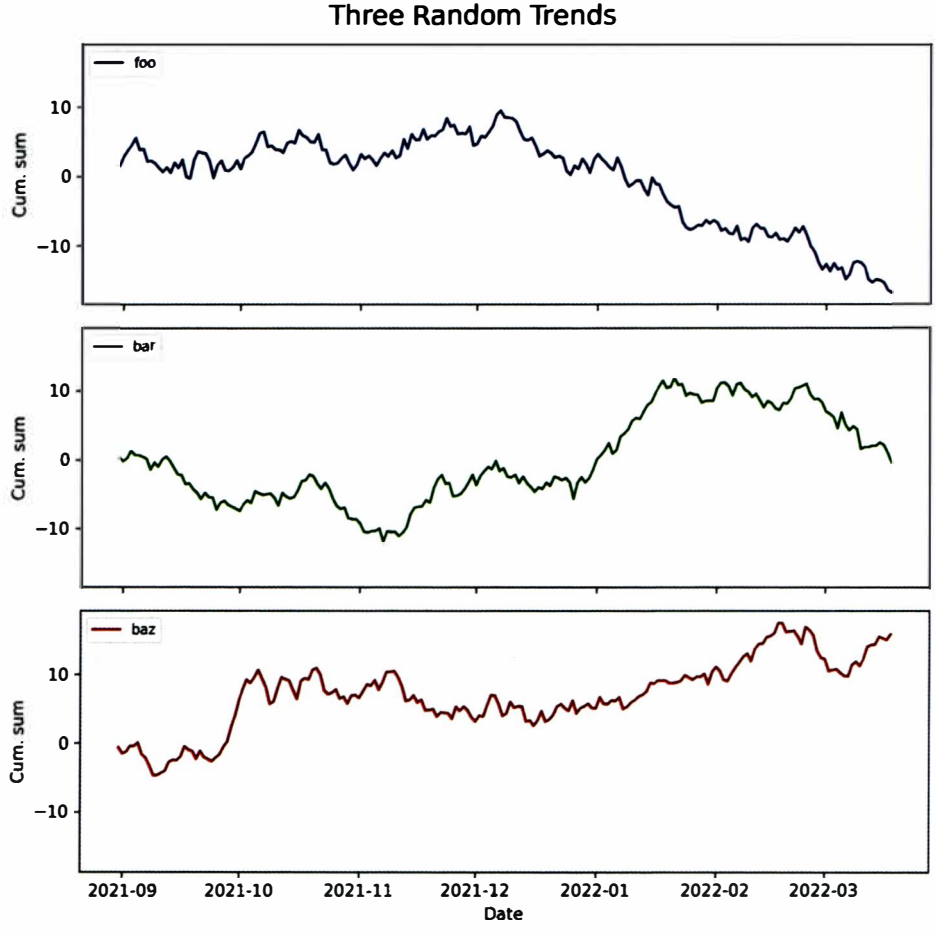


Рис. 10.4. Три подграфика

Мы использовали удобный метод `zip` в Python, чтобы сгенерировать три кортежа с данными линий. Функция `zip` принимает итерируемые объекты длиной *n* (списки или кортежи) и возвращает список из *n* кортежей, формируемый путем сопоставления элементов по порядку.

```
letters = ['a', 'b']
numbers = [1, 2]
zip(letters, numbers)
Out:
[('a', 1), ('b', 2)]
```

В цикле `for` мы применяем функцию `enumerate`, которая дает значения индекса `i` для выбора оси по строке. Объединенные с помощью zip `labelled_data` предоставляют свойства графика.

Обратите внимание на общие оси X и Y, указанные в вызове `subplots` в примере 10-4 (❷). Это упрощает сравнение трех диаграмм, особенно благодаря нормализованной оси Y. Чтобы не перегружать графики избыточными подписями под осями X, мы вызвали `set_xlabel` только для последней строки, используя отрицательную индексацию Python.

Мы рассмотрели два способа интерактивного взаимодействия IPython и Matplotlib — через глобальное состояние (с доступом через `plt`) и объектно-ориентированный API. Теперь перейдем к основным типам графиков для анализа данных.

Типы графиков

Кроме только что продемонстрированного линейного графика, Matplotlib поддерживает множество других их типов. Я покажу пару тех, которые обычно используются для исследовательских визуализаций данных.

Столбчатые диаграммы

Простая столбчатая диаграмма — основа для визуального анализа данных. Как и большинство графиков Matplotlib, столбчатые диаграммы предлагают широкие возможности настройки. Пробежимся по нескольким вариантам, чтобы уяснить суть столбчатых диаграмм.

На рисунке 10.5 изображен результат выполнения кода из примера 10-5. Обратите внимание, что расположение столбиков и подписей необходимо задавать

вручную. Гибкость такого рода ценят закоренелые любители Matplotlib и быстро осваивают новички. Тем не менее, порой такие вещи утомляют. Работу с Matplotlib упрощают множество библиотек-оберток, к тому же не составляет труда написать вспомогательные методы. Как мы увидим в главе 11, графики на базе Matplotlib, встроенные в pandas, гораздо проще использовать.

Пример 10-5. *Простая столбчатая диаграмма*

```
labels = ["Physics", "Chemistry", "Literature", "Peace"]
foo_data = [3, 6, 10, 4]

bar_width = 0.5
xlocations = np.array(range(len(foo_data))) + bar_width ❶
plt.bar(xlocations, foo_data, width=bar_width)
plt.yticks(range(0, 12)) ❷
plt.xticks(xlocations, labels) ❸
plt.title("Prizes won by Fooland")
plt.gca().get_xaxis().tick_bottom()
plt.gca().get_yaxis().tick_left()
plt.gcf().set_size_inches((8, 4))
```

❶ Выравниваем столбики по центру, на расстоянии двух `bar_width` друг от друга.

❷ Жестко задаем значения X, но обычно диапазон вычисляется на лету.

❸ Размещаем засечки и подписи по центру столбиков.

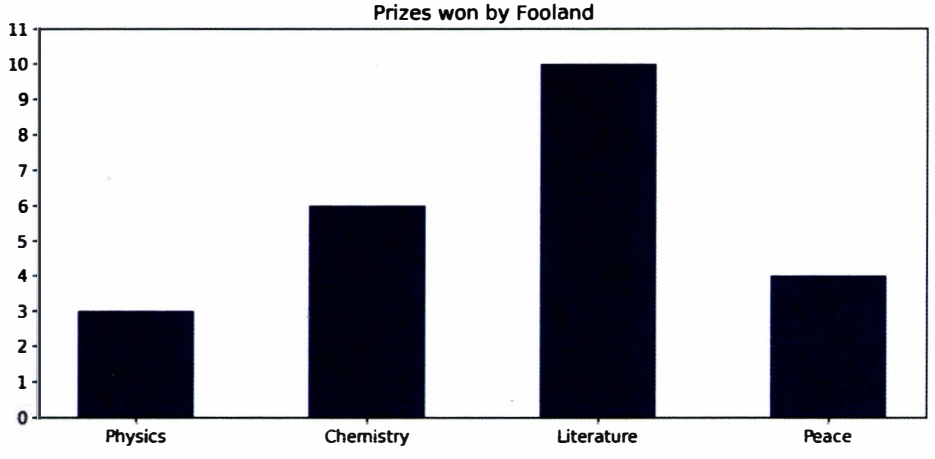


Рис. 10.5. *Простая столбчатая диаграмма*

SKLADCHIK.ORG

Особенно полезны столбчатые диаграммы с несколькими группами. В примере 10-6 мы добавим в код еще немного данных по стране (для мифического Барлэнда) и с помощью функции `subplots` создадим столбчатые диаграммы с группировкой (см. рисунок 10.6). Еще раз зададим расположение столбиков вручную, добавляя две группы столбиков — на этот раз с помощью `ax.bar`. Обратите внимание, что пределы оси X автоматически масштабируются вполне разумно, с шагом 0,5:

```
ax.get_xlim()
# Вывод: (-0.5, 3.5)
```

Используйте подходящие сеттеры (в данном случае `set_xlim`), если автоматическое масштабирование не позволяет добиться желаемого вида.

Пример 10-6. *Создание столбчатой диаграммы с группировкой*

```
labels = ["Physics", "Chemistry", "Literature", "Peace"]
foo_data = [3, 6, 10, 4]
bar_data = [8, 3, 6, 1]

fig, ax = plt.subplots(figsize=(8, 4))
bar_width = 0.4 ❶
xlocs = np.arange(len(foo_data))
ax.bar(xlocs-bar_width, foo_data, bar_width,
       color='#fde0bc', label='Fooland') ❷
ax.bar(xlocs, bar_data, bar_width, color='peru', label='Barland')
#--- ticks, labels, grids, and title ax.set_yticks(range(12))
ax.set_xticks(ticks=range(len(foo_data)))
ax.set_xticklabels(labels)
ax.yaxis.grid(True)
ax.legend(loc='best')
ax.set_ylabel('Number of prizes')
fig.suptitle('Prizes by country')
fig.tight_layout(pad=2) ❸
fig.savefig('mpl_barchart_multi.png', dpi=200) ❹
```

❶ При общей ширине 1 групп из двух столбиков указанная величина bar_width дает внутренние отступы в 0,1 столбца.

❷ Matplotlib поддерживает стандартные HTML-цвета, принимая как их наименования, так и hex-код.

❸ Используем аргумент `pad`, чтобы задать отступ внутри фигуры, как долю от размера шрифта.

❹ Сохраняем фигуру с высоким разрешением в 200 dpi.

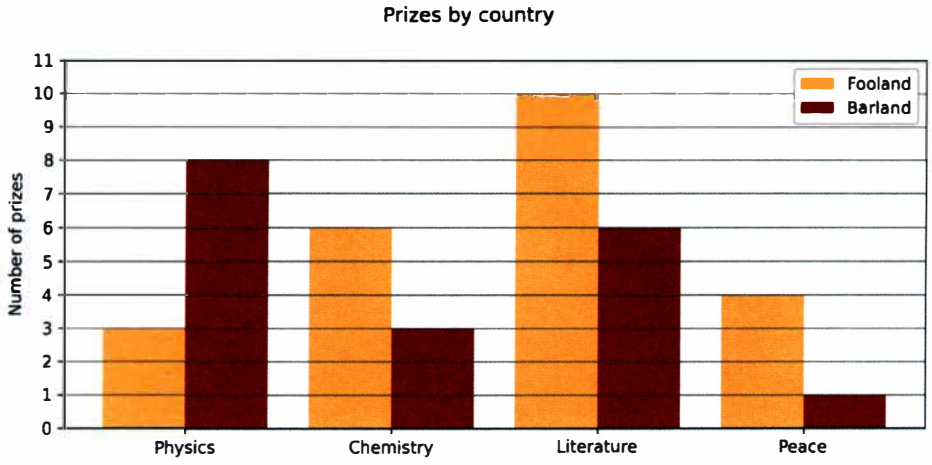


Рис. 10.6. *Сгруппированные столбчатые диаграммы*

Когда столбиков много и/или используются метки, то имеется риск, что они будут накладываться друг на друга, если их разместить на одной и той же линии. В таком случае полезно использовать гистограмму с горизонтальными столбцами. Перевернуть рисунок 10.6 на бок достаточно просто: заменим метод `bar` его горизонтальным аналогом `barh` и поменяем местами метки осей и пределы (см. пример 10-7 и рисунок 10.7).

Пример 10-7. *Преобразуем пример 10-6 для получения горизонтальных столбиков*

```
# ...
ylocs = np.arange(len(foo_data))
ax.barh(ylocs-bar_width, foo_data, bar_width, color='#fde0bc',
        label='Fooland') ❶
ax.barh(ylocs, bar_data, bar_width, color='peru', label='Barland')
# -- labels, grids and title, then save ax.set_xticks(range(12)) ❷
ax.set_yticks(ticks=ylocs-bar_width/2)
ax.set_yticklabels(labels)
ax.xaxis.grid(True) ax.legend(loc='best')
```

```
ax.set_xlabel('Number of prizes')
# ...
```

❶ Вместо `bar` используем `barh`, чтобы создать горизонтальную столбчатую диаграмму.

❷ Меняем местами горизонтальную и вертикальную оси.

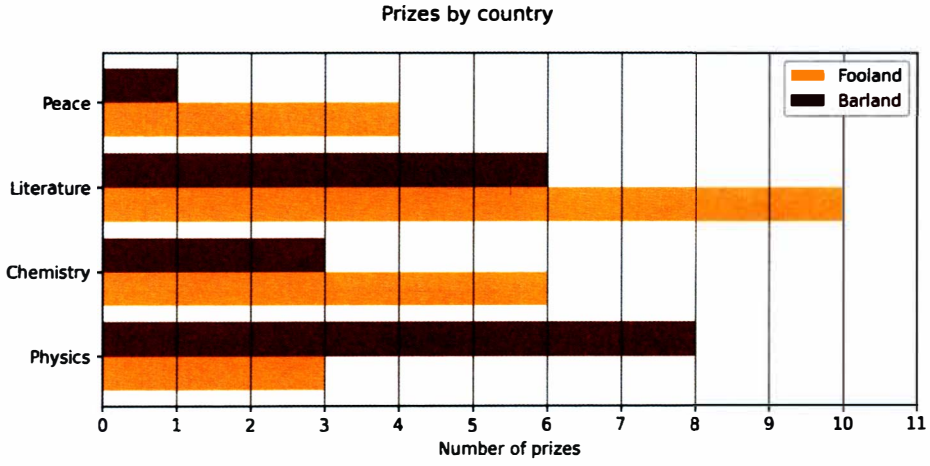


Рис. 10.7. *Столбики, повернутые на бок*

В Matplotlib легко получить сложенные столбики[1]. В примере 10-8 показано, как преобразовать гистограмму с рисунка 10.6 в многорядную сложенную столбчатую диаграмму. Результат работы кода см. на рисунке 10.8. Ключевой прием — использование в методе `bar` аргумента `bottom`, чтобы установить основание новых столбиков на уровень вершин предыдущей группы.

Пример 10-8. *Преобразование примера 10-6 для построения многорядной сложенной столбчатой диаграммы*

```
# ...
bar_width = 0.8
xlocs = np.arange(len(foo_data))
ax.bar(xlocs, foo_data, bar_width, color='#fde0bc', ❶
       label='Fooland')
```

[1] Сомнительно, что многорядные сложенные столбчатые диаграммы так уж хороши для оценки групп данных. См. в блоге Соломона Мессинга (Solomon Messing) неплохой разбор этой темы и один пример «удачного» использования.

```
ax.bar(xlocs, bar_data, bar_width, color='peru', ❷
       label='Barland', bottom=foo_data)
# --- labels, grids and title, then save
ax.set_yticks(range(18))
ax.set_xticks(ticks=xlocs)
ax.set_xticklabels(labels)
# ...
```

❶ Группы столбиков `foo_data` и `bar_data` размещены в одинаковых позициях по оси X.

❷ Нижняя граница группы `bar_data` соответствует верхней границе группы `foo_data` — получаются сложенные столбики.

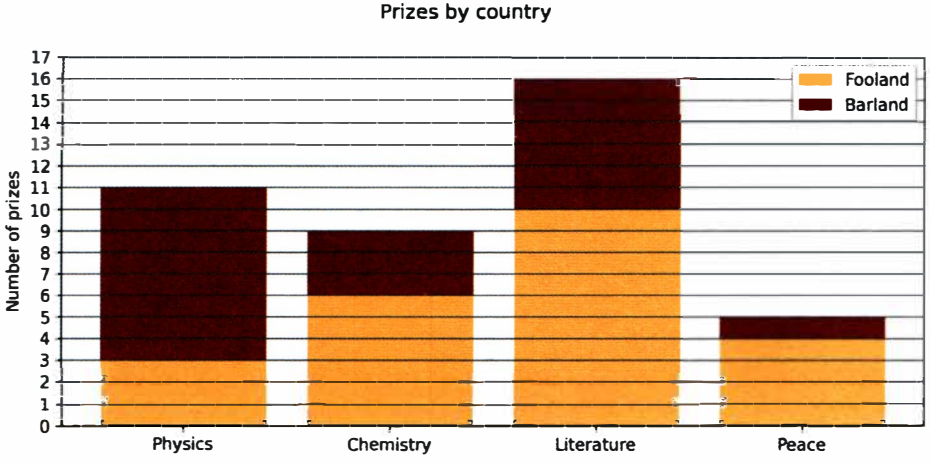


Рис. 10.8. *Многорядная сложенная столбчатая диаграмма*

Диаграмма рассеивания

Еще один полезный тип визуализации — диаграмма рассеивания (scatter plot), которая работает с двумерными массивами точек, предоставляя возможность выбора размера точки, ее цвета и др.

Код в примере 10-9 быстро создает диаграмму рассеивания, используя автомасштабирование Matplotlib для пределов x и y. Мы создаем зашумленную линию, добавляя случайные числа с нормальным распределением (стандартное отклонение σ = 10). На рисунке 10.9 показана полученная диаграмма.

Пример 10-9. *Простая диаграмма рассеивания*

```
num_points = 100
gradient = 0.5
x = np.array(range(num_points))
y = np.random.randn(num_points) * 10 + x*gradient ❶
fig, ax = plt.subplots(figsize=(8, 4)) ax.scatter(x, y) ❷

fig.suptitle('A Simple Scatterplot')
```

❶ Функция `randn` генерирует случайные числа с нормальным распределением, которые мы масштабируем (σ = 10) и комбинируем с линейной зависимостью от X.

❷ Массивы X и Y одинакового размера предоставляют координаты точек.

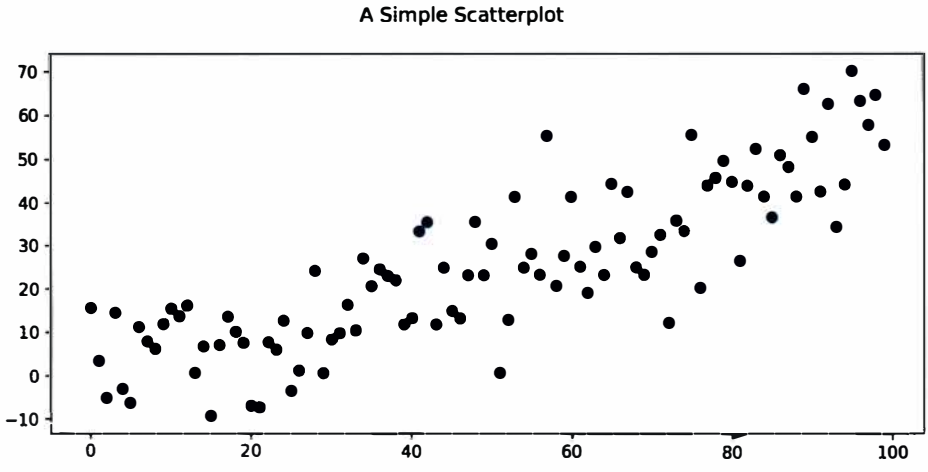


Рис. 10.9. *Простая диаграмма рассеивания*

Можно настроить размер и цвет отдельных точек, передав массивы размеров маркеров и цветовых индексов в текущую цветовую карту по умолчанию. Следует отметить, что мы задаем площадь ограничивающих квадратных маркеров, а не диаметры кругов. То есть, чтобы диаметр кругов увеличился в два раза, площадь маркеров нужно увеличить в четыре раза[1]. В примере 10-10 мы добавляем информацию о размере и цвете к нашей диаграмме рассеивания. Получается рисунок 10.10.

[1] Указание площади маркера вместо ширины или радиуса — оптимальный подход по умолчанию, так как обеспечивает пропорциональность отображаемым значениям.

Пример 10-10. *Настройка размера и цвета точек*

```
num_points = 100
gradient = 0.5
x = np.array(range(num_points))
y = np.random.randn(num_points) * 10 + x*gradient
fig, ax = plt.subplots(figsize=(8, 4))
colors = np.random.rand(num_points) ❶
size = np.pi * (2 + np.random.rand(num_points) * 8) ** 2 ❷
ax.scatter(x, y, s=size, c=colors, alpha=0.5) ❸
fig.suptitle('Scatterplot with Color and Size Specified')
```

❶ Получаем 100 случайных значений цветов в диапазоне от 0 до 1 для карты цветов по умолчанию.

❷ Используем обозначение `**` для возведения в квадрат значений из диапазона ширины маркеров (2–10).

❸ Используем аргумент `alpha`, чтобы сделать маркеры полупрозрачными.

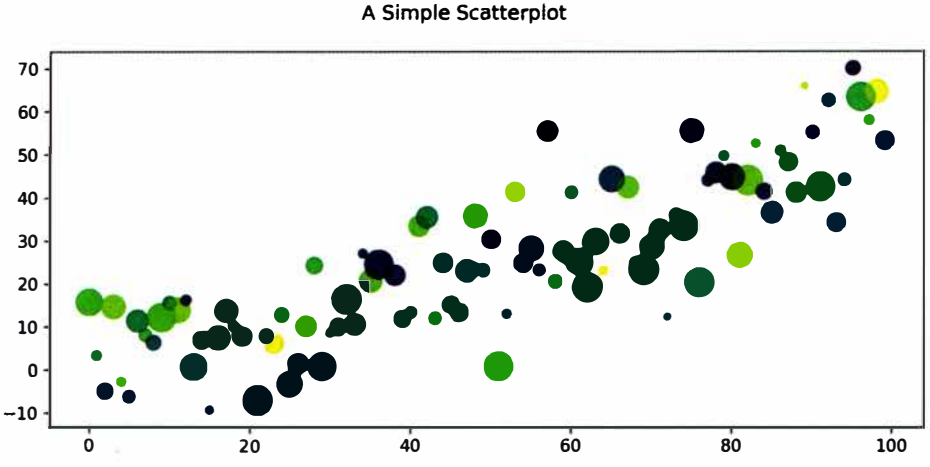


Рис. 10.10. *Настройка размера и цвета точек*



Цветовые карты в Matplotlib

Matplotlib располагает огромным разнообразием цветовых карт, выбор цветов в которых может значительно улучшить качество визуализации.

Добавление линии регрессии

Линия регрессии — это простая прогностическая модель корреляции между двумя переменными, в данном случае координатами X и Y нашей диаграммы рассеивания. Это линия наилучшего соответствия, проходящая через точки диаграммы. Ее добавление демонстрирует полезный прием визуализации данных и позволяет наглядно показать взаимодействие Matplotlib с NumPy.

В примере 10-11 функция `polyfit` библиотеки NumPy используется для вычисления коэффициента наклона (m) и свободного члена (c) линии регрессии для точек, заданных массивами X и Y. Затем мы строим эту линию на тех же осях, что и диаграмму рассеивания (см. рисунок 10.11).

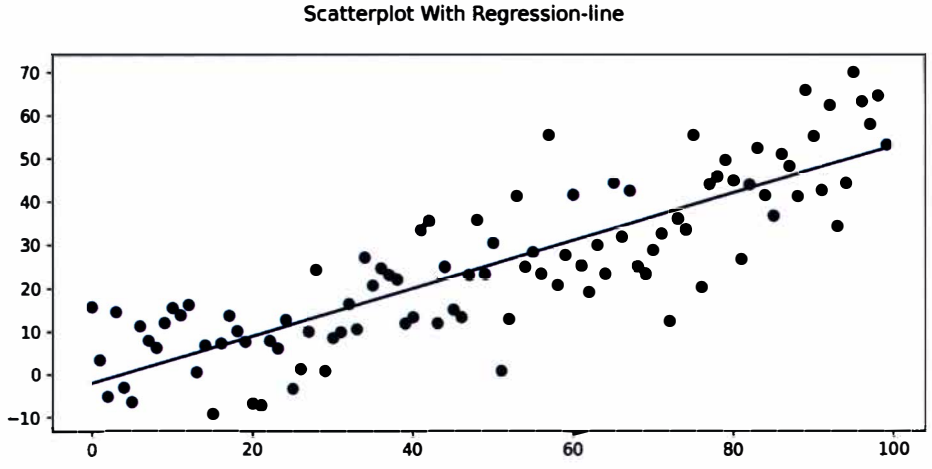


Рис. 10.11. *Диаграмма рассеивания с линией регрессии*

Пример 10-11. *Диаграмма рассеивания с линией регрессии*

```
num_points = 100
gradient = 0.5
x = np.array(range(num_points))
y = np.random.randn(num_points) * 10 + x*gradient
fig, ax = plt.subplots(figsize=(8, 4))
ax.scatter(x, y)
m, c = np.polyfit(x, y,1) ❶
ax.plot(x, m*x + c) ❷
fig.suptitle('Scatterplot With Regression-line')
```

❶ Используем NumPy `polyfit` с полиномом первой степени, чтобы получить градиент линии (`m`) и константу (`c`) для линии наилучшего соответствия, проходящей через случайные точки.

❷ Используем градиент и константу, чтобы построить линию на осях диаграммы рассеивания (`y = mx + c`).

Обычно при создании линии регрессии рекомендуется строить доверительные интервалы. Они дают представление о том, насколько надежно подогнана линия на основе количества и распределения точек. Доверительные интервалы можно построить с помощью Matplotlib и NumPy, но это довольно неудобно. К счастью, есть библиотека, созданная на основе Matplotlib, предлагающая дополнительные специализированные функции для статистического анализа и визуализации данных, которые по мнению многих лучше, чем стандартные графики Matplotlib по умолчанию. Эта библиотека называется seaborn, и сейчас мы немного о ней поговорим.

Seaborn

Существует ряд библиотек, которые оборачивают мощные функции построения графиков Matplotlib, так что ими становится удобнее пользоваться[1] и, что особенно важно для специалистов по визуализации данных, бесшовно работают с pandas.

Bokeh — библиотека для создания интерактивных визуализаций в браузере. Отлично работает с блокнотом IPython (Jupyter Notebook). Создание Bokeh — большое достижение. Философия библиотеки схожа с философией D3.js[2].

Но для интерактивного анализа, необходимого для глубокого понимания ваших данных и выбора оптимальных способов визуализации, я рекомендую seaborn. Библиотека расширяет возможности Matplotlib, предлагая мощные инструменты для построения статистических графиков. Она хорошо интегрирована со стеком PyData, прекрасно взаимодействует с NumPy, pandas и статистическими процедурами, которые предлагают SciPy и statsmodels.

Важным преимуществом seaborn является открытый доступ к Matplotlib API, что позволяет настраивать диаграммы с помощью обширного инструментария

[1] Общепризнано, что настройки Matplotlib по умолчанию оставляют желать лучшего, и их с легкостью может улучшить любая оболочка.

[2] Обе библиотеки — D3 и Bokeh — отдают дань уважения классическому труду по визуализации данных: «Грамматике графики» Леланда Уилкинсона (The Grammar of Graphics by Leland Wilkinson) (Springer).

Matplotlib. Библиотека seaborn — это не замена Matplotlib и соответствующих ей навыков, а впечатляющее расширение.

Для работы с seaborn нужно просто расширить стандартный импорт Matplotlib:

```
import numpy as np import pandas as pd
import seaborn as sns # опирается на matplotlib
import matplotlib as mpl
import matplotlib.pyplot as plt
```

Matplotlib предлагает ряд стилей для графиков, которые можно активировать, вызвав метод `use` с ключом стиля. Используем стиль, установленный в seaborn по умолчанию, который добавит к графикам полупрозрачную серую сетку:

```
matplotlib.style.use('seaborn')
```

Все доступные стили и их визуальные эффекты см. в документации Matplotlib.

Многие из функций seaborn рассчитаны на работу с DataFrame из pandas, позволяя задавать, например, столбцы с координатами для двумерного распределения точек. Возьмем массивы X и Y из примера 10.9 и создадим на их основе тестовый набор данных:

```
data = pd.DataFrame({'dummy x': x, 'dummy y': y})
```

Теперь у нас есть набор данных `data` со столбцами координат X (`'dummy x'`) и Y (`'dummy y'`). В примере 10-12 показано использование специализированной функции `lmplot` из seaborn для построения графика линейной регрессии, представленного на рисунке 10.12. Обратите внимание, что для некоторых графиков seaborn размер фигуры задается высотой (в дюймах) и соотношением сторон (ширина/высота). Также обратите внимание, что у seaborn общий глобальный контекст с `pyplot`.

Пример 10-12. *График линейной регрессии с помощью seaborn*

```
data = pd.DataFrame({'dummy x': x, 'dummy y': y})
sns.lmplot(data=data, x='dummy x', y='dummy y', ❶
           height=4, aspect=2) ❷
plt.tight_layout() ❸
plt.savefig('mpl_scatter_seaborn.png') ❸
```

❶ Аргументы x и y задают имена столбцов в DataFrame, которые определяют координаты точек графика.

❷ Указываем высоту в дюймах и соотношение ширины к высоте, чтобы задать размер фигуры. Здесь взято соотношение, равное 2, чтобы лучше уложиться в размер страницы этой книги.

❸ То, что seaborn делит глобальный контекст с `pyplot`, позволяет сохранять ее графики аналогично Matplotlib.

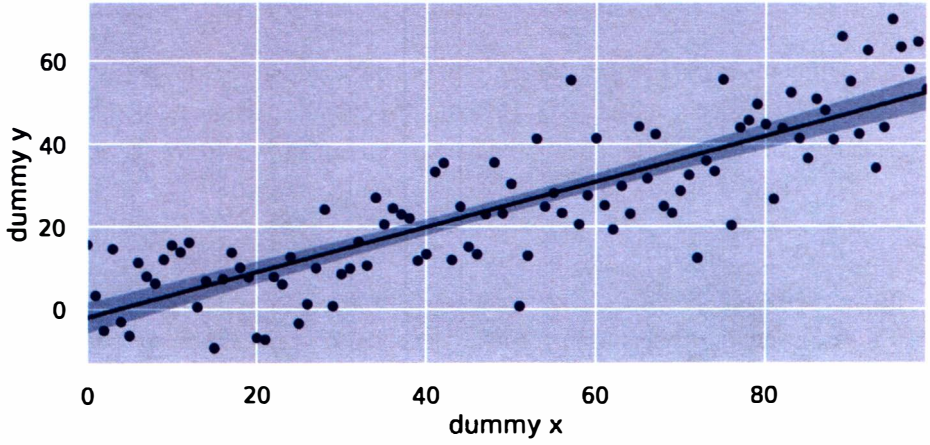


Рис. 10.12. *График линейной регрессии с помощью seaborn*

Библиотека seaborn особое внимание уделяет внешнему виду графиков, поэтому в ней имеется богатый набор визуальных настроек. Внесем несколько изменений в оформление диаграммы с рисунка 10.12 и установим для стандартной ошибки доверительный интервал 68 % (результат см. на рисунке 10.13):

```
sns.lmplot(data=data, x='dummy x', y='dummy y', height=4, aspect=2,
           scatter_kws={"color": "slategray"}, ❶
           line_kws={"linewidth": 2, "linestyle":'--', ❷
                     "color": "seagreen"},
           markers='D', ❸
           ci=68) ❹
```

❶ Укажем ключевые аргументы компонента диаграммы рассеивания: зададим сланцево-серый цвет точек.

❷ Укажем ключевые аргументы компонента линейной диаграммы: зададим толщину и стиль линии.

SKLADCHIK.ORG

❸ Настроим маркеры графика в виде ромбов, используя Matplotlib-код маркеров *D*.

❹ Установим для стандартной ошибки доверительный интервал 68 %.

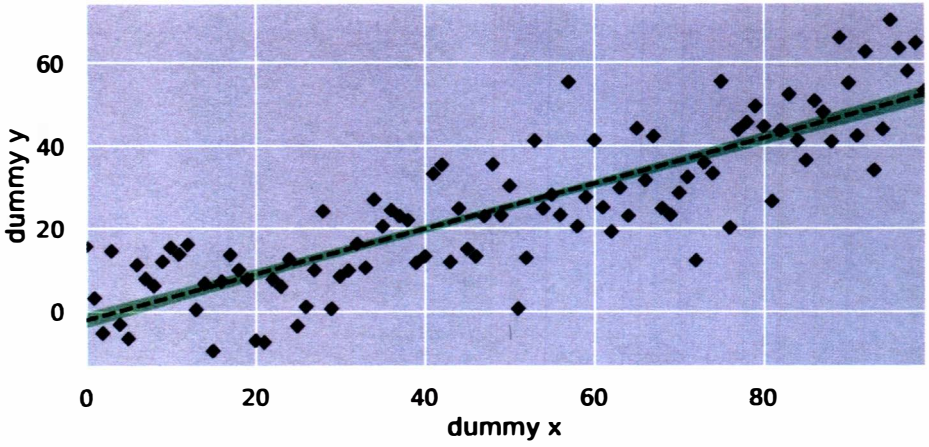


Рис. 10.13. *Настройка диаграммы рассеивания в seaborn*

В дополнение к базовому набору Matplotlib библиотека seaborn предлагает ряд полезных графиков. Рассмотрим один из самых интересных примеров использования FacetGrid из библиотеки seaborn для визуализации многомерных данных.

FacetGrid

Построение нескольких экземпляров одного и того же графика для разных подмножеств данных (часто это называется lattice или trellis plotting) — это хороший способ получить общее представление о структуре данных. Этот подход позволяет отображать на одном графике значительные объемы информации, а также визуализировать взаимосвязи между различными измерениями. Данный метод основан на концепции small multiples («малых кратных»), популяризированной Эдвардом Тафти (Edward Tufte).

Объект FacetGrid требует, чтобы данные были в формате pandas DataFrame и в «аккуратном» (tidy) формате по Хэдли Уикхэму (Hadley Wickham), создателю ggplot2: каждая переменная располагается в столбце, а каждое наблюдение — в строке.

Воспользуемся тестовым набором данных Tips[1] из seaborn, чтобы продемонстрировать FacetGrid в действии. Этот небольшой набор данных содержит

[1] В seaborn есть несколько удобных наборов данных, который можно найти на GitHub.

информацию о распределении чаевых по различным параметрам: день недели, курящий[1] ли посетитель и так далее. Сначала загрузим данные в DataFrame pandas с помощью метода `load_dataset`:

```
In [0]: tips = sns.load_dataset('tips')
Out[0]:
   total_bill    tip      sex      smoker   day   time     size
0     16.99      1.01     Female   No       Sun   Dinner   2
1     10.34      1.66     Male     No       Sun   Dinner   3
2     21.01      3.50     Male     No       Sun   Dinner   3
3     23.68      3.31     Male     No       Sun   Dinner   2
...
```

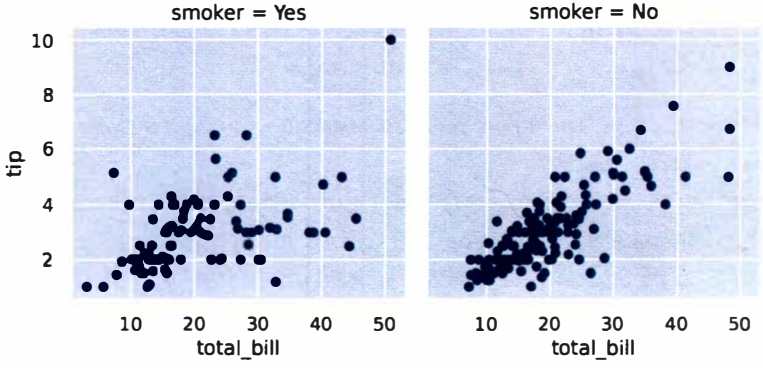


Рис. 10.14. *seaborn FacetGrid с диаграммами рассеивания*

Чтобы создать FacetGrid мы задали DataFrame `tips` и интересующий нас столбец, указывающий, курит ли клиент. Будем использовать этот столбец при построении групп графиков. Есть две категории в столбце smoker: курящие (`'smoker=Yes'`) и некурящие (`'smoker=No'`), поэтому в FacetGrid будет две диаграммы. Затем мы используем метод FacetGrid `map`, чтобы создать несколько диаграмм рассеивания размера чаевых относительно общей суммы счета:

```
g = sns.FacetGrid(tips, col="smoker", height=4, aspect=1)
g.map(plt.scatter, "total_bill", "tip") ❶
```

❶ Метод `map` принимает инстанс графика, в данном случае `plt.scatter`, и два параметра (`tips`), необходимые для диаграммы рассеивания.

[1] Курение вредит вашему здоровью.

Мы получили две диаграммы рассеивания (см. рисунок 10.14), по одной для каждого из статусов курения с корреляцией суммы чаевых и общего счета.

Мы можем включить еще одно измерение данных `tips`, определив свойства маркеров в диаграммах рассеяния. Сделаем маркеры в виде красных ромбов для женщин и синих квадратов для мужчин:

```
pal = dict(Female='red', Male='blue')
g = sns.FacetGrid(tips, col="smoker",
                  hue="sex", hue_kws={"marker": ["D", "s"]}, ❶
                  palette=pal, height=4, aspect=1,)
g.map(plt.scatter, "total_bill", "tip", alpha=.4)
g.add_legend();
```

❶ Добавим цвет маркеров (`hue`) в зависимости от измерения `sex`, укажем форму ромба (`D`) и квадрата (`s`) и применим палитру цветов (`pal`), чтобы сделать маркеры красными и синими. Результат см. на рисунке 10.15.

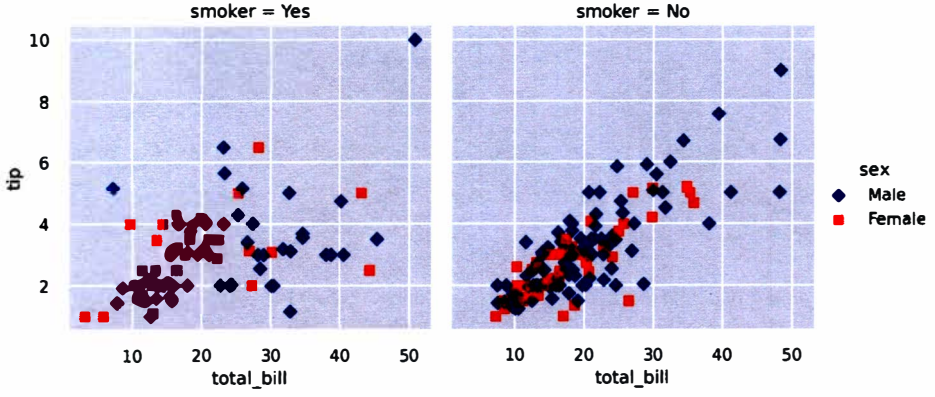


Рис. 10.15. *График рассеивания с ромбовидными (D) и квадратными маркерами (s) для обозначения пола посетителей*

Чтобы создать подмножество данных по измерению мы можем использовать не только столбцы, но и строки. Сочетая строки со столбцами, можно с помощью функции `regplot`[1] исследовать пять измерений:

[1] `regplot` — сокращение от regression plot (график регрессии), функционально эквивалентен lmplot, использовавшемуся в примере 10.12. В lmplot удобно объединяются `regplot` и FacetGrid.

```
pal = dict(Female='red', Male='blue')
g = sns.FacetGrid(tips, col="smoker", row="time", ❶
                  hue="sex", hue_kws={"marker": ["D", "s"]},
                  palette=pal, height=4, aspect=1,)
g.map(sns.regplot, "total_bill", "tip", alpha=.4)
g.add_legend();
```

❶ Добавим строку time, чтобы разделить чаевые во время обеда и ужина.

На рисунке 10.16 показано четыре графика `regplot`, создающие модель линейной регрессии с доверительными интервалами для женских и мужских групп с соответствующими цветами. Заголовки графиков показывают, какие подмножества данных использовались. Каждая строка имеет время и статус курит/не курит.

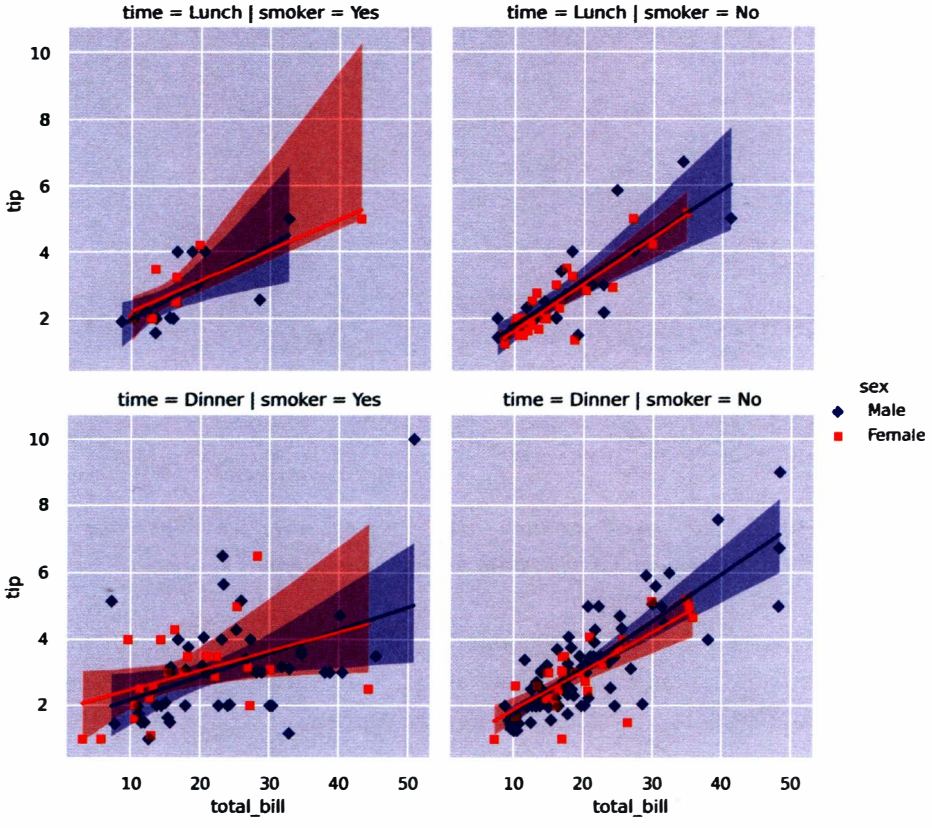


Рис. 10.16. *Визуализация пяти измерений*

Мы можем добиться того же эффекта, используя функцию `lmplot`, которую мы видели в примере 10-12, для удобства объединяющую функциональность FacetGrid и `regplot`. Результат выполнения приведенного ниже кода см. на рисунке 10.16:

```
pal = dict(Female='red', Male='blue')
sns.lmplot(x="total_bill", y="tip", hue="sex",
           markers=["D", "s"], ❶
           col="smoker", row="time", data=tips, palette=pal,
           height=4, aspect=1
           );
```

❶ Обратите внимание, что мы использовали ключевое слово `markers` вместо словаря `kws_hue`, как в примере с FacetGrid.

`lmplot` предлагает удобный сокращенный синтаксис для FacetGrid `regplot`, однако метод `map` в FacetGrid позволяет задействовать весь арсенал диаграмм seaborn и Matplotlib для визуализации многомерных подмножеств данных. Это мощный метод, позволяющий проводить углубленный анализ данных.

PairGrid

Еще один интересный тип графиков seaborn, который позволяет быстро оценивать многомерные данные — PairGrid. В отличие от FacetGrid, набор данных не приходится делить на подмножества, которые сравниваются по заданным измерениям. С помощью PairGrid все измерения набора данных сравниваются попарно в квадратной сетке. По умолчанию сравниваются все измерения, но при объявлении PairGrid[1] можно указать для отображения список конкретных переменных через параметр `vars`.

Продемонстрируем пользу попарного сравнения на примере классического набора данных Iris, показав некоторые важные статистические методы на наборе, содержащем данные трех видов ирисов. Сначала загрузим этот набор данных:

```
In [0]: iris = sns.load_dataset('iris')
In [1]: iris.head()
Out[1]:
```

[1] Параметры `x_vars` и `y_vars` позволяют задавать сетки без квадратов.

```
   sepal_length  sepal_width  petal_length  petal_width  species
0  5.1           3.5          1.4           0.2          setosa
1  4.9           3.0          1.4           0.2          setosa
2  4.7           3.2          1.3           0.2          setosa
...
```

Чтобы отразить взаимосвязь между размерами лепестков и чашелистиков по видам, мы создаем объект `PairGrid`, цветовую кодировку (hue) по видам (`species`), а затем применяем методы сопоставления, чтобы создать графики как на диагонали, так и вне диагонали попарной сетки. Результат см. на рисунке 10.17:

```
sns.set_theme(font_scale=1.5) ❶
g = sns.PairGrid(iris, hue="species") ❷
g.map_diag(plt.hist) ❸
g.map_offdiag(plt.scatter) ❹
g.add_legend();
```

❶ Настраиваем размер шрифта с помощью метода seaborn `set_theme` (все параметры настройки см. в документации).

❷ Указываем цвет маркеров вспомогательных элементов (subbars) в соответствии с видами ирисов.

❸ Размещаем гистограммы характеристик видов по диагонали сетки.

❹ Используем стандартные диаграммы рассеивания для сравнения параметров на диагонали.

Как видно на рисунке 10.17, всего лишь несколько строк кода seaborn значительно упрощают создание информативного набора графиков, коррелирующих различные показатели ирисов. Этот график, который называется матрицей диаграммы рассеивания (scatter-plot matrix), — отличный способ для выявления линейных корреляций между парами переменных в многомерном наборе данных. В нынешнем виде в сетке есть избыточность, например: графики для `sepal_width-petal_length` и `petal_length-septal_width`. `PairGrid` позволяют перенести избыточные графики выше или ниже главной диагонали, чтобы обеспечить иное отражение данных. Дополнительные примеры см. в документации seaborn[1].

[1] Для любознательных: на *сайте bl.ocks.org* есть пример построения матрицы диаграммы рассеивания с помощью D3.js.

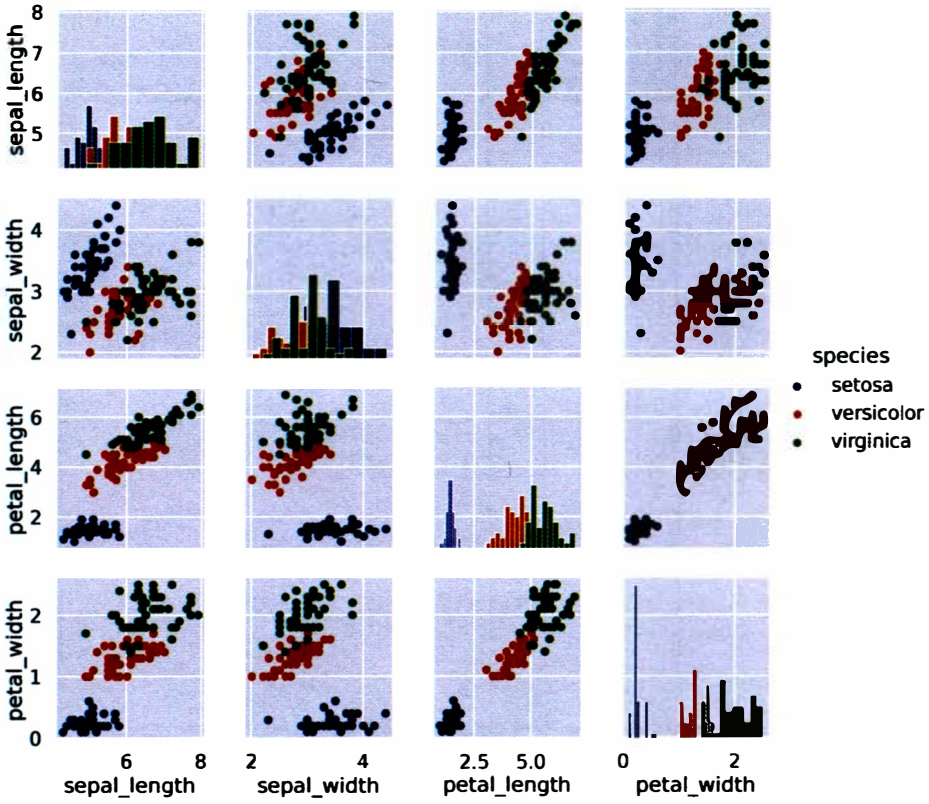


Рис. 10.17. *Итоговая оценка ирисов с помощью PairGrid*

Я познакомил вас с некоторыми графиками seaborn, еще несколько вы увидите в следующей главе, когда мы будем исследовать набор данных о лауреатах Нобелевской премии. В seaborn много очень удобных и мощных инструментов построения графиков, в основном статистических. Рекомендую продолжить изучение этой библиотеки, начиная с основной документации seaborn.

Там есть отличные примеры, хорошо документированный API и несколько руководств, которые дополняют то, что вы узнали из этой главы.

Резюме

Эта глава познакомила вас с Python-библиотекой Matplotlib — мощным инструментом для визуализации данных. Это зрелая и обширная библиотека с подробной документацией и активным сообществом. Если вы задумали какую-то кастомизацию, наверняка где-то отыщется готовый пример для нее. Рекомендую вам запустить Jupyter Notebook и поэкспериментировать с набором данных.

Мы увидели, как библиотека seaborn дополняет Matplotlib полезными статистическими методами и при этом, по мнению многих, ее графики очень эстетичны. Она также предоставляет доступ к объектам Figure и Axes из Matplotlib, что позволяет при необходимости выполнить полную кастомизацию.

В следующей главе мы применим Matplotlib вместе с pandas для анализа недавно собранных и очищенных данных о нобелевских лауреатах. Мы будем использовать некоторые из типов графиков, рассмотренных в этой главе, и познакомимся с новыми.

Анализ данных с помощью pandas

В предыдущей главе мы очистили набор данных о нобелевских лауреатах, собранный из Википедии в главе 6. Пришла пора взяться за анализ нашего чистейшего набора данных, искать любопытные закономерности и все, что может лечь в основу интересной визуализации.

Для начала освободим сознание от предубеждений и внимательно изучим доступные данные, чтобы получить общее представление о возможных визуализациях. В примере 11-1 показана форма набора категориальных, темпоральных и географических данных о лауреатах Нобелевской премии.

Пример 11-1. *Очищенный набор данных о лауреатах Нобелевской премии*

```
[{
 'category': 'Physiology or Medicine',
 'date_of_birth': '8 October 1927',
 'date_of_death': '24 March 2002',
 'gender': 'male',
 'link': 'http://en.wikipedia.org/wiki/C%C3%A9sar_Milstein',
 'name': 'César Milstein'
 'country': 'Argentina',
 'place_of_birth': 'Bahía Blanca, Argentina',
 'place_of_death': 'Cambridge, England',
 'year': 1984,
 'born_in': NaN
},
...
]
```

Данные из примера 11.1 могут показать разные аспекты *истории* Нобелевской премии, например:

— диспропорция женщин и мужчин среди лауреатов;

— анализ по странам (например, в какой стране больше всего лауреатов в области экономики);

— подробности о лауреатах: средний возраст на момент вручения премии или продолжительность жизни;
— миграция из родных стран в страны, предоставившие гражданство (с помощью полей `born_in` и `country`).

Эти направления исследования формируют основу для следующих подразделов, в которых мы проведем анализ набора данных с помощью вопросов, например: «Сколько женщин, помимо Марии Кюри, получили Нобелевскую премию по физике?», «В каких странах больше всего премий на душу населения, а не в абсолютном выражении?» и «Существует ли историческая тенденция в распределении премий по странам, передача первенства от старого научного мира (крупные европейские страны) к новому (США и развивающиеся азиатские страны)?» Перед началом исследования подготовим нужные инструменты и загрузим набор данных о Нобелевской премии.

Начало исследования

Чтобы приступить к исследованию, запустим Jupyter Notebook из командной строки:

```
$ jupyter notebook
```

Используем *магическую* команду `matplotlib` для создания встроенных графиков:

```
%matplotlib inline
```

Импортируем стандартный набор модулей для исследования данных:

```
import pandas as pd import numpy as np
import matplotlib.pyplot as plt import json
import matplotlib import seaborn as sns
```

Внесем несколько изменений в параметры построения графиков и общий вид диаграмм. *Прежде*, чем настраивать размеры рисунков, шрифты и др., необходимо изменить стиль:

SKLADCHIK.ORG

```
matplotlib.style.use('seaborn') ❶

plt.rcParams['figure.figsize'] = (8, 4) ❷
plt.rcParams['font.size'] = '14'
```

❶ Для оформления диаграмм используем встроенный в seaborn стиль, который считается более эстетичным, чем стандартный стиль Matplotlib.

❷ Устанавливаем размер области визуализации по умолчанию 8 × 4 дюйма.

В конце главы 9 мы сохранили набор очищенных данных в файл JSON. Теперь загрузим чистые данные в pandas DataFrame — и будем готовы приступить к исследованию.

```
df = pd.read_json(open('data/nobel_winners_cleaned.json'))
```

Получим основную информацию о структуре набора данных:

```
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 858 entries, 0 to 857
Data columns (total 13 columns):
#    Column          Non-Null Count   Dtype
---  ------          --------------   -----
0    category        858  non-null    object
1    country         858  non-null    object
2    date_of_birth   858  non-null    object
3    date_of_death   559  non-null    object
4    gender          858  non-null    object
5    link            858  non-null    object
6    name            858  non-null    object
7    place_of_birth  831  non-null    object
8    place_of_death  524  non-null    object
9    text            858  non-null    object
10   year            858  non-null    int64
11   award_age       858  non-null    int64
12   born_in         102  non-null    object
13   bio_image       770  non-null    object
14   mini_bio        857  non-null    object
dtypes: int64(2), object(13)
memory usage: 100.7+ KB
```

Обратите внимание, что столбцы date_of_birth и date_of_death имеют стандартный для pandas тип данных `object`. Поэтому нам нужно преобразовать их в тип `datetime64`, чтобы сравнивать даты. Используем для преобразования метод pandas `to_datetime`:

```
df.date_of_birth = pd.to_datetime(df.date_of_birth)
df.date_of_death = pd.to_datetime(df.date_of_death)
```

Запустив `df.info()`, мы должны увидеть эти два столбца с типом datetime:

```
df.info()

...
date_of_birth 858 non-null datetime64[ns, UTC] ❶
date_of_death 559 non-null datetime64[ns, UTC]
...
```

❶ Всемирное координированное время UTC (Coordinated Universal Time) — универсальный мировой стандарт времени. Практически всегда желательно придерживаться этого стандарта.

Метод `to_datetime` обычно работает без дополнительных аргументов и должен вызывает ошибку при некорректных данных, но рекомендуется проверить преобразованные столбцы. В нашем случае проверка прошла успешно.

Построение графиков с помощью pandas

Как Series, так и DataFrame в pandas имеют встроенные методы построения графиков, которые оборачивают самые распространенные диаграммы Matplotlib (некоторые из них мы рассмотрели в предыдущей главе). Это позволяет получать мгновенную визуальную обратную связь при взаимодействии с DataFrame. При визуализации чего-то более сложного, контейнеры pandas будут совместимы со стандартной Matplotlib. Вы также можете использовать стандартные настройки Matplotlib, чтобы адаптировать графики, созданные с помощью pandas.

Рассмотрим пример встроенной визуализации pandas. Начнем с базового графика диспропорции полов среди нобелевских лауреатов. Как известно, Нобелевская премия распределяется между полами неравномерно. Проанализируем это неравенство с применением столбчатой диаграммы по категории *gender*.

Результат выполнения кода из примера 11-2 показан на рисунке 11.1. Видна огромная разница: 811 лауреатов-мужчин из общего количества 858.

Пример 11-2. *Построение графика, отражающего гендерные диспропорции, с помощью интегрированных методов pandas*

```
by_gender = df.groupby('gender')
by_gender.size().plot(kind='bar')
```

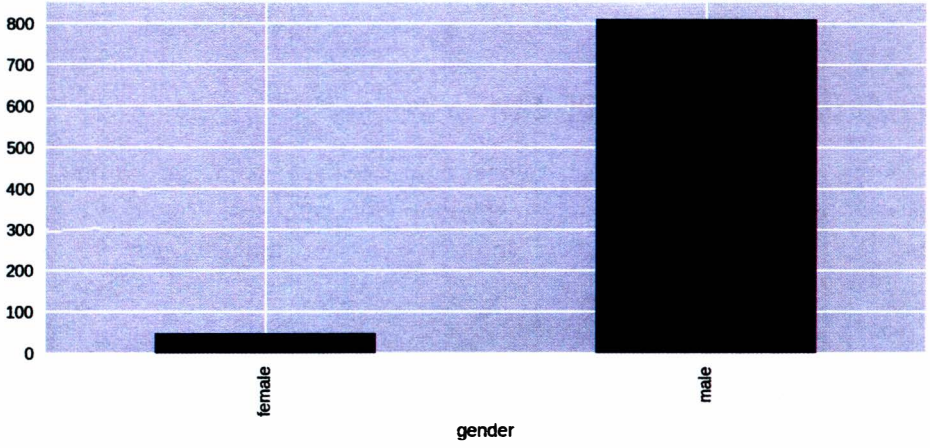


Рис. 11.1. *Распределение премий с учетом пола*

В примере 11-2 с помощью метода `size` создается Series для группы gender. Series имеет свой интегрированный метод `plot`, который превращает числа, содержащиеся в строке, в график.

```
by_gender.size()
Out:
gender
female   47
male     811
dtype: int64
```

Кроме линейных графиков по умолчанию, аргумент `kind` метода pandas `plot` позволяет выбирать другие доступные типы графиков. Из них чаще всего используются:

— `bar` или `barh` (*h* означает горизонтальный) для столбчатых диаграмм;

— `hist` для гистограмм;

— box для блочных диаграмм;
— scatter для диаграмм рассеивания.

Полный перечень интегрированных графиков pandas, а также некоторые функции для построения графиков, принимающие в качестве аргументов DataFrame и Series, см. в документации https://pandas.pydata.org/pandas-docs/stable/user_guide/visualization.html.

Продолжим исследование гендерных диспропорций и начнем расширять наши познания в построении графиков.

Гендерные диспропорции

Разнесем гендерные показатели, отображенные на рисунке 11.1, по номинациям премии. Метод groupby позволяет группировать данные по списку столбцов. Доступ к каждой группе осуществляется через комбинацию ключей:

```
by_cat_gen = df.groupby(['category','gender'])

by_cat_gen.get_group(('Physics', 'female'))[['name', 'year']] ❶
```

❶ Используя ключи category и gender, получаем группу:

```
Out:
      name                      year
269   Maria Goeppert-Mayer      1963
612   Marie Skłodowska-Curie    1903
```

Для получения размера этих групп используется метод size, который возвращает Series с MultiIndex, маркирующим значения как по номинации, так и по полу:

```
by_cat_gen.size()
Out:
category                gender
Chemistry               female   4
                        male     167
Economics               female   1
                        male     74
```

```
...
Physiology or Medicine    female    11
                          male      191
dtype: int64

dtype: int64
```

Мы можем построить график для Series с мультииндексом напрямую, указав аргумент `kind='hbar'` для построения горизонтальной столбчатой диаграммы. Результат работы кода показан на рисунке 11.2.

```
by_cat_gen.size().plot(kind='barh')
```

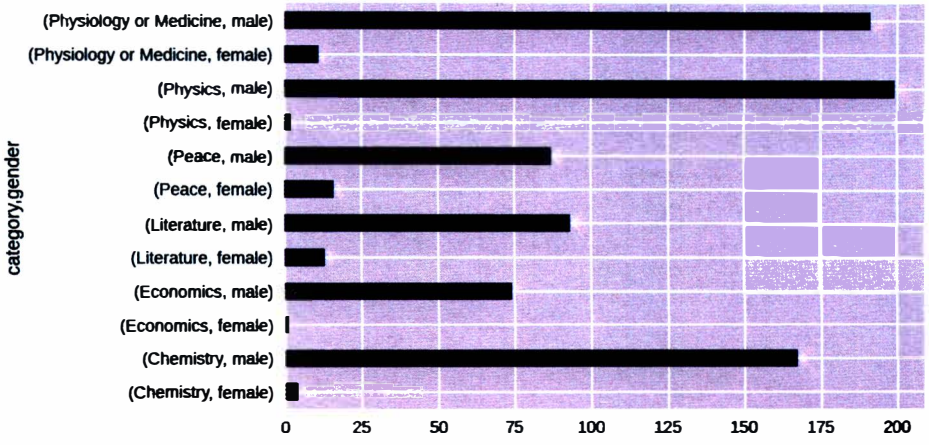


Рис. 11.2. *Визуализация групп с составным ключом*

Грубоватое исполнение рисунка 11.2 осложняет сравнительный анализ дисбаланса по полу. Доработаем диаграмму, чтобы сделать различия более наглядными.

Разбираем группы

Диаграмма на рисунке 11.2 не самая читабельная, даже если улучшить сортировку столбиков. Удобно, что Series в pandas имеют метод `unstack`, который преобразует несколько индексов (в данном случае gender и category) в столбцы и индексы, соответственно, при создании нового DataFrame. График, построенный по этому DataFrame, гораздо читабельнее, так как сравнивает лауреатов по полу. Код ниже создает рисунок 11.3:

```
by_cat_gen.size().unstack().plot(kind='barh')
```

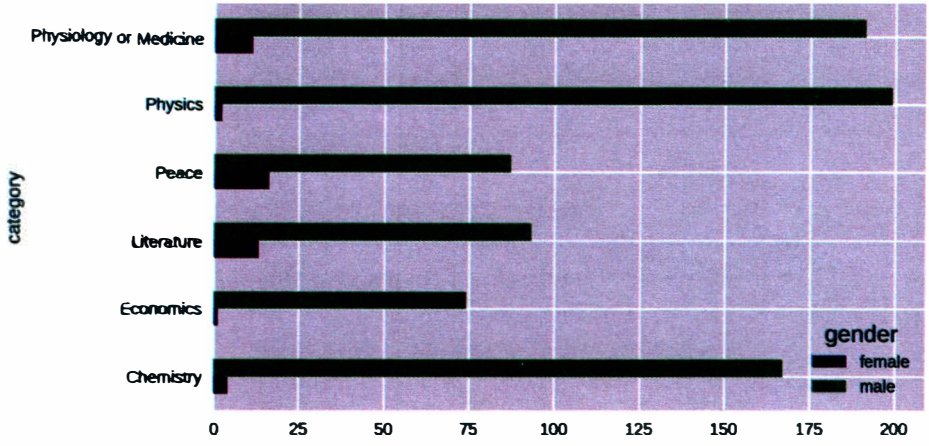


Рис. 11.3. *Развернутый Series с размерами групп*

На рисунке 11.3 видна огромная диспропорция между числом мужчин-лауреатов и женщин-лауреатов. Сделаем данные более конкретными, создав с помощью pandas диаграмму, показывающую процент лауреатов-женщин по номинациям и отсортируем номинации по общему количеству наград.

Сначала мы расформируем группу `by_cat_gen`, чтобы создать DataFrame `cat_gen_sz`:

```
cat_gen_sz = by_cat_gen.size().unstack()
cat_gen_sz.head()
gender        female   male
category
Chemistry     4        167
Economics     1        74
Literature    13       93
Peace         16       87
Physics       2        199
```

Для лучшего понимания выполним манипуляции с pandas в два этапа, сохранив новые данные в двух столбцах. Сначала создадим столбец, содержащий соотношение лауреатов-женщин к общему числу лауреатов.

```
cat_gen_sz['ratio'] = cat_gen_sz.female /\ ❶
                      (cat_gen_sz.female + cat_gen_sz.male)
cat_gen_sz.head()
```

❶ Неудобно расположенный прямой слэш предотвращает разрыв строки в Python, но здесь это оператор деления.

```
gender      female   male  ratio
category
Chemistry        4    167   0.023392
Economics        1     74   0.013333
Literature      13     93   0.122642
Peace           16     87   0.155340
Physics          2    199   0.009950
```

Соотношения из полученного столбца умножим на 100, результатом будет столбец, содержащий проценты лауреатов-женщин:

```
cat_gen_sz['female_pc'] = cat_gen_sz['ratio'] * 100
```

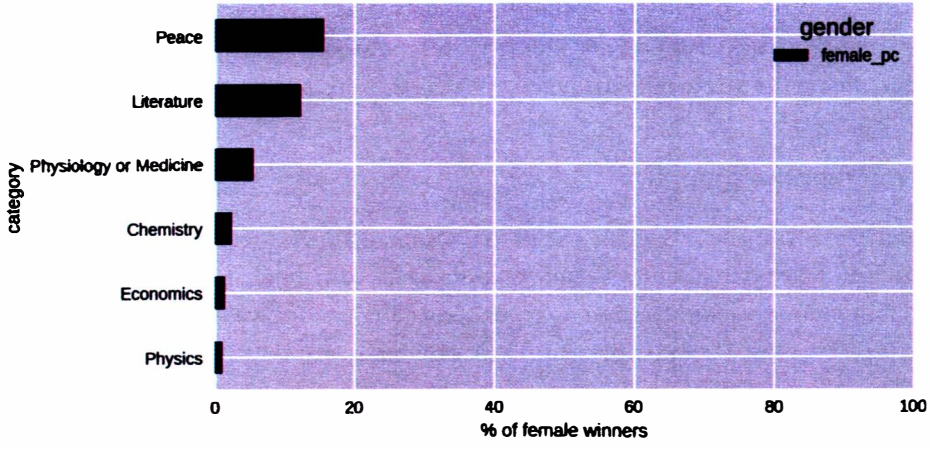


Рис. 11.4. *Процент лауреатов-женщин в каждой из номинаций премии*

Отобразим эти проценты на горизонтальной столбчатой диаграмме, установив предел по оси x равным 100 (%) и отсортировав категории по количеству лауреатов в номинации:

```
cat_gen_sz = cat_gen_sz.sort_values(by='female_pc', ascending=True)
ax = cat_gen_sz[['female_pc']].plot(kind='barh')
ax.set_xlim([0, 100])
ax.set_xlabel('% of female winners')
```

Новый график на рисунке 11.4 четко отражает дисбаланс между лауреатами мужчинами и женщинами.

Если исключить из рассмотрения экономику — недавнее и спорное дополнение к номинациям Нобелевской премии, — то на рисунке 11.4 видно, что наибольший гендерный дисбаланс среди лауреатов наблюдается в физике, где женщин всего две. Вспомним, кто они:

```
df[(df.category == 'Physics') & (df.gender == 'female')]\
   [['name', 'country','year']]
Out:
      name                      country          year
269   Maria Goeppert-Mayer      United States    1963
612   Marie Skłodowska-Curie    Poland           1903
```

Хотя Мария Кюри, одна из четырех дважды лауреатов Нобелевской премии, широко известна, о Марии Гепперт-Майер (Maria Goeppert-Mayer)[1] слышали немногие. Это тем более удивительно, учитывая современные усилия по вовлечению женщин в науку. Моя цель — создать визуализацию, которая позволит пользователям самостоятельно открыть для себя историю Марии Гепперт-Майер.

Исторические тренды

Было бы интересно узнать, увеличилось ли в последнее время количество женщин-лауреатов. Один из способов показать этот процесс — сгруппировать столбики по времени. Быстро построим график, используя функцию `unstack`, как и на рисунке 11.3, но применим ее к столбцам year и gender:

```
by_year_gender = df.groupby(['year','gender'])
year_gen_sz = by_year_gender.size().unstack()
year_gen_sz.plot(kind='bar', figsize=(16,4))
```

На рисунке 11.5 показан получившийся график, функциональный, но довольно нечитабельный. Тенденция в присуждении премии женщинам прослеживается, но у графика много проблем. Чтобы их исправить, воспользуемся замечательной гибкостью Matplotlib и pandas.

[1] Показательно, что никто из тех, кого я спрашивал, не знал имени второй женщины-лауреата Нобелевской премии по физике.

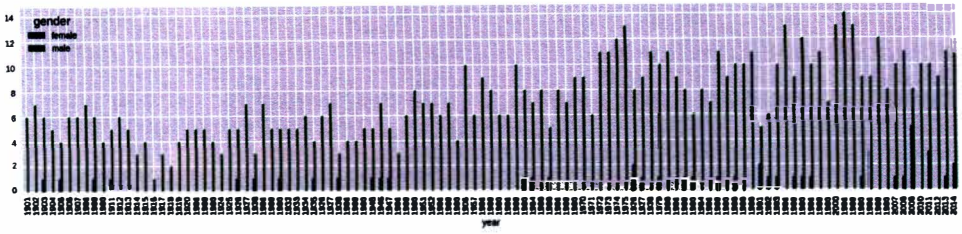


***Рис. 11.5.** Премии по годам с учетом пола*

Для начала уменьшим число меток на оси X. По умолчанию Matplotlib маркирует каждый столбик или группу столбиков столбчатой диаграммы, так что при более чем столетнем периоде метки практически наползают друг на друга. Уменьшить количество меток на осях с помощью Matplotlib можно разными способами. Я покажу, на мой взгляд, самый надежный. Поскольку такого рода действия приходится выполнять неоднократно, имеет смысл создать отдельную функцию. В примере 11-3 показана функция, которая уменьшает число делений на оси X.

***Пример 11-3.** Уменьшение числа меток по оси X*

```
def thin_xticks(ax, tick_gap=10, rotation=45):
    """ Прореживание делений X и настройка поворота """
    ticks = ax.xaxis.get_ticklocs() ❶
    ticklabels = [l.get_text()
                  for l in ax.xaxis.get_ticklabels()] ❶
    # Задать новое положение делений оси и меток в интервале
    # tick_gap (default +10+):
    ax.xaxis.set_ticks(ticks[::tick_gap])
    ax.xaxis.set_ticklabels(ticklabels[::tick_gap],
                            rotation=rotation) ❷
    ax.figure.show()
```

❶ Получаем существующее положение делений и меток на оси X, по одной на каждый столбик.

❷ Поворачиваем метки для удобства чтения, по умолчанию они были направлены по диагонали вверх.

Кроме необходимости сократить число делений, хорошо бы обозначить на оси X (на рисунке 11.5), что в годы Второй мировой войны 1939–1945 премия не присуждалась. Нам нужно вручную задать диапазон оси X, чтобы включить

все годы с начала существования Нобелевской премии по текущий день, за исключением перерыва в годы войны.

Текущие размеры извлеченных из стека групп используют автоматическую индексацию по годам:

```
by_year_gender = df.groupby(['year', 'gender'])
by_year_gender.size().unstack()
Out:
gender   female   male
year
1901     NaN      6.0
1902     NaN      7.0
...
2014     2.0      11.0
[111 rows x 2 columns]
```

Чтобы увидеть перерыв в распределении премии, требуется переиндексировать эту Series с помощью другой, содержащей полный диапазон лет:

```
new_index = pd.Index(np.arange(1901, 2015), name='year') ❶
by_year_gender = df.groupby(['year','gender'])
year_gen_sz = by_year_gender.size().unstack()
   .reindex(new_index) ❷
```

❶ Создаем индекс под названием year, охватывающий все годы вручения Нобелевской премии.

❷ Заменяем прерывающийся индекс новым, непрерывным.

Еще одна проблема диаграммы с рисунка 11.5 — чрезмерное количество столбиков. Хотя столбики, соответствующие мужчинам и женщинам, расположены рядом, их количество создает хаотичность и артефакты алиасинга. Лучше разнести диаграммы для мужчин и женщин и разместить их одну над другой, чтобы было проще сравнивать. Применим метод построения подграфиков, который мы рассматривали в «Осях и подграфиках», используя данные pandas, но настраивая график с использованием возможностей Matplotlib. В примере 11-4 показано, как это сделать, чтобы получить график с рисунка 11.6.

SKLADCHIK.ORG

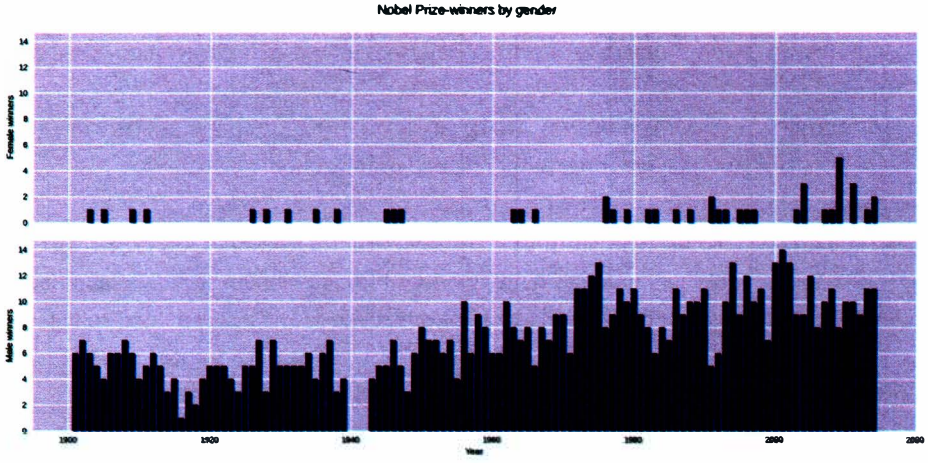


Рис. 11.6. Распределение премий на двух осях по годам и полу

***Пример 11-4.** Вертикальное расположение диаграмм распределения премий по полу и годам*

```
new_index = pd.Index(np.arange(1901, 2015), name='year')
by_year_gender = df.groupby(['year','gender'])

year_gen_sz = by_year_gender.size().unstack().reindex(new_index)

fig, axes = plt.subplots(nrows=2, ncols=1, ❶
           sharex=True, sharey=True, figsize=(16, 8)) ❷

ax_f = axes[0]
ax_m = axes[1]

fig.suptitle('Nobel Prize-winners by gender', fontsize=16)

ax_f.bar(year_gen_sz.index, year_gen_sz.female) ❸
ax_f.set_ylabel('Female winners')

ax_m.bar(year_gen_sz.index, year_gen_sz.male)
ax_m.set_ylabel('Male winners')

ax_m.set_xlabel('Year')
```

❶ Создаем две оси в сетке размером две (строки) на один (столбец).

❷ Используем общие оси X и Y, что обеспечивает корректное сравнение диаграмм.

❸ Применяем метод столбчатой диаграммы (`bar`) с непрерывным индексом year и данными, разделенными на два столбца по полу.

Вывод из нашего исследования распределения по полам: существует огромное расхождение, но, как показано на рисунке 11.6, в последние годы наблюдается некоторый прогресс. Более того, если экономика стоит особняком, то в физике и химии разница наиболее заметна. Поскольку лауреатов-женщин было мало, то больше здесь нечего показать.

Теперь рассмотрим национальные тенденции и попробуем найти какие-нибудь интересные моменты для визуализации.

Национальные тренды

Очевидная отправная точка при рассмотрении национальных трендов — построение графика абсолютного количества лауреатов по странам. Для этого достаточно одной строки кода, здесь разбитой надвое ради удобства чтения:

```
df.groupby('country').size().order(ascending=False)
      .plot(kind='bar', figsize=(12,4))
```

Полученная диаграмма (рисунок 11.7) показывает, что львиная доля лауреатов приходится на США.

Число премий на страну будет больше для стран с большей численностью населения. Давайте сделаем визуализацию, отражающую пересчет этого числа на душу населения.

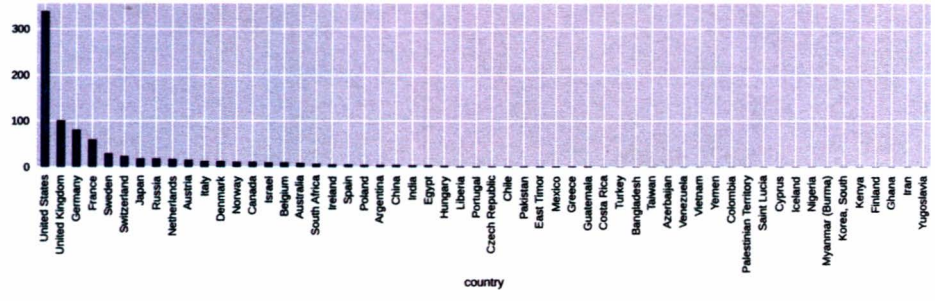


Рис. 11.7. Абсолютное число лауреатов по странам

Количество лауреатов на душу населения

Абсолютное число лауреатов неизбежно смещено в сторону крупных стран, отсюда вопрос: как изменятся показатели с учетом численности населения? Чтобы узнать долю премий, приходящуюся на душу населения, разделим их число на численность населения. В подразделе «Получение данных о странах для визуализации нобелевских лауреатов» на стр. 181, мы скачали веб-данные о странах и сохранили их в файле JSON. Возьмем его и построим график премии относительно численности населения.

Сначала получим размеры групп стран, используя названия стран как метки индекса:

```
nat_group = df.groupby('country')
ngsz = nat_group.size()
ngsz.index
Out:
Index([u'Argentina', u'Australia', u'Austria', u'Azerbaijan', ...])
```

Теперь загрузим данные стран в DataFrame и припомним, какие это данные:

```
df_countries = pd.read_json('data/winning_country_data.json',\
                            orient='index')

df_countries.loc['Japan'] # индекс по названию страны
Out:
gini            38.1
name            Japan
alpha3Code      JPN
area            377930.0
latlng          [36.0, 138.0]
capital         Tokyo
population      127080000
Name: Japan, dtype: object
```

Наш набор данных о странах уже проиндексирован столбцом `name`. Если мы добавим к нему `ngsz` (national group-size) Series, где также есть индекс по названию страны, то эти две Series будут объединены на основе общих индексов, что добавит к данным по стране новый столбец `nobel_wins`. Мы используем этот столбец (разделив его на численность населения), чтобы создать `nobel_wins_per_capita`:

```
df_countries = df_countries.set_index('name')
df_countries['nobel_wins'] = ngsz
df_countries['nobel_wins_per_capita'] =\
      df_countries.nobel_wins / df_countries.population
```

Осталось только отсортировать DataFrame `df_countries` по его новому столбцу `nobel_wins_per_cap` и построить график лауреатов Нобелевской премии в пересчете на душу населения, показанный на рисунке 11.8.

```
df.countries.sort_values(by='nobel_wins_per_capita',\
   ascending=False).nobel_per_capita.plot(kind='bar',\
   figsize=(12, 4))
```

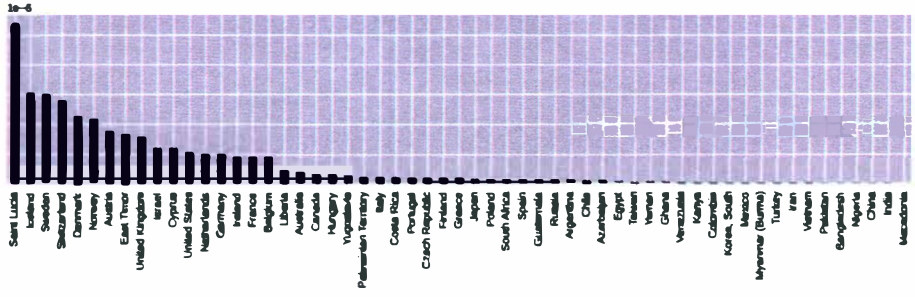


Рис. 11.8. *Показатель «количество лауреатов на душу населения»*

Диаграмма показывает, что первое место теперь занимает островное государство Сент-Люсия (в Карибском море). Благодаря небольшой численности населения — 175 000 человек — лауреат Нобелевской премии поэт Дерек Уолкотт Derek Walcott обеспечил для родной страны высший показатель Нобелевских премий на душу населения.

Посмотрим, как обстоят дела с показателем у более крупных стран, отфильтровав результаты по странам, где было более двух лауреатов Нобелевской премии:

```
df_countries[df_countries.nobel_wins > 2]\
      .sort_values(by='nobel_wins_per_capita', ascending=False)\
      .nobel_wins_per_capita.plot(kind='bar')
```

Результат на рисунке 11.9 показывает, что тут выше всех скандинавские страны и Швейцария.

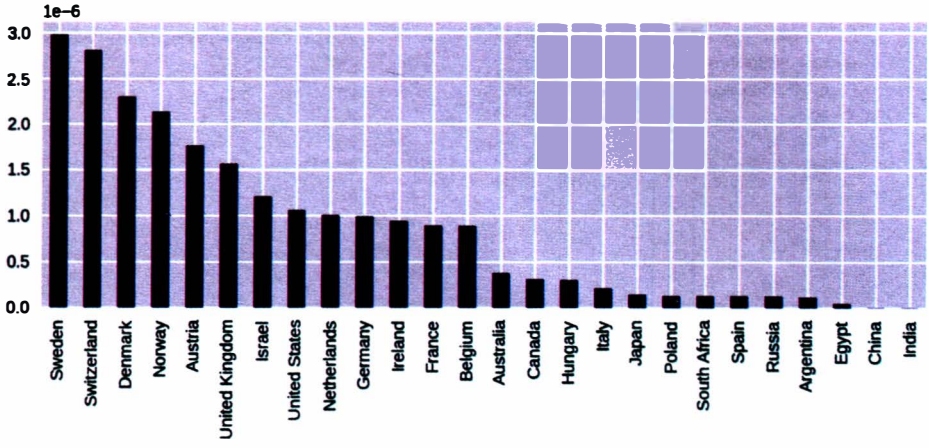


Рис. 11.9. *Количество национальных премий на душу населения, отфильтрованное по странам с тремя и более лауреатами*

Замена показателя лауреатов в целом для страны на показатель их количества на душу населения резко меняет картину. Немного доработаем поиск и сосредоточимся на номинациях премии, поищем там интересные закономерности.

Премии по номинациям

Возьмем абсолютные показатели лауреатов по странам и рассмотрим их применительно к номинациям премии. Для этого сгруппируем данные по столбцам country и category, получим размер этих групп, расформируем полученную Series, а затем построим график столбцов полученного DataFrame. Для начала получим номинации с размерами группы стран:

```
nat_cat_sz = df.groupby(['country', 'category']).size()
.unstack()
nat_cat_sz
Out:
category      Chemistry  Economics  Literature Peace \...
country
Argentina     1          NaN        NaN        2
Australia     NaN        1          1          NaN
Austria       3          1          1          2
Azerbaijan    NaN        NaN        NaN        NaN
Bangladesh    NaN        NaN        NaN        1
```

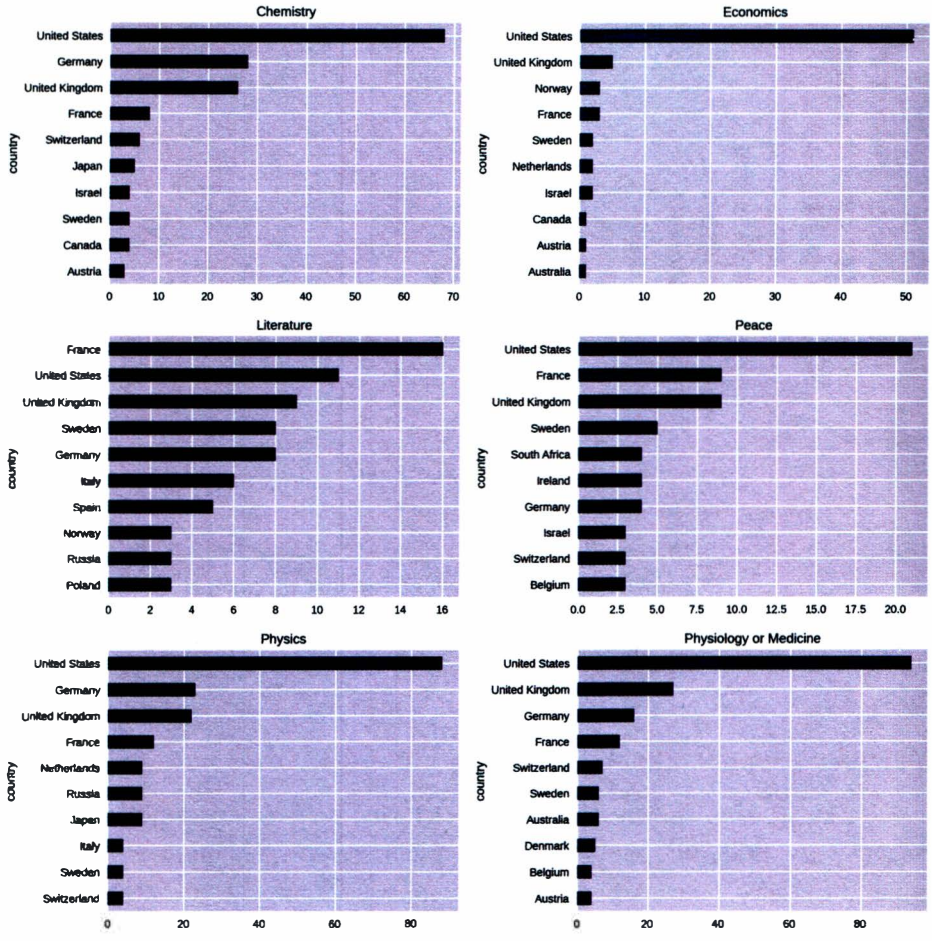


Рис. 11.10. *Премия по странам и номинациям*

Затем используем DataFrame `nat_cat_sz`, чтобы создать подграфики для шести номинаций Нобелевской премии:

```
COL_NUM = 2
ROW_NUM = 3

fig, axes = plt.subplots(ROW_NUM, COL_NUM, figsize=(12,12))

for i, (label, col) in enumerate(nat_cat_sz.items()): ❶
    ax = axes[i//COL_NUM, i%COL_NUM] ❷
    col = col.order(ascending=False)[:10] ❸
    col = col.sort_values(ascending=True) ❹
```

```
    col.plot(kind='barh', ax=ax)
    ax.set_title(label)

plt.tight_layout() ❺
```

❶ Метод `items` возвращает итератор для DataFrame в виде кортежей (column_label, column).

❷ В Python 3 появился удобный оператор целочисленного деления `//`, который возвращает целочисленный результат, отбросив дробную часть.

❸ `Order` упорядочивает Series столбца, предварительно сделав копию. Это эквивалентно записи `sort(inplace=False)`.

❹ Отобрав топ-10 стран, меняем порядок сортировки для построения горизонтальной столбчатой диаграммы, где наибольшие значения отображаются сверху.

❺ `tight_layout` предотвращает перекрытие меток между подграфиками. Если возникают проблемы из-за `tight_layout`, см. конец подраздела «Заголовки и метки осей» на стр. 320.

В результате получаем диаграммы, как на рисунке 11.10.

Несколько интересных фактов из рисунка 11.10 — доминирование США в номинации «Экономика», отражающее экономический консенсус после Второй мировой войны, и лидерство Франции в номинации «Литература».

Исторические тренды в распределении премии

После анализа статистических показателей по странам давайте исследуем историческую динамику распределения Нобелевской премии. Исследуем этот вопрос с помощью линейных графиков.

Для начала увеличим размер шрифта по умолчанию до 20 пунктов, чтобы подписи к графикам были более разборчивыми:

```
plt.rcParams['font.size'] = 20
```

Для анализа распределения премий по годам и странам нам потребуется новый неуложенный (unstacked) DataFrame, созданный на основе этих двух столбцов Как и ранее, добавляем `new_index` для указания периода с 1901 г. по 2015 г. без пропусков:

```
new_index = pd.Index(np.arange(1901, 2015), name='year')
```

```
by_year_nat_sz = df.groupby(['year', 'country'])\
   .size().unstack().reindex(new_index)
```

Нас интересует кривая изменения числа лауреатов Нобелевской премии по странам за всю ее историю. Далее мы можем более подробно изучить эти изменения в отдельных номинациях, но сейчас рассмотрим общую картину. У pandas как раз есть подходящий для этой цели метод `cumsum` (кумулятивной суммы). Возьмем столбец США и вычертим график.

```
by_year_nat_sz['United States'].cumsum().plot()
```

Получится график с рисунка 11.11.

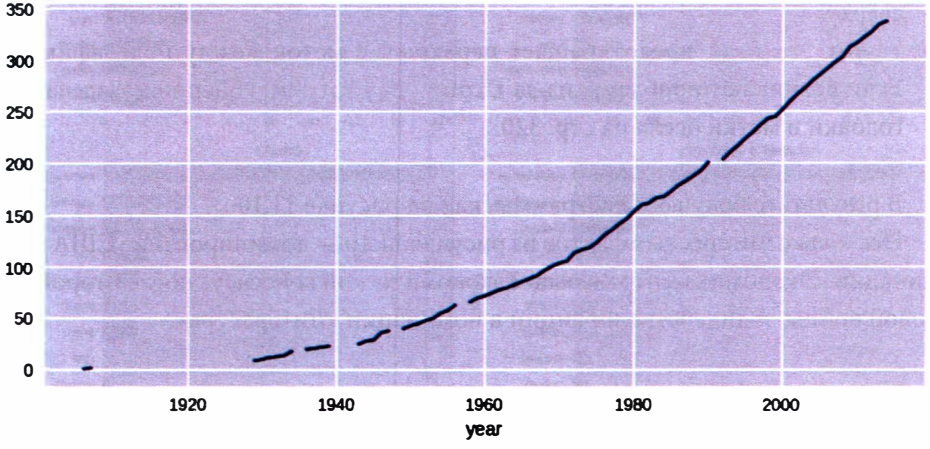


Рис. 11.11. *Кумулятивная сумма лауреатов из США по годам*

Разрывы на линии графика соответствуют полям `NaN` — годам, когда граждане США не удостаивались премии. В этих случаях метод `cumsum` возвращает `NaN`. Давайте заполним поля нулями, чтобы убрать разрывы:

```
by_year_nat_sz['United States'].fillna(0)
.cumsum().plot()
```

На рисунке 11.12 показана получившаяся более аккуратная линия графика.

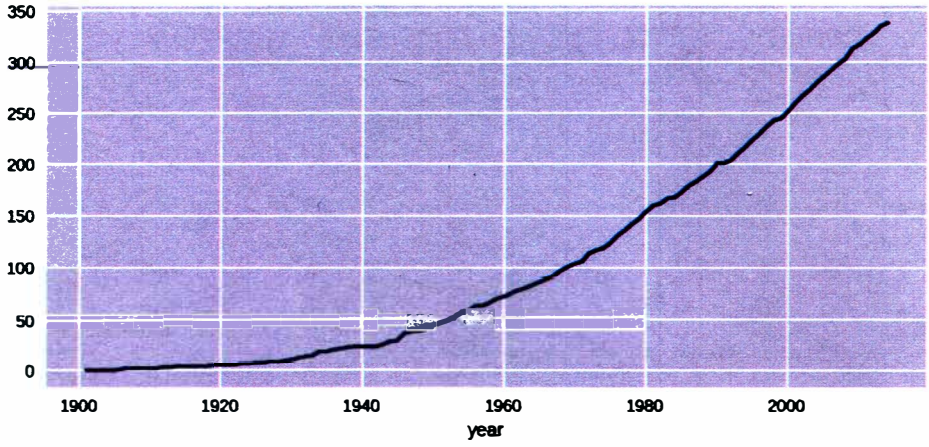


Рис. 11.12. *Кумулятивная сумма лауреатов из США по годам*

Теперь сравним показатели числа лауреатов из США и остального мира.

```
by_year_nat_sz = df.groupby(['year', 'country'])
      .size().unstack().fillna(0)

not_US = by_year_nat_sz.columns.tolist() ❶
not_US.remove('United States')

by_year_nat_sz['Not US'] = by_year_nat_sz[not_US].sum(axis=1) ❷
ax = by_year_nat_sz[['United States', 'Not US']]\
      .cumsum().plot(style=['-', '--']) ❸
```

❶ Получаем список столбцов названий стран и удаляем США.

❷ Используем список названий стран без США для создания столбца `'Not_US'`, в котором отображается общее число премий для стран из списка `not_US`.

❸ По умолчанию линии в графиках pandas цветные. Чтобы различать линии в черно-белой печатной версии, используем аргумент `style`, делая одну линию сплошной (-), а другую пунктирной (--) в соответствии со стилями Matplotlib (см. документацию).

Этот код создает график, показанный на рисунке 11.13.

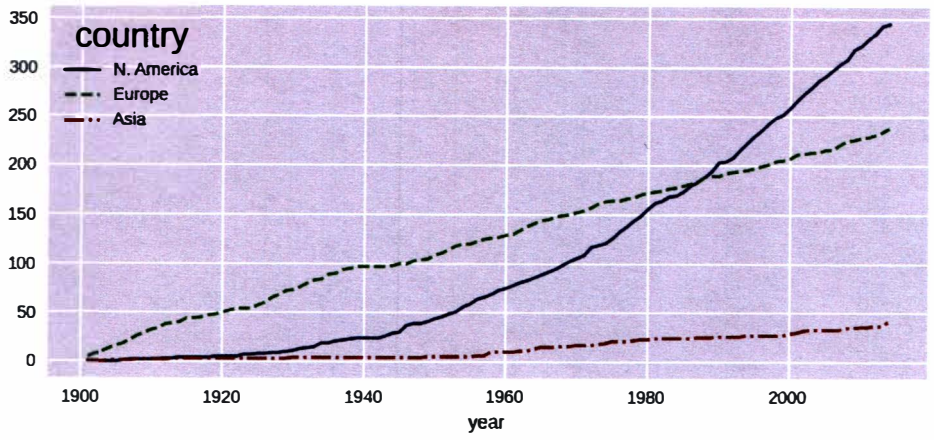


Рис. 11.13. *Сравнение динамики награждений: США против остального мира*

В то время как показатель `'Not_US'` демонстрирует стабильный рост за всю историю премии, то для США виден резкий скачок после окончания Второй мировой. Продолжим исследование: поищем региональные различия. Рассмотрим по две-три страны Северной Америки, Европы и Азии с наибольшим числом лауреатов

```
by_year_nat_sz = df.groupby(['year', 'country'])\
   .size().unstack().reindex(new_index).fillna(0)

regions = [ ❶
   {'label':'N. America',
    'countries':['United States', 'Canada']},
   {'label':'Europe',
    'countries':['United Kingdom', 'Germany', 'France']},
   {'label':'Asia',
    'countries':['Japan', 'Russia', 'India']}
]

for region in regions: ❷
   by_year_nat_sz[region['label']] =\
      by_year_nat_sz[region['countries']].sum(axis=1)

by_year_nat_sz[[r['label'] for r in regions]].cumsum()\
   .plot(style=['-', '--', '-.'])
   # стиль линий: сплошная, пунктирная, штрихпунктирная ❸
```

SKLADCHIK.ORG

❶ Для списка мы выбираем по две-три страны с наибольшим числом лауреатов из разных частей света.

❷ Создаем новый столбец с меткой региона для каждого словаря из списка `regions`, суммируя его элементы `countries`.

❸ Строим график кумулятивной суммы по столбцам регионов.

Получаем график, изображенный на рисунке 11.14. Темпы присуждения премий в Азии демонстрируют незначительный рост на протяжении всего периода, однако основной тренд связан с резким увеличением числа наград в Северной Америке около середины 1940-х годов, которая к середине 1980-х превзошла Европу по общему количеству премий на фоне снижения европейских показателей.

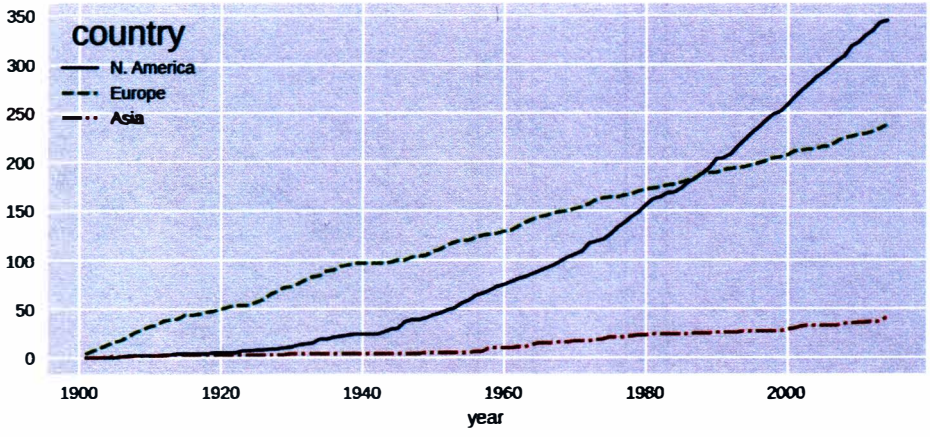


Рис. 11.14. *Историческая динамика присуждения премии по регионам*

Детализируем предыдущие национальные графики, просуммировав показатели присуждения премий для 16 стран с наивысшим числом лауреатов (за исключением Соединенных Штатов):

```
COL_NUM = 4
ROW_NUM = 4

by_nat_sz = df.groupby('country').size()
by_nat_sz.sort_values(ascending=False,\
   inplace=True) ❶

fig, axes = plt.subplots(COL_NUM, ROW_NUM,\ ❷
   sharex=True, sharey=True, ❷
   figsize=(12,12))
```

```
for i, nat in enumerate(by_nat.index[1:17]): ❸
    ax = axes[i/COL_NUM, i%ROW_NUM]
    by_year_nat_sz[nat].cumsum().plot(ax=ax) ❹
    ax.set_title(nat)
```

❶ Сортируем группы стран по количеству лауреатов: от большей к меньшей.

❷ Получаем сетку 4x4 с общими осями X и Y для сравнения нормализованных данных.

❸ Создаем кортежи с использованием отсортированного индекса, начиная со второй строки (1). США с индексом 0 исключается.

❹ Выбираем столбец с наименованием страны `nat` отображаем кумулятивную сумму ее премий на оси сетки `ax`.

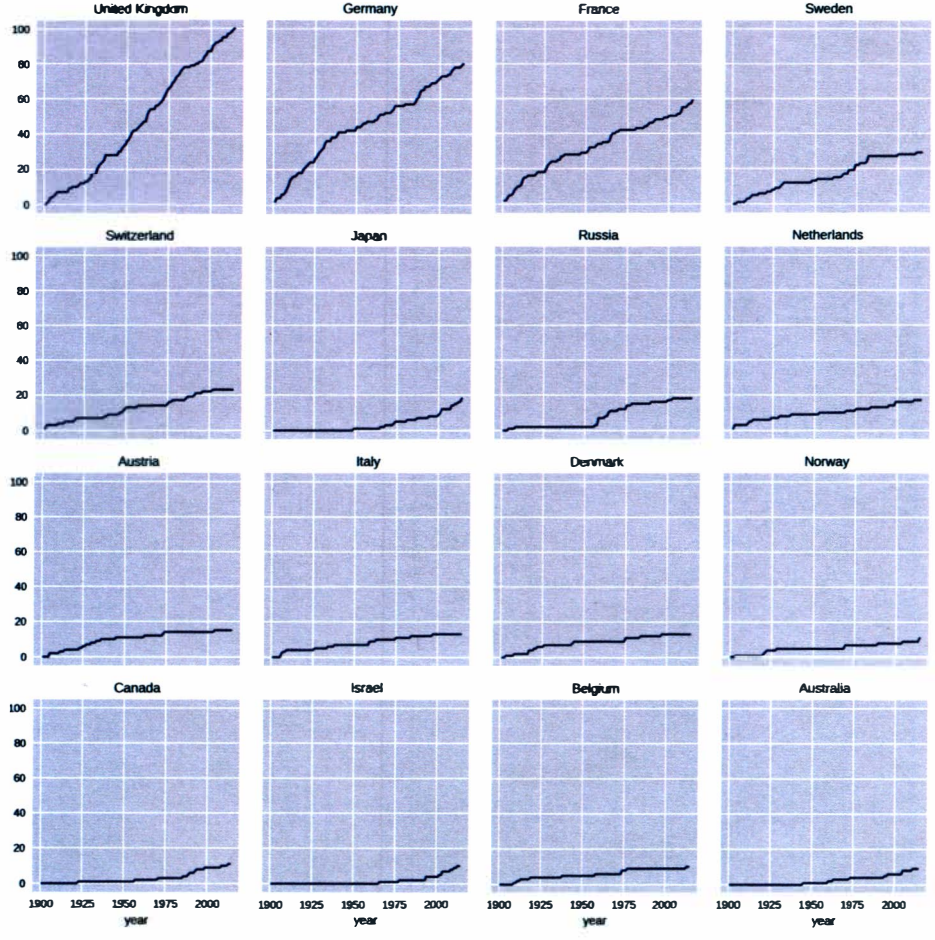


Рис. 11.15. *Доля премий для 16 стран с наибольшим (после США) количеством лауреатов*

Результат показан на рисунке 11.15. Мы видим, что такие страны, как Япония, Австралия и Израиль, демонстрируют исторический рост числа премий, тогда как у других показатели стабилизируются.

Еще один хороший способ оценивать уровень получения премий с течением времени по странам — использовать тепловую карту (heatmap) и разделить итоги по десятилетиям. В библиотеке seaborn отличная тепловая карта. Импортируем seaborn и используем метод `set` для масштабирования размера шрифта подписей:

```
import seaborn as sns

sns.set(font_scale = 1.3)
```

Разделение данных на фрагменты — бины — также известно как квантование (*binning*). В pandas есть удобный для этого метод `cut`, который принимает столбец непрерывных значений (в нашем случае — годы присуждения Нобелевской премии) и возвращает диапазоны указанного размера. Передав результат работы `cut` в `groupby` объекта DataFrame, мы получим группировку по диапазону проиндексированных значений. Следующий код создает рисунок 11.16:

```
bins = np.arange(df.year.min(), df.year.max(), 10) ❶

by_year_nat_binned = df.groupby('country',\
   [pd.cut(df.year, bins, precision=0)])\ ❷
   .size().unstack().fillna(0)

plt.figure(figsize=(8, 8))

sns.heatmap(\
   by_year_nat_binned[by_year_nat_binned.sum(axis=1) > 2],\ ❸
   cmap='rocket_r') ❹
```

❶ Получаем диапазоны бинов по десятилетиям, начиная с 1901 года (1901, 1911, 1921…).

❷ Преобразуем годы Нобелевской премии в десятилетние интервалы, используя диапазоны с точностью до целых `precision=0`, чтобы получать целочисленные годы.

❸ Перед составлением тепловой карты отфильтровываем страны, где было более двух лауреатов Нобелевской премии.

❹ Используем непрерывную тепловую карту `rocket_r`, подсвечивающую различия. Все цветовые палитры pandas см. в документации seaborn.

На рисунке 11.16 подмечены интересные тенденции, например, кратковременный рост числа премий для России в 1950-х годах, который пошел на спад примерно в 1980-х.

После исследования разных аспектов получения Нобелевской премии по странам уделим внимание отдельным лауреатам. Можно ли найти что-то интересное о них в имеющихся данных?

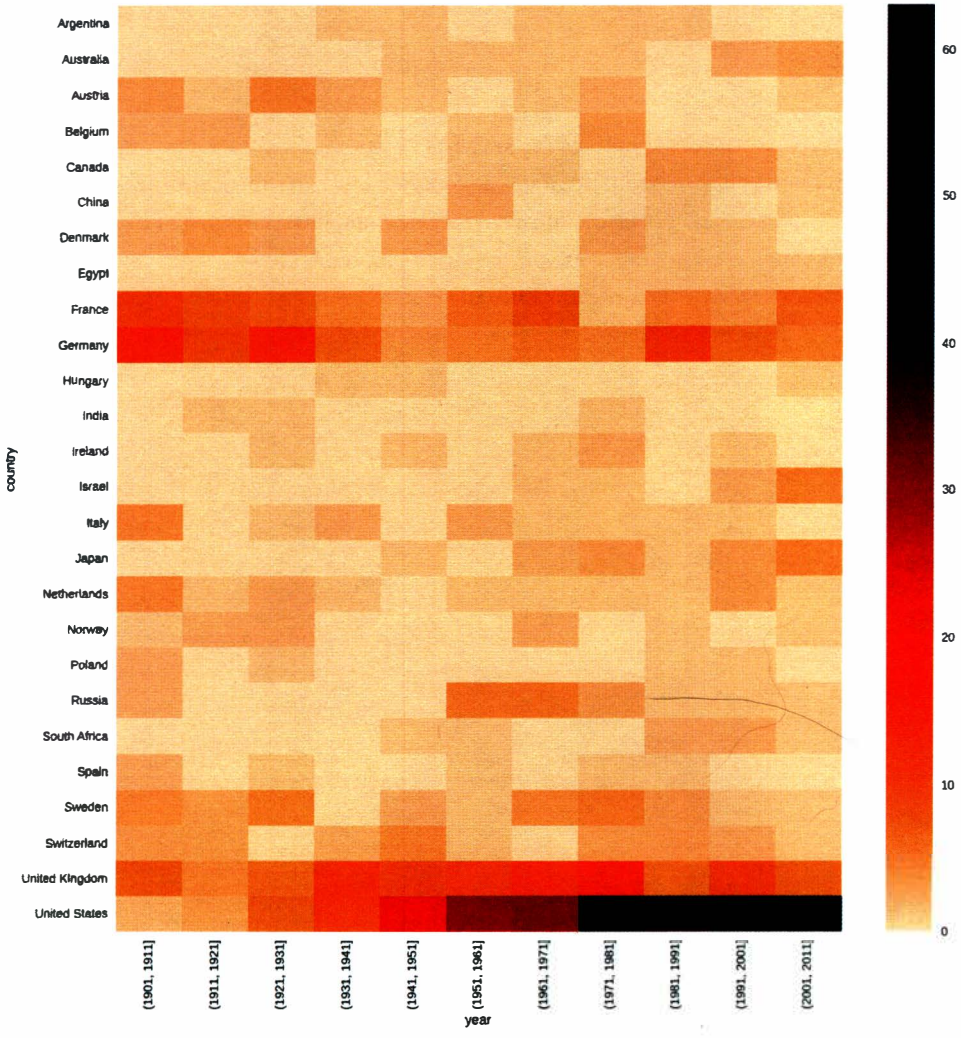


Рис. 11.16. *Выборка «Нобелевская премия по странам и десятилетиям»*

Возраст и ожидаемая продолжительность жизни лауреатов

Мы знаем даты рождения всех лауреатов и даты смерти 559 из них, а также годы получения премии. У нас достаточно персональных данных для наших изысканий. Исследуем распределение лауреатов по возрастам и попытаемся составить представление о продолжительности их жизни.

Возраст на момент получения премии

В главе 9 мы вычли возраст лауреатов из года получения ими Нобелевской премии и добавили к набору данных столбец `'award_age'`. Быстрый и простой способ оценки распределения — использование гистограммы pandas:

```
df['award_age'].hist(bins=20)
```

Здесь мы делим ось X на 20 дискретных интервалов. На рисунке 11.17 показано, что пик присуждения премий приходится на начало шестого десятка лет, а вероятность получения награды после 100 лет практически равна нулю. Обратите внимание на выброс в районе 20 лет — это отмечена 17-летняя лауреатка Премии мира Малала Юсуфзай.

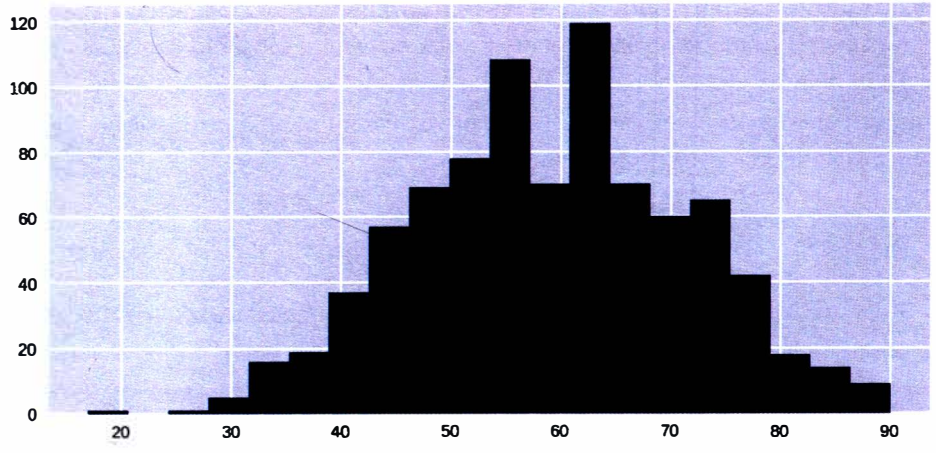


Рис. 11.17. *Распределение по возрасту получения премии*

Чтобы лучше понять распределение, используем функцию seaborn `displot`, которая добавит к гистограмме ядерную оценку плотности (Kernel Density

Estimation, KDE)[1]. Следующий однострочный код создает рисунок 11.18, где показано, что возраст около 60 лет — золотая середина:

```
sns.displot(df['award_age'], kde=True, height=4, aspect=2)
```

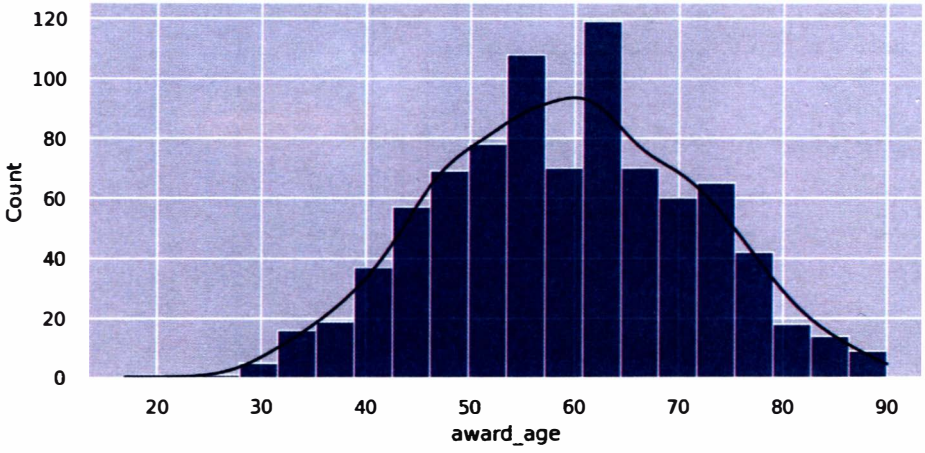


Рис. 11.18. *Распределение возрастов на момент награждения с наложенным KDE*

Блочная диаграмма (также «ящик с усами») хорошо подходит для визуализации непрерывных данных. Она показывает квартили, первый и третий из которых соответствуют границам «ящика», а второй (медиана) — горизонтальная черта внутри «ящика» — маркирует среднее значение. Обычно, как и на рисунке 11.19, концы «усов» (горизонтальные засечки) указывают на максимальное и минимальное значение данных. Используем блочную диаграмму seaborn и разделим количество премий по полу лауреатов:

```
sns.boxplot(df, x='gender', y='award_age')
```

Код создает диаграмму (рисунок 11.19), которая показывает, что распределения по полу похожи, хотя у женщин немного ниже средний возраст получения премии. Обратите внимание: статистические данные для женщин обладают значительно большей неопределенностью, что связано с меньшим количеством лауреатов женского пола.

[1] Подробнее см. Википедию. Фактически данные сглаживаются, и строится функция плотности вероятности.

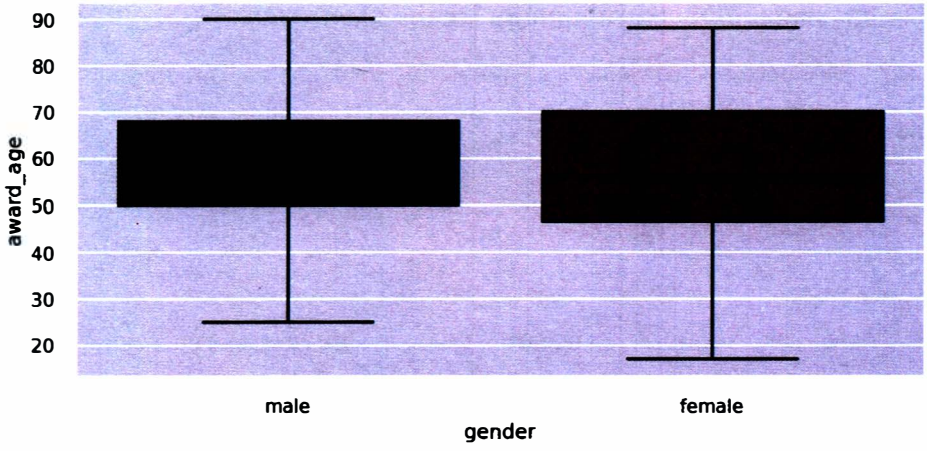


Рис. 11.19. *Распределение возраста получения премии по полу лауреатов*

Более точное представление о распределении по возрасту и полу может дать скрипичная диаграмма (violin plot) из библиотеки seaborn, сочетающая в себе блочную диаграмму с ядерной оценкой плотности. Диаграмму на рисунке 11.20 даст следующий код:

```
sns.violinplot(data=df, x='gender', y='award_age')
```

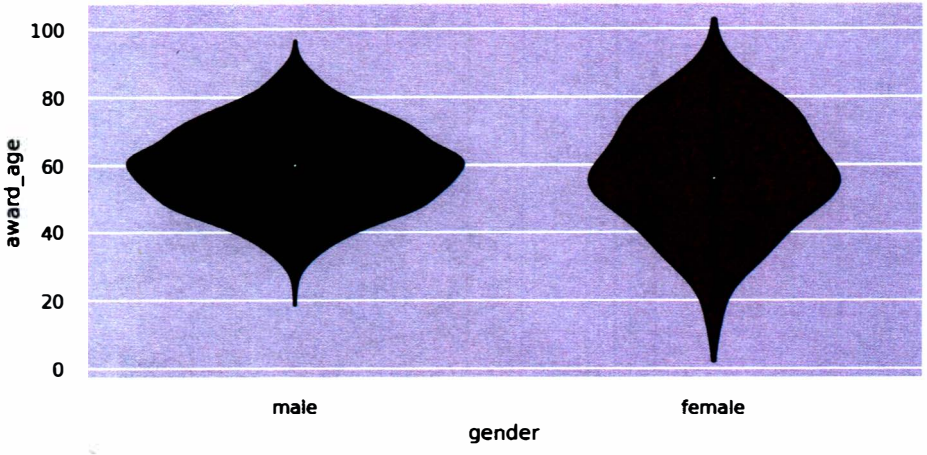


Рис. 11.20. *Скрипичные диаграммы распределения возраста получения премии по полу лауреатов*

Ожидаемая продолжительность жизни лауреатов

Рассчитаем продолжительность жизни лауреатов Нобелевской премии: вычтем даты рождения из соответствующих дат смерти. Сохраним результат в столбце `'age_at_death'`:

```
df['age_at_death'] = (df.date_of_death - df.date_of_birth)\
                     .dt.days/365 ❶
```

❶ Данные типа `datetime64` поддерживают арифметические операции, возвращая столбцы с типом pandas `Timedelta`. Используем метод `dt` для извлечения интервала в днях, затем делим на 365, чтобы получить возраст на момент смерти в формате float.

Сделаем копию столбца `'age_at_death'`[1], удалив все строки с `NaN`, и построим на этой основе гистограмму и KDE, показанные на рисунке 11.21:

```
age_at_death = df[df.age_at_death.notnull()].age_at_death ❶
sns.displot(age_at_death, bins=40, kde=True, aspect=2, height=4)
```

❶ Удаляем все `NaN` для очистки данных и предотвращения ошибок визуализации (например, `distplot` дает ошибку преобразования `NaN`).

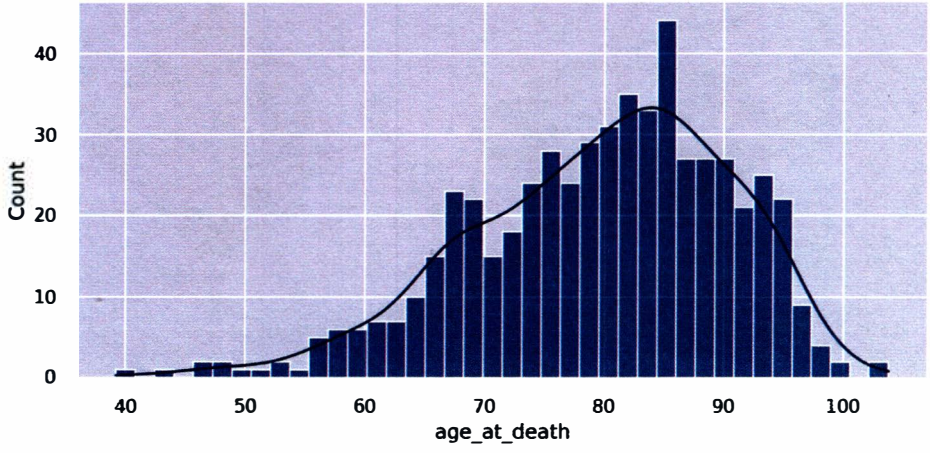


Рис. 11.21. *Ожидаемая продолжительность жизни лауреатов Нобелевской премии*

[1] При пересчете дней в годы мы игнорируем високосные годы и другие малозаметные усложняющие факторы.

Рисунок 11.21 показывает, что лауреаты Нобелевской премии — долгожители, их средний возраст около 80 лет. Это тем более удивительно, что подавляющее их большинство — мужчины, у которых средняя продолжительность жизни[1] в целом значительно ниже, чем у женщин. Один из факторов, способствующих такому долголетию, — смещение отбора, которое мы наблюдали ранее. Лауреаты обычно получают премию в возрасте 50–60 лет, что исключает из выборки тех, кто умер раньше этого возраста, искусственно завышая показатели долголетия.

На рисунке 11.21 видно, что среди лауреатов есть люди, дожившие до ста и более лет. Найдем их:

```
df[df.age_at_death > 100][['name', 'category', 'year']]
Out:
      name                  category                year
101   Ronald Coase          Economics               1991
328   Rita Levi-Montalcini  Physiology or Medicine  1986
```

Наложим две кривые KDE для сравнения продолжительности жизни лауреатов мужского и женского пола:

```
df_temp = df_temp[df.age_at_death.notnull()] ❶
sns.kdeplot(df_temp[df_temp.gender == 'male']
   .age_at_death, shade=True, label='male')
sns.kdeplot(df_temp[df_temp.gender == 'female']
   .age_at_death, shade=True, label='female')

plt.legend()
```

❶ Создаем DataFrame только с валидными полями `'age_at_death'`.

Как видно из рисунка 11.22, несмотря на малое количество женщин-лауреатов и более пологое распределение, средние значения для обоих полов близки. Жизнь нобелевских лауреатов-женщин, по-видимому, короче, чем у женщин в среднем.

[1] В зависимости от страны эта разница составляет 5–6 лет. Статистические данные см. в Our World in Data («Наш мир в цифрах»).



Рис. 11.22. *Ожидаемая продолжительность жизни лауреатов Нобелевской премии в зависимости от пола*

Скрипичная диаграмма на рисунке 11.23 показывает данные под другим углом:

```
sns.violinplot(data=df, x='gender', y='age_at_death',\
               aspect=2, height=4)
```

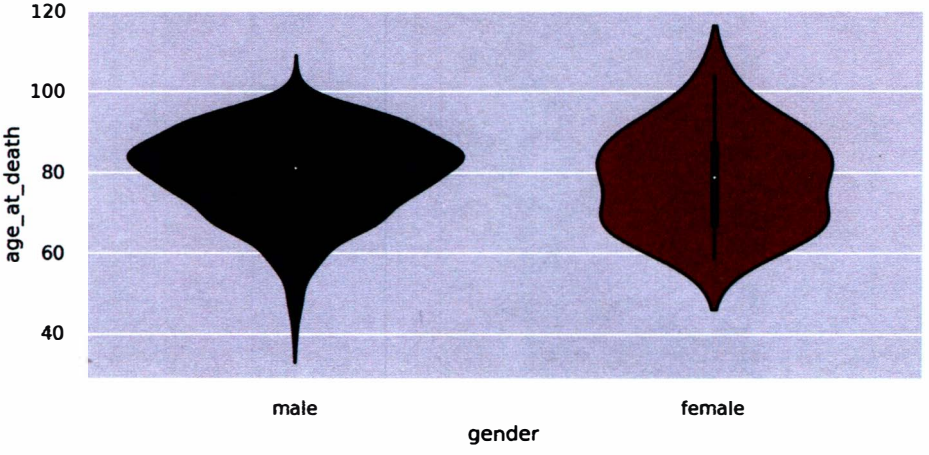


Рис. 11.23. *Ожидаемая продолжительность жизни лауреатов в зависимости от пола*

SKLADCHIK.ORG

Повышение продолжительности жизни с течением времени

Проведем небольшой историко-демографический анализ и выясним, есть ли взаимосвязь между датой рождения лауреатов Нобелевской премии и продолжительностью их жизни. Применим метод seaborn `lmplot`, чтобы получить диаграмму рассеивания и линейную регрессию с доверительными интервалами:

```
df_temp = df[df.age_at_death.notnull()] ❶
data = pd.DataFrame( ❷
   {'age at death': df_temp.age_at_death,
    'date of birth': df_temp.date_of_birth.dt.year})
sns.lmplot(data=data, x='date of birth', y='age at death',
   height=6, aspect=1.5)
```

❶ Создаем временный DataFrame, удалив все строки с отсутствующими значениями в поле `'age_at_death'`.

❷ Создаем новый DataFrame, содержащий только два целевых столбца из обработанного `df_temp`. Извлекаем из `date_of_birth` только год с помощью метода `.dt`.

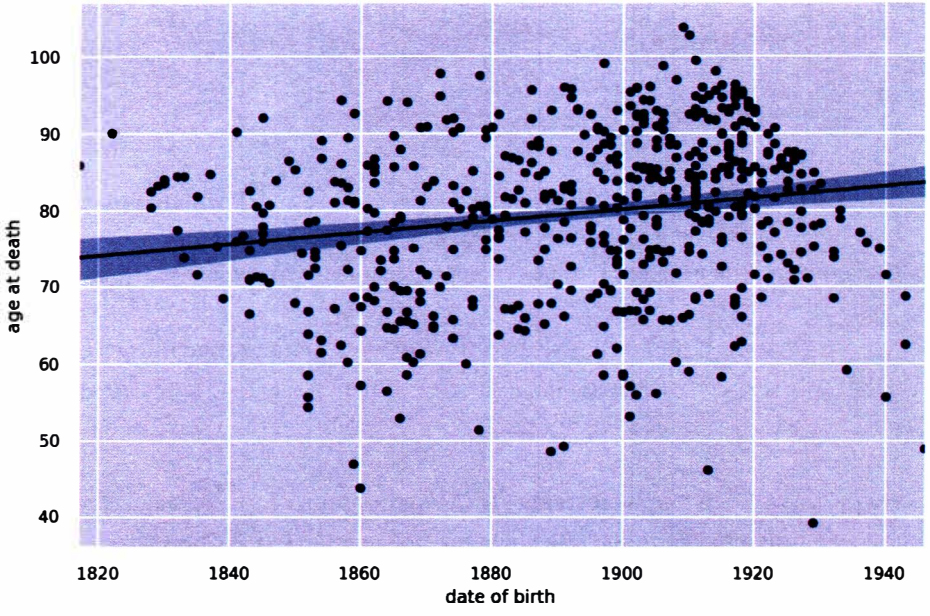


Рис. 11.24. *Корреляция даты рождения с продолжительностью жизни*

На рисунке 11.24 показано, что за время существования премии продолжительность жизни увеличилась примерно на 10 лет.

Нобелевская «диаспора»

В главе 9, очищая набор данных о лауреатах Нобелевской премии, мы обнаружили дублирующиеся записи, фиксирующие страну рождения и страну проживания на момент присуждения премии. Таких лауреатов оказалось 104, и мы сохранили их данные. Что они могут нам поведать?

Для визуализации миграционных паттернов лауреатов целесообразно использовать тепловую карту (heatmap), отображающую пары «страна рождения / страна проживания» (`born_in`/`country`). Код ниже создает тепловую карту, как на рисунке 11.25:

```
by_bornin_nat = df[df.born_in.notnull()].groupby(\ ❶
   ['born_in', 'country']).size().unstack()
by_bornin_nat.index.name = 'Born in' ❷
by_bornin_nat.columns.name = 'Moved to'
plt.figure(figsize=(12, 12))

ax = sns.heatmap(by_bornin_nat, vmin=0, vmax=8, cmap="crest",\ ❸
                 linewidth=0.5)
ax.set_title('The Nobel Diaspora')
```

❶ Выбираем все строки с заполненным полем `'born_in'` и группируем данные по этому полю и столбцу country.

❷ Для повышения информативности переименовываем индексы строк и столбцов.

❸ Хотя `heatmap` библиотеки seaborn автоматически устанавливает границы данных, мы вручную задаем параметры `vmin` и `vmax`, чтобы обеспечить корректное отображение всех ячеек.

На рисунке 11.25 видны интересные закономерности, отражающие историю преследований и предоставления убежищ. Во-первых, США стали страной, принявшей большинство переехавших лауреатов Нобелевской премии, за ними следует Великобритания. Обратите внимание, что в обоих случаях (за исключением притока из Канады) ученые в основном приехали из Германии. Следующая по величине группа переселенцев — из Италии, Венгрии и Австрии. Изучение

отдельных лиц в этих группах показывает, что большинство из них переехали в преддверии Второй мировой войны из-за фашистских режимов и усиления преследований еврейских меньшинств.

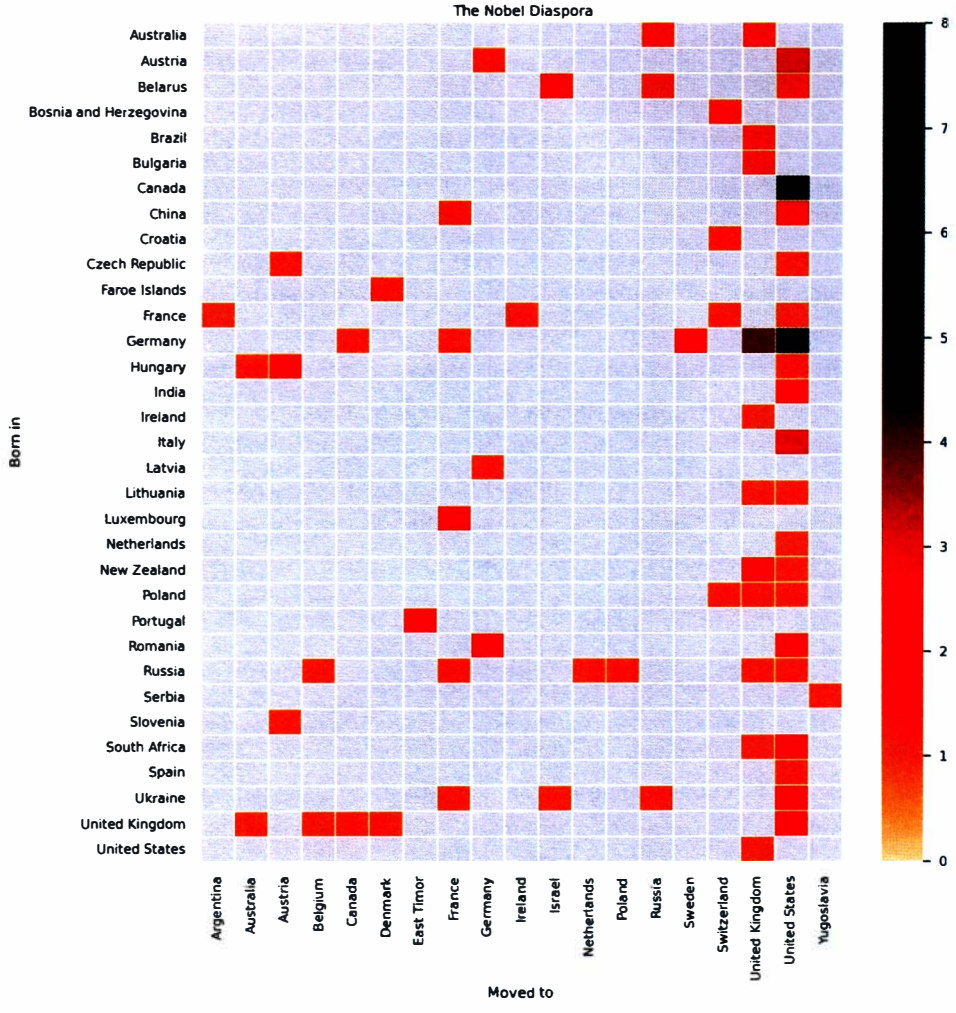


Рис. 11.25. *Нобелевская «диаспора»*

К примеру, все четыре лауреата Нобелевской премии, переехавшие из Германии в Великобританию, были учеными-исследователями еврейского происхождения, которые бежали, когда к власти пришли нацисты:

```
df[(df.born_in == 'Germany') & (df.country == 'United Kingdom')]
   [['name', 'date_of_birth', 'category']]
```

```
Out:
      name                date_of_birth    category
119   Ernst Boris Chain   1906-06-19       Physiology or Medicine
484   Hans Adolf Krebs    1900-08-25       Physiology or Medicine
486   Max Born            1882-12-11       Physics
503   Bernard Katz        1911-03-26       Physiology or Medicine
```

Эрнст Чейн (Ernst Chain) стоял у истоков промышленного производства пенициллина. Ганс Кребс (Hans Krebs) совершил одно из важнейших открытий в биохимии: последовательность рекреаций, известных как цикл Кребса, который регулирует выработку энергии клетками. Макс Борн (Max Born) — один из создателей квантовой механики, а Бернард Кац (Bernard Katz) открыл фундаментальные свойства синаптических связей в нейронах.

Среди лауреатов-эмигрантов есть немало прославленных имен. Еще один интересный факт — будущие лауреаты были в числе десяти тысяч еврейских детей из Германии, Австрии, Чехословакии и Польши, которых вывезли в Великобританию в ходе знаменитой операции «Киндертранспорт», организованной за девять месяцев до начала Второй мировой войны. Четверо из этих детей стали лауреатами Нобелевской премии.

Резюме

В этой главе мы исследовали набор данных о лауреатах Нобелевской премии. Проверили такие ключевые поля, как пол, номинация, страна и год (получения премии) в поисках интересных тенденций и историй, которые можно рассказать или воплотить в визуальной форме. Мы создали довольно много графиков Matplotlib (с помощью pandas) и seaborn: от столбчатых диаграмм до сложных статистических сводок данных, таких как скрипичные диаграммы и теплокарты. Освоение этих и других инструментов из арсенала Python для построения диаграмм позволит вам быстро оценивать наборы данных, что необходимо для создания их визуализаций. Мы нашли более чем достаточно аспектов, по которым можно строить веб-визуализацию. В следующей главе мы разработаем именно такую визуализацию о нобелевских лауреатах, включив в нее основные инсайты из этой главы.

РАЗДЕЛ IV

Передача данных

В этом разделе мы рассмотрим, как передать отобранный, очищенный и исследованный набор данных о нобелевских лауреатах в браузер, где JavaScript и D3 преобразуют их в увлекательную интерактивную визуализацию (см. рисунок IV-1).

Самое замечательное, что такой язык общего назначения, как Python, с одинаковой легкостью позволяет как создавать веб-сервер, используя всего несколько лаконичных строк, так и извлекать данные с помощью мощных библиотек обработки данных.

Ключевой серверный инструмент нашего тулчейна — Flask, легковесный, но мощный веб-фреймворк на Python. В главе 12 мы рассмотрим, как передавать статические данные (через системные файлы), а также динамические, чаще всего как выборки из базы данных с фильтрацией по параметрам запроса. В главе 13 мы рассмотрим, как две основанные на Flask библиотеки позволяют создать RESTful API с помощью нескольких строк кода на Python.

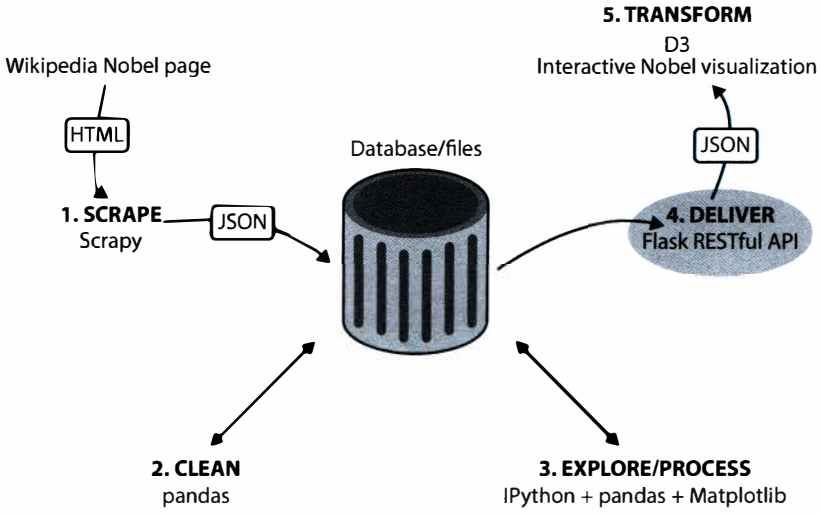


Рис. IV-1. *Передача данных*



Исходный код для этого раздела доступен в репозитории книги на GitHub.

Передача данных

В главе 6 мы разбирались, как захватить из интернета интересующие нас данные с помощью скрейпера. Мы получили набор данных о нобелевских лауреатах с помощью Scrapy, а затем в главах 9 и 11 с помощью pandas очистили и исследовали этот набор.

В этой главе на примере нашего набора мы рассмотрим, как доставить статические или динамические данные из Python-сервера в JavaScript в браузере клиента. Данные о лауреатах Нобелевской премии, сохраненные в формате JSON, состоят из списка объектов, подобных показанному в примере 12-1.

Пример 12-1. *Данные, полученные с помощью скрейпинга и затем очищенные:*

```
[
  {
  "category": "Physiology or Medicine",
  "country": "Argentina",
  "date_of_birth": "1927-10-08T00:00:00.000Z",
  "date_of_death": "2002-03-24T00:00:00.000Z",
  "gender": "male",
  "link": "http:\/\/en.wikipedia.org\/wiki\/C%C3%A9sar_Milstein",
  "name": "C\u00e9sar Milstein",
  "place_of_birth": "Bah\u00eda Blanca, Argentina",
  "place_of_death": "Cambridge, England",
  "text": "C\u00e9sar Milstein, Physiology or Medicine, 1984",
  "year": 1984,
  "award_age": 57,
  "born_in": "",
  "bio_image": "full/6bf65058d573e07b72231407842018afc98fd3ea.jpg",
  "mini_bio": "<p><b>César Milstein</b>, <a href='http://en.w..."
  }
  ['...']
]
```

Как и в остальной части этой книги, акцент будет сделан на минимизации объема веб-разработки, чтобы сосредоточиться на создании визуализации средствами JavaScript.



Совет бывалого: стремитесь выполнять как можно больше операций с данными средствами Python — аналогичные действия на JavaScript значительно сложнее. В соответствии с этим данные должны доставляться в форме, максимально приближенной к той, в которой они будут потребляться (для D3 это обычно массив объектов JSON, подобный созданному в главе 9).

Передача данных

Веб-сервер нам понадобится для обработки HTTP-запросов от браузера, передачи исходных статических HTML и CSS-файлов веб-страницы, а также любых последующих AJAX-запросов данных. Во время разработки веб-сервер обычно работает на порту локального хоста (в большинстве систем его IP-адрес 127.0.0.1). Традиционно для инициализации веб-сайта или, в нашем случае, одностраничного приложения (SPA), представляющего нашу веб-визуализацию, используется HTML-файл *index.html*.

Однострочные веб-серверы

При разработке или запуске демо-проектов, использующих статический контент, часто бывает полезно иметь простой веб-сервер для локальной доставки HTML-файлов, CSS, JavaScript и JSON в браузер. Веб-сервер обычно работает на порту 8000 или 8080. В Python предусмотрено встроенное решение — модуль `http.server`, который можно запустить из корневого каталога проекта командой:

```
$ python -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

Node.js предлагает альтернативный сервер для разработки — `http-server`. Его устанавливают через `npm install -g http-server` и запускают из корневого каталога проекта:

```
viz $ http-server
Starting up http-server, serving ./
...
Available on:
   http://127.0.0.1:8080
```

Однострочный веб-сервер приемлем для создания прототипов визуализации и набросков идей, но он не дает возможности управлять даже базовой функциональностью сервера — маршрутизацией URL или использованием динамических шаблонов. К счастью, в Python есть отличный небольшой веб-сервер, предоставляющий все функции, которые могут понадобиться для веб-визуализаций, и позволяющий при этом следовать к нашей цели — минимизировать boilerplate-код, стоящий между обработанными Python данными и высококлассной визуализацией на JavaScript. Миниатюрный веб-сервер Flask — достойное дополнение нашего лучшего в своем классе тулчейна.

Организация файлов Flask

Полезная информация о том, как организовать файлы проекта, редко встречается в учебных пособиях, возможно потому, что это вопрос личных предпочтений. Тем не менее, хорошая организация файлов может окупиться, особенно если работаешь в команде.

На рисунке 12.1 показано примерное расположение файлов при переходе от базового JavaScript-прототипа визуализации с меткой `basic` с использованием однострочного сервера через более сложный проект с меткой `basic+` к типичной простой настройке Flask с меткой `flask_project`.

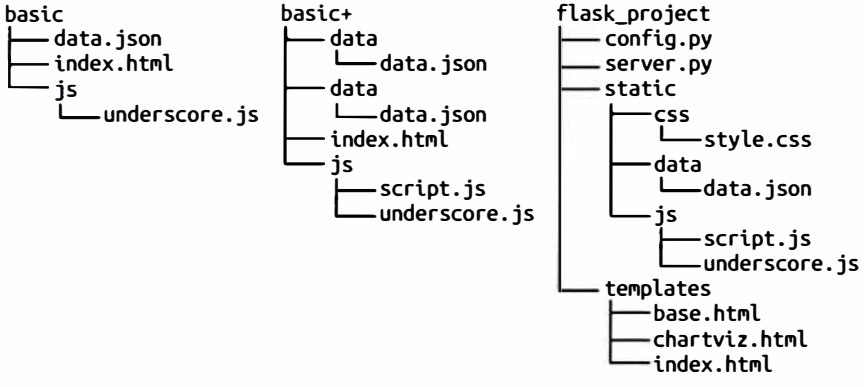


Рис. 12.1. *Организация файлов проекта сервера*

Главное в организации файлов — это последовательность. Она очень помогает хранить расположение файлов в процедурной памяти.

Передача данных с помощью Flask

Если вы используете пакеты Python Anaconda (см. главу 1), то Flask вам уже доступен. В противном случае инсталлируйте его с помощью `pip`:

```
$ pip install Flask
```

После добавления Flask настроим сервер с помощью нескольких строк, чтобы отобразить универсальное программное приветствие:

```
# server.py
from flask import Flask app = Flask(__name__)

@app.route("/") ❶
def hello():
   return "Hello World!"

if __name__== "__main__":
   app.run(port=8000, debug=True) ❷
```

❶ Направляем веб-трафик с помощью маршрутов Flask. Это корневой маршрут (то есть, *http://localhost:8000*).

❷ Задаем порт локального хоста для работы сервера (по умолчанию 5000). В режиме отладки Flask выводит на экран полезные данные из журнала, а в случае возникновения ошибки можно получить отчет в браузере.

Теперь перейдем в каталог, содержащий *nobel_viz.py*, и запустим модуль:

```
$ python server.py
   *  Serving Flask app 'server' (lazy loading)
   *  Environment: production
      WARNING: This is a development server.
      Do not use it in a production deployment.
      Use a production WSGI server instead.
   *  Debug mode: off
   *  Running on http://127.0.0.1:8000/ (Press CTRL+C to quit)
```

SKLADCHIK.ORG

Теперь вы можете перейти в свой браузер и увидеть результат, как на рисунке 12.2.



Рис. 12.2. *Простое сообщение, переданное в браузер*

Создание шаблонов с помощью Jinja2

По умолчанию Flask использует интуитивно понятную библиотеку шаблонов Jinja2, которая применяет переменные Python для настройки HTML-страницы. В коде ниже показан небольшой шаблон, который перебирает массив лауреатов для создания маркированного (неупорядоченного) списка:

```
<!— testj2.html —>
<! DOCTYPE html>
<meta charset="utf-8">

<body>
   <h2>\{{heading \}}</h2>
   <ul>
      {% for winner in winners %}
      <li><a  href="{{'http://wikipedia.com/wiki/'
            + winner.name}}">
         {{winner.name}}</a>
         {{', category: ' + winner.category}}
      </li>
      {% endfor %}
   </ul>
</body>
```

При использовании Jinja2 в сочетании с Flask обычно применяется метод `render_template`. Он создает HTML-ответ из шаблона, находящегося в каталоге *templates* проекта (по умолчанию). Любые аргументы, переданные в `render_template` после первой ссылки на файл шаблона, становятся доступными для этого шаблона. Итак, когда пользователь

посетит адрес */demolist*, следующий код с помощью *testj2.html* из каталога шаблонов нашего проекта отрендерит шаблон, создавая список, показанный на рисунке 12.3:

```
# server_jinja.py # …
app = Flask(__name__)
# ...

winners = [
   {'name': 'Albert Einstein', 'category':'Physics'},
   {'name': 'V.S. Naipaul', 'category':'Literature'},
   {'name': 'Dorothy Hodgkin', 'category':'Chemistry'}
]

@app.route('/winners')
def winners_list():
   return render_template('testj2.html',
                          heading="A little winners' list",
                          winners=winners
                          )
```

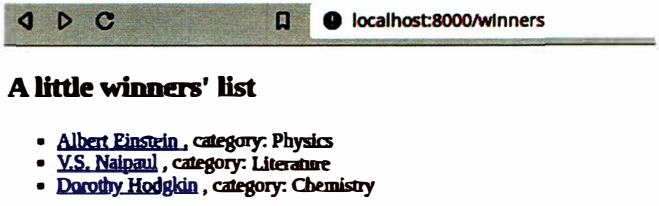


Рис. 12.3. *Список лауреатов — рендеринг шаблона testj2.html*

Jinja2 — мощный и продуманный язык шаблонов со всесторонней документацией, который позволяет уверенно использовать данные для рендеринга HTML-страниц на стороне сервера.

Как мы увидим в «Динамическом отображении данных с помощью Flask API» на стр. 396, сопоставление шаблонов в маршрутизации Flask упрощает развертывание простого web API. Также легко использовать шаблоны, чтобы генерировать динамические веб-страницы (см. рисунок 12.4). Шаблоны могут быть

полезны при визуализации для создания статических HTML-страниц на стороне сервера, но, как правило, на основе простого HTML-каркаса строят визуализацию с помощью JavaScript. При настройке визуализации на JavaScript основная задача сервера (помимо доставки статических файлов, необходимых для инициализации процесса) — динамическое взаимодействие с данными через AJAX-запросы браузера (обычно их предоставление).

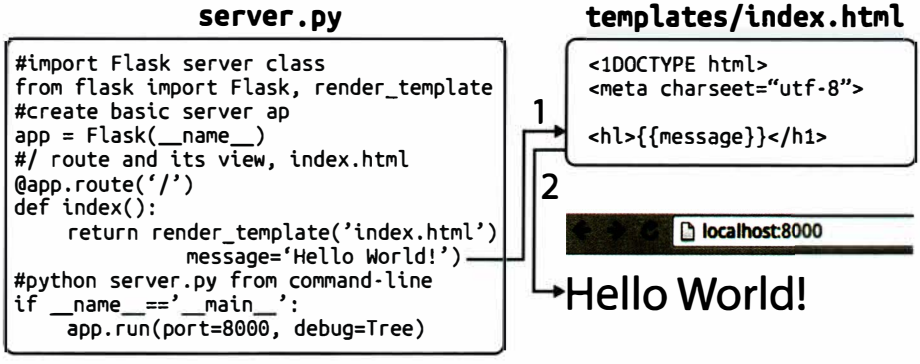


Рис. 12.4. *(1) Шаблон index.html используется для создания веб-страницы с помощью переменной message, которая затем (2) передается в браузер*

Flask прекрасно подходит для доставки полноценных веб-сайтов с мощной HTML-шаблонизацией, использованием объектов blueprint для модульности крупных проектов, поддержкой распространенных паттернов и множеством полезных плагинов. Хорошая отправная точка для изучения Flask — официальное руководство пользователя, а детали API описаны в соответствующем подразделе руководства. Одностраничные приложения, характерные для большинства веб-визуализаций, для доставки необходимых статических файлов не требуют особых премудростей на стороне сервера. Flask нас в основном интересует способностью предоставлять простые и эффективные серверы данных с надежным RESTful API, который можно создать с помощью нескольких строк кода на Python. Но давайте сначала рассмотрим, как доставлять и использовать данные в виде файлов JSON и CSV, а затем приступим к изучению API для работы с данными.

Доставка файлов данных

Многие веб-сайты, стремящиеся избежать накладных расходов на динамически конфигурируемые данные, предоставляют информацию в *статическом* формате. В этом случае все HTML-файлы и, что особенно важно, данные (обычно

в форматах JSON или CSV) хранятся в файловой системе сервера и готовы к передаче без дополнительных операций, таких как запросы к базе данных.

Статические страницы легко кешировать, поэтому их можно доставлять гораздо быстрее. Это также повышает уровень безопасности, поскольку запросы к базе данных — распространенный вектор атак злоумышленников (например, SQL-инъекции). Однако плата за повышение скорости и безопасности — потеря гибкости. Ограничение на использование заранее подготовленных страниц исключает взаимодействия с пользователем, требующие многовариантных комбинаций данных.

Для начинающего визуализатора данных использование статических данных имеет очевидные преимущества. Вы можете легко создать автономный проект без необходимости использования web API и распространять свою работу (в процессе разработки) в виде единой папки HTML, CSS и JSON.

Простейшие примеры веб-визуализаций со статическими файлами, управляемыми данными, можно найти в многочисленных примерах D3.js на *https://bl.ocks.org/mbostock*[1]. Их структура аналогична базовой странице, рассмотренной в подразделе «Базовая страница с плейсхолдерами» на стр. 142.

Хотя в этих примерах JavaScript и CSS встраиваются в HTML-страницу с помощью тегов `<script>` и `<style>`, я бы рекомендовал использовать отдельные CSS и JavaScript-файлы. Это позволяет применять редактор с поддержкой форматирования и упрощает отладку.

В примере 12-2 показана базовая страница *index.html* с плейсхолдерами данных `<h2>` и `<div>`, а также тегом `<script>`, который загружает локальный файл *script.js*. Мы встроили CSS в HTML-страницу, поскольку задали только стиль `font-family`. Мы создали следующую файловую структуру, где в подкаталоге *data* находится файл с набором данных *nobel_winners.json*:

```
viz
├── data
│   └── nobel_winners.json
├── index.html
└── script.js
```

Пример 12-2. *Базовая HTML-страница с плейсхолдерами*

```
<! DOCTYPE html>
<meta charset="utf-8">
```

[1] Создатель D3.js Майк Бостоk ценит и активно продвигает примеры. В этом выступлении он подробно раскрывает роль примеров в успехе D3.

```
<style>
   body{font-family: sans-serif;}
</style>

<h2 id='data-title'></h2>
<div id='data'>
   <pre></pre>
</div>
<script src="lib/d3.v7.min.js"></script>
<script src="script.js"></script>
```

Статический файл данных для этих примеров — единственный файл JSON (*nobel_winners.json*), находящийся в подкаталоге *data*. Для использования этих данных требуется отправить запрос AJAX на сервер. D3 содержит специализированные библиотеки для AJAX-вызовов, с методами для `json`, `csv` и `tsv`, более удобными для веб-визуализаторов.

В примере 12-3 показано, как загружать данные с помощью метода `d3.json`, используя коллбэк-функцию. Для отправки запросов и получения данных библиотека D3 использует под капотом JavaScript Fetch API. Он возвращает промис (Promise) JavaScript, который можно дополнить методом `then` — он вернет данные при отсутствии ошибок.

Пример 12-3. *Загрузка данных с помощью метода json из D3*

```
d3.json("data/nobel_winners_cleaned.json")
   .then((data) => {
   d3.select("h2#data-title").text("All the Nobel-winners");
d3.select("div#data pre").html(JSON.stringify(data, null, 4)); ❶
});
```

❶ Метод `JSON.stringify` в JavaScript — удобный способ форматирования объекта для вывода. В данном случае мы вставляем по 4 пробела для каждого отступа.

Если запустить однострочный сервер (например, `python -m http.server`) в каталоге *viz* и открыть страницу localhost в браузере, вы увидите результат, аналогичный рисунку 12.5, что подтверждает успешную передачу данных в JavaScript для визуализации.

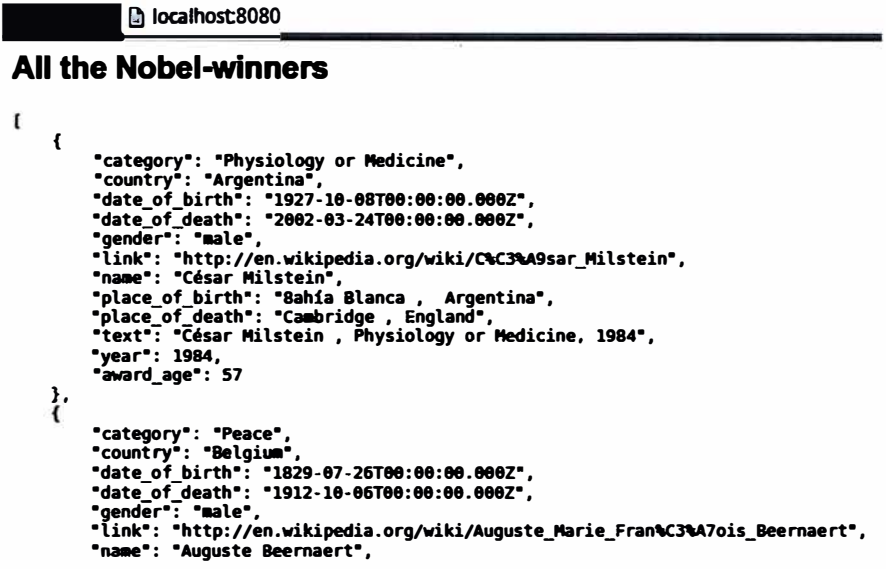


Рис. 12.5. Доставка JSON в браузер

Наш набор данных *nobel_winners.json* не особенно велик, но если добавлять в него текст биографий или другие текстовые данные, он может легко вырасти так, что превысит пропускную способность браузера и заставит пользователя нервничать в ожидании. Один из способов ограничить время загрузки — разбить данные на подмножества по одному из измерений. Очевидно, что в нашем случае нужно разбить лауреатов по странам. Несколько строк кода pandas создают подходящий каталог *data*:

```
import pandas as pd

df_winners = pd.read_json('data/nobel_winners.json')

for name, group in df_winners.groupby('country'): ❶
   group.to_json('data/winners_by_country' + name + '.json',\
                 orient='records')
```

❶ Сгруппируем DataFrame лауреатов по странам (`country`) и пройдем циклом по именам и элементам группы.

В результате получим подкаталог `winners_by_country` в каталоге *data:*

```
$ ls data/winners_by_country
```

```
Argentina.json    Azerbaijan.json        Canada.json
Colombia.json     Czech Republic.json    Egypt.json …
```

Теперь мы можем выборочно загружать данные по странам с помощью специально разработанной функции:

```
let loadCountryWinnersJSON = function (country) {
   d3.json("data/winners_by_country/" + country + ".json")
      .then(function (data) {
         d3.select("h2#data-title").text(
            "All the Nobel-winners from " + country
         );
         d3.select("div#data pre").html(JSON.stringify(data, null, 4));
      })
      .catch((error) => console.log(error));
};
```

Указанный ниже пример вызова функции загрузит всех нобелевских лауреатов из Австралии (результат см. на рисунке 12.6):

```
loadCountryWinnersJSON('Australia');
```

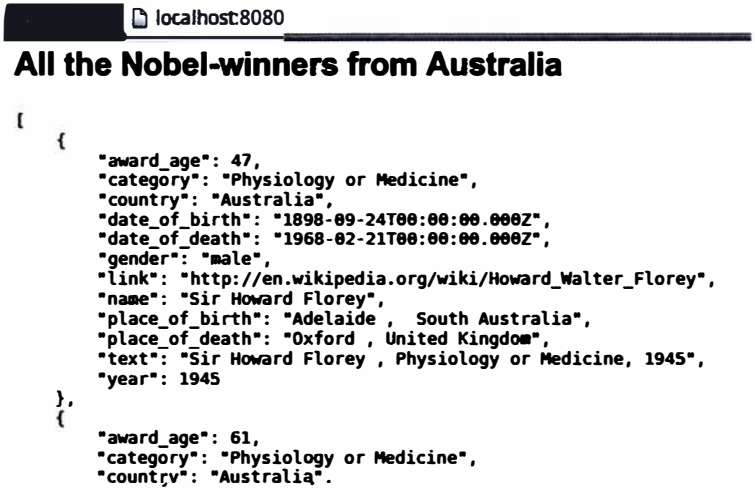


Рис. 12.6. *Отбор лауреатов по странам*

Возможность выбора лауреатов по странам могла бы снизить нагрузку на полосу пропускания данных и сократить обусловленную ею задержку, но что, если

потребуется выбрать лауреатов по году или полу? Если делить данные по всем измерениям (категориальному, временному и др.), для каждого потребуется отдельный подкаталог, что создаст беспорядок в файлах и во всех связанных с ними процессах. А что, если будут запросы на данные с более тонкой детализацией (например, обо всех лауреатах премий из США с 2000 года)? В этом случае необходим сервер данных, способный динамически реагировать на такие запросы, обычно появляющиеся при взаимодействии с пользователем. В следующем подразделе вы увидите, как создать такой сервер с помощью Flask.

Динамическое обновление данных с помощью Flask API

Доставка данных на веб-страницы через JSON или CSV-файлы лежит в основе многих впечатляющих веб-визуализаций и прекрасно подходит для создания прототипов и небольших демонстраций. Но у такого подхода есть ряд ограничений, наиболее очевидное — размер наборов данных, доставка которых осуществима. По мере увеличения наборов данных, когда размер файлов начинает исчисляться в мегабайтах, загрузка страниц замедляется, и одновременно растет разочарование пользователей. Для большинства задач визуализации данных (особенно дашбордов или исследовательских диаграмм) оптимальной стратегией становится доставка данных по требованию, инициированному через пользовательский интерфейс. Для такой доставки данных идеально подходит небольшой сервер данных, а Python-фреймворк Flask предоставляет все необходимые инструменты для его реализации.

Для динамической доставки данных требуется API, который позволит JavaScript их запрашивать.

Простой Data API на Flask

С помощью модуля Dataset мы можем легко адаптировать имеющийся у нас сервер к базе данных SQL. Здесь мы используем Dataset и специализированный JSON-энкодер (см. пример 3-2) для преобразования данных Python в JSON-совместимые строки ISO:

```
# server_sql.py
from flask
import Flask, request, abort
import dataset
```

```
import json
import datetime

app = Flask(__name__)
db = dataset.connect('sqlite:///data/nobel_winners.db')

@app.route('/api/winners')
def get_country_data():
    print 'Request args: ' + str(dict(request.args))
    query_dict = {}
    for key in ['country', 'category', 'year']: ❶
        arg = request.args.get(key) ❷
        if arg:
            query_dict[key] = arg

        winners = list(db['winners'].find(**query_dict)) ❸
        if winners:
            return dumps(winners)
        abort(404) # ресурс не найден

class JSONDateTimeEncoder(json.JSONEncoder): ❹
    def default(self, obj):
        if isinstance(obj, (datetime.date, datetime.datetime)):
            return obj.isoformat()
        else:
            return json.JSONEncoder.default(self, obj)

def dumps(obj):
    return json.dumps(obj, cls=JSONDateTimeEncoder)

if __name__ =='__main__': app.run(port=8000, debug=True)
```

❶ Ограничиваем запросы к базе данных ключами из этого списка.

❷ С помощью `request.args` получаем доступ к аргументам запроса (например, `'?coun try=Australia&category=Chemistry'`).

❸ Используя метод `find` запрашиваем словарь аргументов, который будет распакован с помощью ** (то есть, `find(country='Australia', category='Literature')`). Преобразуем итератор в список, готовый к сериализации.

❹ Это специализированный JSON-энкодер, показанный в примере 3-2.

После запуска сервера (`python server_sql.py`) мы можем протестировать наш небольшой API с помощью curl. Получим всех нобелевских лауреатов по физике из Японии:

```
$ curl -d category=Physics -d country=Japan
   --get http://localhost:8000/api/

[{"index": 761, "category": "Physics", "country": "Japan",
"date_of_birth": "1907-01-23T00:00:00",
"date_of_death": "1981-09-08T00:00:00", "gender": "male",
"link": "http://en.wikipedia.org/wiki/Hideki_Yukawa",
"name": "Hideki Yukawa", "place_of_birth": "Tokyo , Japan",
"place_of_death": "Kyoto , Japan",
"text": "Hideki Yukawa , Physics, 1949", "year": 1949,
"award_age": 42}, {"index": 762, "category": "Physics",
"country": "Japan", "date_of_birth": "1906-03-31T00:00:00",
"date_of_death": "1979-07-08T00:00:00", "gender": "male", ... }]
```

Теперь вы видите, как легко создать простой API. Его можно расширить множеством способов, но для быстрого прототипирования это решение оптимально.

Но как добавить пагинацию, аутентификацию и другие функции полноценного RESTful API? В следующей главе мы увидим, как улучшить наш API с помощью Python-библиотек вроде marmalade.

Использование динамической или статической доставки

Что предпочесть: статическую или динамическую доставку? Выбор во многом зависит от контекста и в любом случае приходится идти на компромисс. Пропускная способность варьируется в зависимости от региона и устройства. Например, если ваша визуализация должна быть доступна со смартфона в сельской местности, ограничения данных будут существенно отличаться от ограничений работающего в локальной сети внутреннего приложения для обработки данных.

Нужно прежде всего ориентироваться на пользовательский опыт. Если небольшое ожидание в начале, пока кешируются данные, компенсируется молниеносной JavaScript-визуализацией, то доставка статических данных — достаточно оптимальное решение. Если же пользователю позволено нарезать большой многомерный набор данных или делать его срезы, вряд ли получится обойтись

SKLADCHIK.ORG

без раздражающе долгого времени ожидания. Как показывает опыт, для любого набора данных менее 200 КБ подойдет доставка исключительно статического контента. Если объемы данных исчисляются, как минимум, в мегабайтах, скорее всего понадобится API на основе БД, откуда можно будет извлекать данные.

Резюме

В этой главе объяснены основы доставки статических файлов веб-сервером и динамической доставки данных, а также намечены контуры простого RESTful API на базе Flask. Хотя Flask позволяет быстро создать базовый API для данных, реализация пагинации, выборочных запросов и полного спектра HTTP-методов потребует дополнительных усилий. В первом издании я использовал готовые Python-библиотеки для создания RESTful API, но они быстро устаревают, вероятно, из-за того, что можно просто и с большей гибкостью собрать специализированные решения из отдельных Python-библиотек. К тому же это отличный способ изучить подобные инструменты, поэтому создание именно такого RESTful API — тема следующей главы.

RESTful Data с помощью Flask

В подразделе «Простой Data API с помощью Flask» на стр. 396 мы выяснили, как создать очень простой API с использованием Flask и модуля Dataset. Такой быстро созданный API подходит для простых визуализаций данных, но по мере усложнения требований к данным потребуется API, который соблюдает соглашения по получению, а иногда по созданию, обновлению и удалению[1]. В подразделе «Использование Python для получения данных через web API» на стр. 177 мы рассмотрели типы API и причины заслуженной популярности RESTful API[2]. В этой главе мы рассмотрим, как сконструировать из нескольких Flask-библиотек гибкий RESTful API.

Инструменты для работы с RESTful

Как показано в подразделе «Простой Data API с помощью Flask», требования data API довольно простые. Ему нужен сервер, который обрабатывает HTTP-запросы: GET для получения данных, POST (для добавления) или DELETE. Эти пути запроса (типа `api/winners`) затем обрабатываются соответствующими функциями. Функции извлекают данные из БД бэкенда, по необходимости отфильтрованные по параметрам запроса (например, строк вида `?category=comic&name=Groucho`, добавленных в конец URL). Эти данные нужно либо вернуть, либо сериализовать в требуемый формат, как правило, на базе JSON. Для организации полного цикла работы с данными в экосистеме Flask/Python есть несколько идеально подходящих библиотек:

— Flask выполняет функции сервера.
— Flask SQLAlchemy — расширение Flask для интеграции SQLAlchemy, выбранной нами Python-библиотеки с применением объектно-реляционного отображения (ORM, Object-Relational Mapping).

[1] Эти методы создания (create), чтения (read), обновления (update) и удаления (delete) складываются в аббревиатуру CRUD.

[2] Ключевые принципы RESTful: идентификация ресурсов через кешируемые stateless URI/URL и управление с помощью таких HTTP-методов, как GET или POST.

— Flask-Marshmallow — добавляющее поддержку библиотеки для сериализации объектов marshmallow.

Установим все нужные расширения с помощью `pip`:

```
$ pip install Flask-SQLALchemy flask-marshmallow
        marshmallow-sqlalchemy
```

Создание базы данных

В «Сохранении очищенных наборов данных» на стр. 312 было показано, как просто сохранить pandas DataFrame в SQL с помощью метода `to_sql`. Это удобный способ, но в полученной таблице отсутствует поле первичного ключа, уникального идентификатора для строки таблицы. Наличие первичного ключа является хорошей практикой и критически важно для операций создания/удаления строк через веб-API. Поэтому теперь создадим SQL-таблицу другим путем.

Сначала с помощью SQLAlchemy создадим базу данных SQLite, добавив столбец ID в качестве первичного ключа таблицы winners:

```
from sqlalchemy import Column, Integer, String, Text
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

Base = declarative_base()


class Winner(Base):
    __tablename__ = 'winners'
    id = Column(Integer, primary_key=True) ❶
    category = Column(String)
    country = Column(String)
    date_of_birth = Column(String) # string form dates
    date_of_death = Column(String)
    # ...
# создание базы SQLite и начало сессии
engine = sqlalchemy.create_engine(
            'sqlite:///data/nobel_winners_cleaned_api.db')
```

```
Base.metadata.create_all(engine)
Session = sessionmaker(bind=engine)
session = Session()
```

❶ Чтобы однозначно идентифицировать лауреатов, мы задаем первичный ключ.

Обратите внимание: чтобы избежать проблем с сериализацией объектов datetime, мы сохраняем даты в строковом формате. Перед сохранением в БД преобразуем соответствующие столбцы DataFrame:

```
df['date_of_birth'] = df['date_of_birth'].astype(str)
df['date_of_death'] = dl['date_of_death'].astype(str)
df.date_of_birth
#0    1927-10-08
#4    1829-07-26
#5    1862-08-29
..
```

Проитерируемся по строкам DataFrame, добавляя их в БД как записи словаря, а затем закоммитим все изменения одной транзакцией:

```
for d in df_tosql.to_dict(orient='records'):
      session.add(Winner(**d))
session.commit()
```

Создав правильно структурированную базу данных, рассмотрим, как взаимодействовать с ней во Flask.

Flask RESTful для работы с данными

Создадим стандартный Flask-сервер. Для начала импортируем стандартные модули Flask наряду с расширениями SQLALchemy и marshmallow, после чего инициализируем приложение:

```
from flask import Flask, request, jsonify
from flask_sqlalchemy import SQLAlchemy
```

```
from flask_marshmallow import Marshmallow

# Инициализация приложения
app = Flask(__name__)
```

Внутри приложения Flask инициализируем SQLAlchemy, объявив ее для нашей SQLite БД:

```
app.config['SQLALCHEMY_DATABASE_URI'] =\
      'sqlite:///data/nobel_winners_cleaned_api_test.db'

db = SQLAlchemy(app)
```

Теперь с помощью экземпляра db определим таблицу winners, как подкласс декларативной базовой модели. Это схема соответствует структуре таблицы winners, созданной в предыдущем подразделе:

```
class Winner(db.Model):
    __tablename__ = 'winners'
    id = db.Column(db.Integer, primary_key=True)
    category = db.Column(db.String)
    country = db.Column(db.String)
    date_of_birth = db.Column(db.String)
    date_of_death = db.Column(db.String)
    gender = db.Column(db.String)
    link = db.Column(db.String)
    name = db.Column(db.String)
    place_of_birth = db.Column(db.String)
    place_of_death = db.Column(db.String)
    text = db.Column(db.Text)
    year = db.Column(db.Integer)
    award_age = db.Column(db.Integer)

def __repr__(self):
    return "<Winner(name='%s', category='%s', year='%s')>"\
          % (self.name, self.category, self.year)
```

Сериализация с помощью marshmallow

Компактная Python-библиотека marshmallow выполняет только одну работу, но делает ее хорошо. Цитата из документации:

> marshmallow — ORM/ODM/фреймворк-независимая библиотека, предназначенная для преобразования сложных типов данных, таких как объекты, в нативные типы Python и обратно.

Для десериализации входных данных в объекты приложений и их валидации marshmallow использует схемы, аналогичные схемам SQLALchemy. Для нашей задачи важна способность marshmallow извлекать данные из SQLite (через SQLAlchemy), и преобразовывать их в JSON-совместимый формат.

Чтобы использовать Flask-Marshmallow, сначала создадим экземпляр marshmallow (`ma`), который инициализируем с помощью приложения Flask. Затем мы используем его для создания схемы marshmallow, используя в качестве базы модель SQLAlchemy `Winner`. Свойство `fields` схемы позволяет явно указать сериализуемые поля БД:

```
ma = Marshmallow(app)

class WinnerSchema(ma.Schema):
    class Meta:
        model = Winner
        fields = ('category', 'country', 'date_of_birth', ❶
                  'date_of_death', 'gender', 'link', 'name',
                  'place_of_birth', 'place_of_death', 'text',
                  'year', 'award_age')

winner_schema = WinnerSchema() ❷
winners_schema = WinnerSchema(many=True)
```

❶ Указываем поля базы данных для сериализации.
❷ Создаем два экземпляра схемы: для одиночных записей и коллекций.

Добавление маршрутов RESTful API

Теперь, когда готова основа, определим несколько маршрутов Flask для небольшого RESTful API. В качестве первого теста создадим маршрут, который вернет всех нобелевских лауреатов из таблицы базы данных:

```
@app.route('/winners/')
def winner_list():
all_winners = Winner.query.all() ❶
result = winners_schema.jsonify(all_winners) ❷
return result
```

❶ Запрос к базе данных для выбора всех строк таблицы winners.

❷ Схема marshmallow для множества строк принимает результат all_winners и сериализует его в JSON.

Протестируем API с помощью команды curl:

```
$ curl http://localhost:5000/winners/
[
  {
    "award_age": 57,
    "category": "Physiology or Medicine",
    "country": "Argentina",
    "date_of_birth": "1927-10-08",
    "date_of_death": "2002-03-24",
    "gender": "male",
    "link": "http://en.wikipedia.org/wiki/C%C3%A9sar_Milstein",
    "name": "C\u00e9sar Milstein",
    "place_of_birth": "Bah\u00eda Blanca, Argentina",
    "place_of_death": "Cambridge, England",
    "text": "C\u00e9sar Milstein, Physiology or Medicine, 1984",
    "year": 1984
  },
  {
    "award_age": 80,
    "category": "Peace", ...
  }...
]
```

Итак, у нас есть API-эндпоинт, который возвращает всех лауреатов. А как извлечь записи по ID (первичному ключу таблицы winners)? Для этого используем механизм сопоставления шаблонов маршрутов Flask: извлекаем ID из вызова API и выполняем целевой запрос к БД. Затем мы сериализуем результат в JSON с помощью однострочной схемы marshmallow:

```
@app.route('/winners/<id>/')
def winner_detail(id):
    winner = Winner.query.get_or_404(id) ❶
    result = winner_schema.jsonify(winner)
    return result
```

❶ По умолчанию Flask-SQLAlchemy в случае отсутствующих записей возвращает сообщение об ошибке 404, которое marshmallow может сериализовать в JSON.

Тестирование с помощью curl показывает, что ожидаемый единственный объект JSON возвращен:

```
$ curl http://localhost:5000/winners/10/
{
  "award_age": 60,
  "category": "Chemistry",
  "country": "Belgium",
  "date_of_birth": "1917-01-25",
  "date_of_death": "2003-05-28",
  "gender": "male",
  "link": "http://en.wikipedia.org/wiki/Ilya_Prigogine",
  "name": "Ilya Prigogine",
  "place_of_birth": "Moscow, Russia",
  "place_of_death": "Brussels, Belgium",
  "text": "Ilya Prigogine, born in Russia, Chemistry, 1977",
  "year": 1977
}
```

Возможность вызвать API и получить список всех лауреатов не представляет особой практической ценности. Добавим возможность фильтрации результатов с помощью аргументов запроса. Эти аргументы находятся в строке запроса в URL после символа `?`, разделяются амперсандами `&` и завершаются точкой. Например,

`http://nobel.net/api/winners?category=Physics&year=1980`. Flask предоставляет объект `request.args` с методом `to_dict`, который возвращает словарь аргументов URL[1]. Используем его для настройки фильтров таблицы данных, применяя пары ключ-значение через метод `filter_by` в SQLAlchemy. Вот простая реализация:

```
@app.route('/winners/')
def winner_list():
    valid_filters = ('year', 'category', 'gender', 'country', 'name') ❶
    filters = request.args.to_dict()

    args = {name: value for name, value in filters.items()
            if name in valid_filters} ❷
    # Этот цикл for дает тот же результат,
    # что и генератор словарей выше
    # args = {}
    # for vf in valid_filters:
    #  if vf in filters:
    #     args[vf] = filters.get(vf)
    app.logger.info(f'Filtering with the fields: {args}')
    all_winners = Winner.query.filter_by(**args) ❸
    result = winners_schema.jsonify(all_winners)
    return result
```

❶ Разрешаем фильтрацию по этим полям.

❷ Перебираем заданные фильтры, отбирая допустимые поля для формирования словаря. С помощью генератора словарей Python создаем словарь `args`.

❸ Применяем распаковку словарей Python, чтобы задать параметры метода.

Проверим возможности фильтрации с помощью curl, используя аргументы `-d` (данные) для указания параметров запроса:

```
$ curl -d category=Physics -d year=1933 -get \
        http://localhost:5000/winners/
```

[1] Технически строки запроса в URL образуют мультисловарь, в котором несколько значений могут иметь один ключ. Для нашего API предполагается наличие одного значения на ключ, поэтому преобразование в словарь допустимо.

```
[
  {
    "award_age": 31,
    "category": "Physics",
    "country": "United Kingdom",
    "date_of_birth": "1902-08-08",
    "date_of_death": "1984-10-20",
    "gender": "male",
    "link": "http://en.wikipedia.org/wiki/Paul_Dirac",
    "name": "Paul Dirac",
    "place_of_birth": "Bristol, England",
    "place_of_death": "Tallahassee, Florida, US",
    "text": "Paul Dirac, Physics, 1933",
    "year": 1933
  },
  {
    "award_age": 46,
    "category": "Physics",
    "country": "Austria",
    "date_of_birth": "1887-08-12",
    "date_of_death": "1961-01-04",
    "gender": "male",
    "link": "http://en.wikipedia.org/wiki/Erwin_Schr%C3%B6dinger",
    "name": "Erwin Schr\u00f6dinger",
    "place_of_birth": "Erdberg, Vienna, Austria",
    "place_of_death": "Vienna, Austria",
    "text": "Erwin Schr\u00f6dinger, Physics, 1933",
    "year": 1933
  }
]
```


Теперь мы реализовали детальную фильтрацию нашего набора данных — для многих визуализаций этого достаточно, чтобы работать с большими пользовательскими наборами, подгружаемыми по требованию через RESTful API. Одно из таких требований — возможность публиковать или создавать записи данных с помощью API или веб-формы. Нужно *знать*, как это делать. Можно использовать API как централизованное хранилище данных с возможностью пополнения из различных источников С помощью Flask и его расширений такая функциональность реализуется очень просто.

SKLADCHIK.ORG

Отправка данных через API

Необязательный аргумент `methods` определяет допустимые HTTP-методы в маршрутах Flask. По умолчанию используется метод GET, но указав POST, мы можем отправлять по этому маршруту данные в формате, который поддерживает объект `request`, например JSON-данные.

Добавим еще один эндпоинт `/winners` со списком `methods`, содержащим `POST`, затем используем данные из JSON, чтобы создать словарь `winner_data`, который применяется для создания записи в таблице winners. Затем запись добавляется в сессию базы данных и коммитится. Новая запись возвращается с помощью сериализации marshmallow:

```
@app.route('/winners/', methods=['POST'])
def add_winner():
    valid_fields = winner_schema.fields

    winner_data = {name: value for name,
            value in request.json.items() if name in valid_fields}
    app.logger.info(f"Creating a winner with these fields:
                    {winner_data}")
    new_winner = Winner(**winner_data)
    db.session.add(new_winner) db.session.commit()
    return winner_schema.jsonify(new_winner)
```

Тестируем результат с помощью curl:

```
$ curl http://localhost:5000/winners/ \
    -X POST \
    -H "Content-Type: application/json" \
    -d '{"category":"Physics","year":2021,
         "name":"Syukuro Manabe","country":"Japan"}' ❶
{
    "award_age": null,
    "category": "Physics",
    "country": "Japan",
    "date_of_birth": null,
    "date_of_death": null,
    "gender": null,
    "link": null,
    "name": "Syukuro Manabe",
```

```
    "place_of_birth": null,
    "place_of_death": null,
    "text": null,
    "year": 2021
}
```

❶ Входные данные — строка JSON.

Возможно, для управления данными будет полезнее API-эндпоинт, позволяющий обновлять данные лауреата. Используем для этого HTTP-метод PATCH, вызываемый по заданному URL. Как и в случае создания нового лауреата с помощью POST, проходим в цикле по словарю `request.json` и используем все валидные поля (в данном случае доступные сериализатору marshmallow) для обновления атрибутов лауреата по его ID:

```
@app.route('/winners/<id>/', methods=['PATCH'])
def update_winner(id):
    winner = Winner.query.get_or_404(id)
    valid_fields = winner_schema.fields
    winner_data = {name: value for name, value
                   in request.json.items() if name in valid_fields}
    app.logger.info(f"Updating a winner with these fields:
                    {winner_data}")
    for k, v in winner_data.items():
        setattr(winner, k, v) db.session.commit()
    return winner_schema.jsonify(winner)
```

Для демонстрации используем этот PATCH-эндпоинт, чтобы обновить имя лауреата и год присуждения премии:

```
$ curl http://localhost:5000/winners/3/ \
      -X PATCH \
      -H "Content-Type: application/json" \
      -d '{"name":"Morris Maeterlink","year":"1912"}'
{
    "award_age": 49,
    "category": "Literature",
    "country": "Belgium",
    "date_of_birth": "1862-08-29",
```

```
    "date_of_death": "1949-05-06",
    "gender": "male",
    "link": "http://en.wikipedia.org/wiki/Maurice_Maeterlinck",
    "name": "Morris Maeterlink",
    "place_of_birth": "Ghent, Belgium",
    "place_of_death": "Nice, France",
    "text": "Maurice Maeterlinck, Literature, 1911", ❶
    "year": 1912
}
```

❶ Исходные данные

На данном этапе мы создали целевой API, позволяющий выбирать данные с учетом тонко настраиваемых фильтров и обновлять или создавать данные о лауреатах. С увеличением числа эндпоинтов для дополнительных таблиц БД может усложниться маршрутизация Flask и связанные с ней методы. Проблему решает класс Flask MethodView, который объединяет все API-эндпоинты в рамках одного подкласса, что повышает чистоту кода и упрощает расширение функциональности. Реализация API на основе представлений MethodView ощутимо снизит когнитивную нагрузку, что особенно важно при масштабировании функционала API.

Расширение API с помощью MethodView

Большую часть кода нашего API можно переиспользовать, а переход на MethodView значительно сократит объем boilerplate-кода. MethodView позволяет объединить эндпоинты и соответствующие HTTP-методы (GET, POST и др.) в едином экземпляре класса, который легко расширять и адаптировать. Чтобы перенести нашу таблицу лауреатов в отдельный ресурс, определим методы маршрутизации Flask в классе `MethodView` и внесем минимальные изменения. Сначала импортируем класс `MethodView`:

```
#...
from flask.views import MethodView
#...
```

Нам не придется изменять ни модель SQLAlchemy, ни схемы marshmallow. Теперь для нашей коллекции лауреатов создадим экземпляр `MethodView`

с методами, соответствующими HTTP-методам. Используем повторно существующие методы маршрутов. Затем мы используем метод `add_url_rule` из приложения Flask, возвращающий эндпоинт, который обрабатывает представление:

```
class WinnersListView(MethodView):

    def get(self):
        valid_filters = ('year', 'category', 'gender', 'country', 'name')
        filters = request.args.to_dict()
        args = {name: value for name, value in filters.items()
                if name in valid_filters}
        app.logger.info('Filtering with the %s fields' % (str(args)))
        all_winners = Winner.query.filter_by(**args)
        result = winners_schema.jsonify(all_winners)
        return result

    def post(self):
        valid_fields = winner_schema.fields
        winner_data = {name: value for name,
              value in request.json.items() if name in valid_fields}
        app.logger.info("Creating a winner with these fields: %s" %
                str(winner_data))
        new_winner = Winner(**winner_data)
        db.session.add(new_winner) db.session.commit()
        return winner_schema.jsonify(new_winner)

app.add_url_rule("/winners/",
        view_func=WinnersListView.as_view("winners_list_view"))
```

HTTP-методы для отдельных записей таблицы создаются по той же схеме. Для полноты добавим метод delete. Успешный HTTP-запрос DELETE должен возвращать код состояния 204 (No Content) и пустой пакет:

```
class WinnerView(MethodView):

    def get(self, winner_id):
        winner = Winner.query.get_or_404(winner_id)
        result = winner_schema.jsonify(winner)
        return result
```

```
    def patch(self, winner_id):
        winner = Winner.query.get_or_404(winner_id)
        valid_fields = winner_schema.fields
        winner_data = {name: value for name,
              value in request.json.items() if name in valid_fields}
        app.logger.info("Updating a winner with these fields: %s" %
              str(winner_data))
        for k, v in winner_data.items():
              setattr(winner, k, v)
        db.session.commit()
        return winner_schema.jsonify(winner)

    def delete(self, winner_id):
        winner = Winner.query.get_or_404(winner_id)
        db.session.delete(winner)
        db.session.commit()
        return '', 204

app.add_url_rule("/winners/<winner_id>",
                 view_func=WinnerView.as_view("winner_view")) ❶
```

❶ Именованные аргументы, соответствующие URL-шаблону, передаются в методы MethodView.

С помощью curl удалим одного из лауреатов с выводом подробной информации:

```
$ curl http://localhost:5000/winners/858 -X DELETE -v
* Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 5000 (#0)
> DELETE /winners/858 HTTP/1.1
> Host: localhost:5000
> User-Agent: curl/7.47.0
> Accept: */*
>
* HTTP 1.0, assume close after body
< HTTP/1.0 204 NO CONTENT
< Content-Type: application/json
< Server: Werkzeug/2.0.2 Python/3.8.9
```

```
< Date: Sun, 27 Mar 2022 15:35:51 GMT
<
* Closing connection 0
```

Используя представления на основе MethodView для выделенных эндпоинтов, мы избавляемся от boilerplate-кода маршрутизации Flask, что упрощает работу с кодовой базой и ее расширение. В качестве примера реализуем полезную функциональность API — разделение данных на страницы (пагинацию).

Пагинация возвращаемых данных

При возврате больших наборов данных очень полезна и даже необходима возможность API разбивать данные на части (страницы).

SQLAlchemy предоставляет для запросов удобный метод `paginate`, позволяющий возвращать данные страницами заданного размера. Чтобы добавить пагинацию в наш API, достаточно ввести два параметра: номер и размер страницы. Мы будем использовать `_page` и `_page-size`, добавив спереди символ подчеркивания, чтобы они отличались от фильтрующих запросов, которые мы можем применять.

Ниже показан адаптированный метод `get`:

```
class WinnersListView(MethodView):

    def get(self):
        valid_filters = ('year', 'category', 'gender', 'country', 'name')
        filters = request.args.to_dict()
        args = {name: value for name, value in filters.items()
                if name in valid_filters}

        app.logger.info(f'Filtering with the {args} fields')

        page = request.args.get("_page", 1, type=int) ❶
        per_page = request.args.get("_per-page", 20, type=int)

        winners = Winner.query.filter_by(**args).paginate(page,
                per_page) ❷
        winners_dumped = winners_schema.dump(winners.items)
```

```
    results = {
        "results": winners_dumped,
        "filters": args,
        "pagination": ❸
        {
            "count": winners.total,
            "page": page,
            "per_page": per_page,
            "pages": winners.pages,
        },
    }

    make_pagination_links('winners', results) ❹

    return jsonify(results)
# ...
```

❶ Задаем только необходимые параметры пагинации.

❷ Используем метод `paginate` с параметрами `page` (страница) и `per_page` (размер страницы).

❸ Возвращаем разбитые на страницы результаты и необходимые сопутствующие данные. В словаре `pagination` мы предоставляем ключевую информацию: номер текущей страницы и размер всего набора данных.

❹ С помощью этой функции добавляем несколько удобных URL для перехода на предыдущие или следующие страницы.

Принято возвращать адреса эндпоинтов для предыдущей и следующей страниц, что упрощает использование всего набора данных. У нас есть функция `make_pagination_links`, которая добавляет эти URL-адреса в словарь пагинации. Чтобы построить URL-строку запроса, воспользуемся Python-библиотекой *urllib*:

```
#...
import urllib.parse
#...
def make_pagination_links(url, results):
    pag = results['pagination']
    query_string = urllib.parse.urlencode(results['filters']) ❶
```

```
    page = pag['page']
    if page > 1:
        prev_page = url + '?_page=%d&_per-page=%d%s' % (page-1,
                                                        pag['per_page'],
                                                        query_string)
    else:
        prev_page = ''

    if page < pag['pages']: ❷
        next_page = url + '?_page=%d&_per-page=%d%s' % (page+1,
                                                        pag['per_page'],
                                                        query_string)
    else:
        next_page = ''

    pag['prev_page'] = prev_page
    pag['next_page'] = next_page
```

❶ Формируем строку запроса из параметров фильтрации, например `&category=Chemis try&year=1976`. Модуль parse библиотеки *urllib* преобразует словарь фильтров в корректный URL-запрос.

❷ Добавляем в результаты адреса предыдущей/следующей страниц с учетом параметров фильтрации.

С помощью curl проверим пагинацию данных. Добавим фильтр для отбора всех лауреатов Нобелевской премии по физике:

```
$ curl -d category=Physics --get http://localhost:5000/winners/
{
   "filters": {
      "category": "Physics"
   },
   "pagination": {
      "count": 201,
      "next_page": "?_page=2&_per-page=20&category=Physics", ❶
      "page": 1,
      "pages": 11,
      "per_page": 20,
```

```
    "prev_page": ""
  },
  "results": [ ❷
    {
      "award_age": 81,
      "category": "Physics",
      "country": "Belgium",
      "date_of_birth": "1932-11-06",
      "date_of_death": "NaT",
      "gender": "male",
      "link": "http://en.wikipedia.org/wiki/Fran%C3%A7ois_Englert",
      "name": "Fran\u00e7ois Englert",
      "place_of_birth": "Etterbeek , Brussels , Belgium",
      "place_of_death": null,
      "text": "Fran\u00e7ois Englert , Physics, 2013",
      "year": 2013
    },
    {
      "award_age": 37,
      "category": "Physics",
      "country": "Denmark",
      "date_of_birth": "1885-10-07",
      "date_of_death": "1962-11-18",
      "gender": "male",
      "link": "http://en.wikipedia.org/wiki/Niels_Bohr",
      "name": "Niels Bohr",
...
}]}
```

❶ Это первая страница, поэтому предыдущей у нее нет, но для удобства обработки данных указан URL следующей страницы.

❷ Список results, содержащий первых 20 физиков из таблицы winners.

При помощи библиотек типа marshmallow, интегрированных как расширения Flask, достаточно просто создать собственный API, не обращаясь к специализированной библиотеке RESTful Flask, которая, как показывает опыт, может быстро устареть.

Удаленное развертывание API на Heroku

Недавно созданный нами локальный dev-сервер идеально подходит для прототипирования, тестирования потоков данных и решения задач визуализации, связанных с обработкой слишком больших для JSON-файлов наборов данных. Однако это означает, что никто не сможет использовать нашу визуализацию без доступа к нашему локальному серверу. Поэтому требуется разместить сервер данных в интернете в качестве удаленного ресурса. Есть несколько способов сделать это, но питонисты, включая меня, очень любят Heroku — облачный сервис, который упрощает развертывания Flask-сервера. Давайте убедимся в этом, разместив наш сервер данных о нобелевских лауреатах в сети.

Сначала создадим бесплатный аккаунт на Heroku. Затем установим клиентские инструменты Heroku для вашей ОС.

Установив инструменты, войдите на Heroku, запустив из командной строки `login`:

```
$ heroku login
heroku: Press any key to open up the browser to login or q to exit
   › Warning: If browser does not open, visit
   › https://cli-auth.heroku.com/auth/browser/***
heroku: Waiting for login...
Logging in... done
Logged in as me@example.com
```

После входа создадим приложение Heroku и развернем его в интернете. Создадим каталог для приложения (*heroku_api*) и поместим туда файл *api_rest.py* с нашим Flask API. Также потребуются Procfile, файл *requirements.txt* и база данных SQLite `nobel_win ners_cleaned_api.db`:

```
heroku_api
├── api_rest.py
├── data
│   ├── nobel_winners_cleaned_api.db
├── Procfile
└── requirements.txt
```

Heroku использует файл Procfile, чтобы знать, что и как развертывать. В данном случае мы будем использовать HTTP-сервер Gunicorn WSGI для Python, чтобы направлять веб-трафик в приложение Flask и запускать его как Heroku-приложение. Содержимое Procfile может выглядеть так:

```
web: gunicorn api_rest: app
```

SKLADCHIK.ORG

Помимо Procfile, Heroku требует указать Python-библиотеки, устанавливаемые для приложения. Их список содержит файл *requirements.txt*:

```
Flask==2.0.2
gunicorn==20.1.0
Flask-Cors==3.0.10
flask-marshmallow==0.14.0
Flask-SQLAlchemy==2.5.1
Jinja2==3.0.1
marshmallow==3.15.0
marshmallow-sqlalchemy==0.28.0
SQLAlchemy==1.4.26
Werkzeug==2.0.2
```

Создадим Heroku-приложение, запустив `create` из командной строки.

Инициализируем с помощью `git` директорию Git и добавим существующие файлы:

```
$ git init
$ git add.
$ git commit -m "First commit"
```

После инициализации Git создаем Heroku-приложение[1]:

```
$ heroku create flask-rest-pyjs2
```

Для развертывания на Heroku теперь достаточно просто выполнить `git push`:

```
$ git push heroku master
```

Любые изменения в локальном коде можно отправлять в Heroku через git push для обновления сайта.

Давайте с помощью curl протестируем API, запросив первую страницу лауреатов-физиков:

```
$ curl -d category=Physics --get
https://flask-rest-pyjs2.herokuapp.com/winners/
```

[1] Настройку можно выполнить через дашборд Heroku, после чего привязать текущий Git-каталог к приложению с помощью `git remote - <app_name>`.

```
{"filters":{"category":"Physics"},"pagination":{"count":201,
"next_page":"winners/?_page=2&_per-page=20&category=Physics",
"page":1, "pages":11,"per_page":20,"prev_page":""},
"results":[{"award_age":81, "category":"Physics",
"country":"Belgium","date_of_birth":"1932-11-06",
"date_of_death":"NaT","gender":"male",
"link":"http://en.wikipedia.org/wiki/Fran%C3%A7ois_Englert",
"name":"Fran\u00e7ois Englert", ... }
```


CORS

Чтобы использовать API из браузера, необходимо учитывать ограничения межсайтового обмена ресурсами CORS (Cross-Origin Resource Sharing) для серверных запросов данных. Мы будем применять расширение Flask-CORS и активировать его с настройками по умолчанию, разрешающими запросам с любого домена получать доступ к серверу данных. Для этого нужно добавить в наше приложение Flask всего пару строк кода:

```
# ...
from flask_cors import CORS
# Инициализация приложения
app = Flask(__name__)
CORS(app)
```

Библиотека Flask-CORS позволяет точно указывать домены, которым разрешен доступ к ресурсам. В данном случае мы разрешаем общий доступ.

Использование API с помощью JavaScript

Для запроса данных из веб-приложения/страницы мы будем использовать `fetch`. В приведенном примере извлекаются все данные с пагинацией. Страницы последовательно запрашиваются до тех пор, пока свойство `next_page` не примет пустое значение:

```
let data
async function init() {
  data = await getData(
            'winners/?category=Physics&country=United States') ❶
  console.log(`${data.length} US Physics winners:`, data)
```

```
    // Отправить данные в подходящую функцию построения диаграмм
    drawChart(data)
}

init()

async function getData(ep='winners/?category=Physics'){ ❶
    let API_URL = 'https://flask-rest-pyjs2.herokuapp.com/'
    let data = []
    while(true) {
        let response = await fetch(API_URL + ep) ❷
        .then(res => res.json()) ❸
        .then(data => {
            return data
        })

        ep = response.pagination.next_page
        data = data.concat(response.results) // добавить результаты
                                             // страницы
        if(!ep) break // нет следующей страницы — выход из цикла
    }
    return data
}
```

❶ Используем асинхронные функции, чтобы асинхронно получать данные с сервера.

❷ Оператор `await` приостанавливает выполнение кода, пока промис не завершится и не вернет значение.

❸ Преобразуем полученные данные в формат JSON, передаем их при следующем вызове `then`, который возвращает данные сервера.

Вызов JS выводит на консоль ожидаемый результат:

```
89 US Physics winners:
[{
      award_age: 42
      category: "Physics"
      country: "United States"
      date_of_birth: "1969-12-16"
```

```
        date_of_death: "NaT"
        gender: "male"
        link: "http://en.wikipedia.org/wiki/Adam_G._Riess"
        name: "Adam G. Riess"
        place_of_birth: "Washington, D.C., United States"
        place_of_death: null
        text: "Adam G. Riess , Physics, 2011"
        year: 2011
    }, ...
]}
```

Теперь наш веб-ориентированный RESTful API доступен из любой точки мира (с учетом ограничений CORS). Благодаря низким накладным расходам и простоте использования Heroku — вне конкуренции. Эта платформа работает уже довольно давно и за это время была хорошо доработана.

Резюме

Надеюсь, в этой главе мне удалось показать, что с помощью нескольких мощных расширений можно легко создать собственный RESTful API. Конечно, чтобы довести его до промышленного стандарта, потребуется еще немало работы и тестирования, но с большинством наших задач этот API справится, предоставляя возможность работать с большими наборами данных и позволяя пользователю свободно вести исследования. Как минимум, этот пример показывает, как быстро развернуть сервер для тестирования пробных визуализаций обработанных данных. Дашборды — еще одна область, где требуется извлечение данных из внешних источников.

Удаленное развертывание API на Heroku позволяет сегментировать большие наборы данных без локальных серверов — это идеально для демонстрации масштабных визуализаций клиентам или коллегам.

РАЗДЕЛ V

Визуализация данных с помощью D3 и Plotly

В этом разделе мы превратим набор данных о лауреатах Нобелевской премии (который мы скрейпили в главе 6 и очищали в главе 9) в современную интерактивную веб-визуализацию. Помогут нам в этом мощная JS-библиотека визуализации D3 (см. рисунок V-1) и Python-библиотека Plotly.

Мы подробно рассмотрим создание визуализации данных о лауреатах Нобелевской премии, попутно приобретая знания о D3 и JavaScript. Прежде всего определим требования к визуализации, используя наблюдения из главы 11.

Исходный код этой визуализации на Python и JavaScript вы найдете в каталоге *nobel_viz* в репозитории книги на GitHub (подробнее см. «Сопутствующий код» на стр. 36).

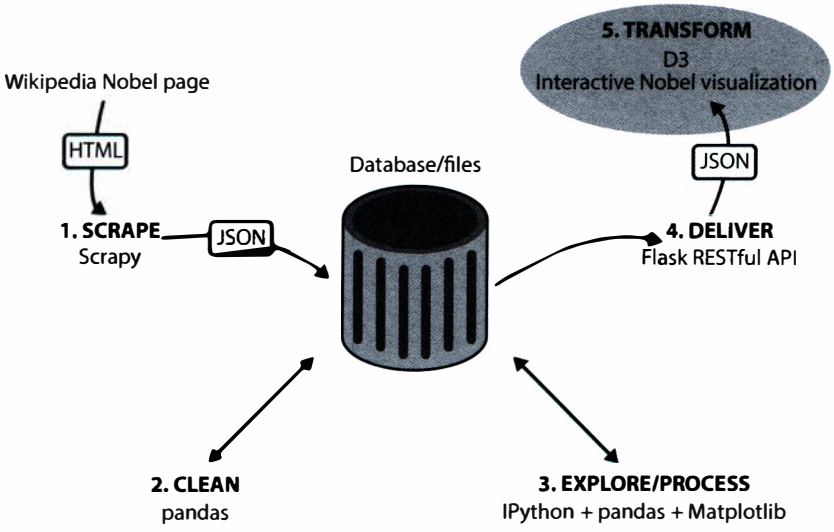


Рис. V-1. *Наш тулчейн для визуализации: получение данных*



Исходный код для этого раздела доступен в репозитории книги на GitHub: https://github.com/Kyrand/dataviz-with-python-and-js-ed-2.

Перенос диаграмм в интернет с помощью Matplotlib и Plotly

В этой главе мы рассмотрим, как перенести в веб-формат данные, обработанные с помощью pandas. Статическая визуализация часто является оптимальной для представления данных. Создадим такую с помощью Matplotlib. Однако взаимодействие пользователя с визуализацией обогащает ее. Мы рассмотрим, как с помощью Python-библиотеки Plotly создавать интерактивные визуализации в блокноте Jupyter и переносить их (включая UI) на веб-страницу.

Работая с Python-версией Plotly, мы параллельно получим навыки использования ее нативной JS-библиотеки, расширяющей возможности веб-визуализации. Убедимся в этом, с помощью JS предоставив пользователю возможность взаимодействовать с диаграммами Plotly.

Создание статических диаграмм с помощью Matplotlib

У статических диаграмм свои плюсы — их создатели полностью контролируют редактирование. Одна из сильных сторон Matplotlib — с ее помощью можно создавать высококачественные диаграммы в широком диапазоне форматов: от PNG высокой четкости до SVG-изображений с векторными примитивами, которые великолепно масштабируются в зависимости от размера документа.

Для веб-графики рекомендуется использовать формат PNG (Portable Network Graphics, «портативная сетевая графика»). Как видно из названия, он был специально создан для этой цели. Давайте выложим в интернет несколько диаграмм из главы 11 в формате PNG и сделаем попутно небольшое веб-представление.

Мы выяснили, начертив пару столбчатых диаграмм, что абсолютные показатели по странам дают иную картину, чем данные на душу населения. Теперь превратим это небольшое открытие в веб-представление. Чтобы легче читались названия стран, используем горизонтальные столбчатые диаграммы.

Для текущего объекта осей Matplotlib, возвращаемого методом plot библиотеки pandas, изменяем цвет фона на светло-серый (#eee) и добавляем подписи осей:

```
ax = df_countries[df_countries.nobel_wins > 2]\ ❶
   .sort_values(by='nobel_wins_per_capita', ascending=True)\
   .nobel_wins_per_capita.plot(kind='barh',\ ❷
      figsize=(5, 10), title="Relative prize numbers")
ax.set_xlabel("Nobel prizes per capita")
ax.set_facecolor("#eee")
plt.tight_layout() ❸
plt.savefig("country_relative_prize_numbers.png")
```

❶ Порог отбора стран — минимум три премии.
❷ Задаем горизонтальную диаграмму — `kind='barh'`.
❸ Используем метод `tight_layout`, который автоматически подбирает параметры, чтобы элементы диаграммы не вышли за пределы сохраненной фигуры.

Ту же процедуру повторяем для абсолютных значений, получив две горизонтальные столбчатые диаграммы PNG. Для их веб-отображения используем рассмотренные в главе 4 HTML и CSS:

```
<!— index.html —>
<div class="main">
   <h1 class='title'>The Nobel Prize</h1>
   <h2>A few exploratory nuggets</h2>

   <div class="intro">
      <p>Some nuggets of data mined from a dataset of Nobel prize
      winners (2016).</p>
   </div>

   <div class="container" id="by-country-container">

      <div class="info-box">
         <p>These two charts compare Nobel prize winners by
         country. [...] that Sweden, winner by a relative metric,
         hosts the prize.</p>
      </div>
```

```
        <div class="chart-wrapper" id="by-country">

          <div class="chart">
            <img src="images/country_absolute_prize_numbers.png" ❶
            alt="">
          </div>
          <div class="chart">
            <img src="images/country_relative_prize_numbers.png"
            alt="">
          </div>

        </div>
      </div>
    </div>
```

❶ Путь к подкаталогу с изображениями указан относительно *index.html*.

Следом за заголовком, подзаголовком и вступлением идет основной контейнер, внутри которого находится блок `div` с классом chart-wrapper, содержащий две диаграммы и информационный блок.

С помощью CSS задаем размер, позиционирование и стиль контента. Чтобы выстроить диаграммы и информационный блок в один ряд, мы используем Flexbox CSS. С помощью свойства flex задаем относительную ширину: контейнер chart-wrapper будет в два раза шире, чем info-box:

```
html,
body {
  height: 100%;
  font-family: Georgia, serif;
  background: #fff1e5;
  font-size: 1.2em;
}

h1.title {
  font-size: 2.1em;
}

.main {
  padding: 10px;
```

```
    padding-bottom: 100px;
    min-width: 800px;
    max-width: 1200px;
}

.container {
    display: flex;
}

.chart-wrapper {
    display: flex; ❶
    flex: 2; ❷
}

.chart {
    flex: 1;
    padding: 0 1.5em;
}
.chart img {
max-height: 600px;
}

.info-box {
    font-family: sans-serif;
    flex: 1; ❷
    font-size: 0.7em;
    padding: 0 1.5em;
    background: #ffebd9;
}
```

❶ Элементы с классами chart и info-box наследуют свойства flex от родительского контейнера chart-wrapper.

❷ Контейнер chart-wrapper в два раза шире, чем info-box.

Слева на рисунке 14.1 показана получившаяся веб-страница.

SKLADCHIK.ORG

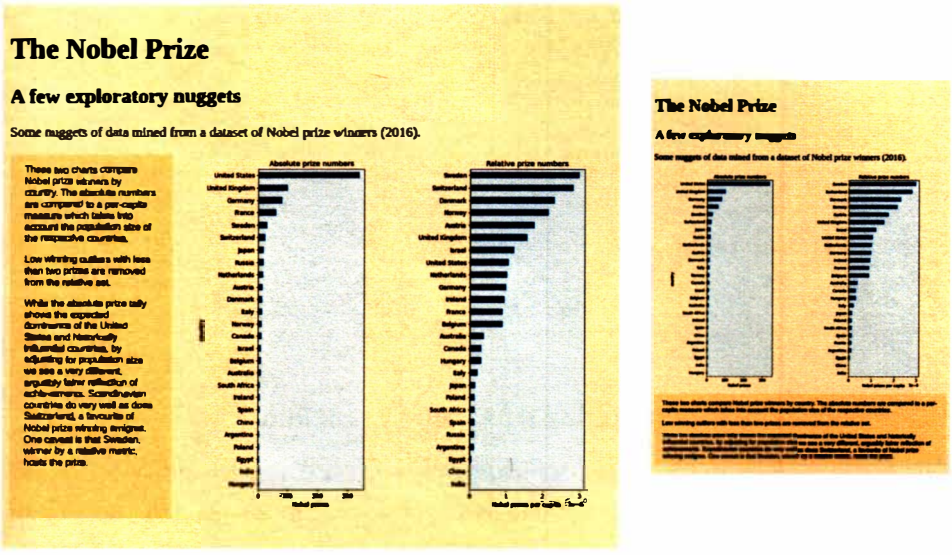


Рис. 14.1. Две статические диаграммы

Адаптация к размерам экрана

Современная веб-разработка и визуализация данных сталкиваются с проблемой адаптации под множество устройств для доступа в интернет. В большинстве случаев функции смартфонов и планшетов для прокрутки (pan) и масштабирования (pinch/zoom) позволяют просматривать один и тот же вариант визуализации на всех устройствах. Однако сделать визуализации адаптируемыми непросто: довольно быстро происходит комбинаторный взрыв. Часто оптимальным решением становится компромиссная компоновка.

В некоторых случаях для адаптации стилей под размер экрана устройства эффективно использовать директиву CSS `@media` — изменение ширины экрана выступает триггером для применения специализированных стилей. Продемонстрируем это на примере веб-страницы с данными нобелевских лауреатов.

Макет диаграммы на рисунке 14.1 подходит для большинства экранов ноутбуков или ПК, но чем меньше экран устройства, тем теснее становится диаграммам и информационному полю, поэтому информационное поле удлиняется, чтобы вместить текст. Активация изменения флекс-контейнера при ширине экрана 1000 пикселей улучшит восприятие визуализации на небольших устройствах.

Добавляем @media screen для изменения значения `flex-direction` на устройствах с шириной экрана ≤1000px. В результате информационное поле и диаграммы будут отображаться не в строку, а в столбец, и в обратном

порядке — информационное поле окажется внизу, как показано в правой части рисунка 14.1:

```
/* Для браузера шириной 1000 px и менее */
@media screen and (max-width: 1000px) {
   #by-country-container {
      flex-direction: column-reverse;
   }
}
```

Использование изображений и активов удаленно

Мы можем использовать такие удаленные ресурсы, как размещенные в Dropbox или на Google изображения, получив публичную ссылку на них и указав ее в качестве источника. Например, следующие теги `<img>` используют для рисунка 14.1 изображения из Dropbox, а не размещенные локально:

```
<div class="chart">
   <img src="https://www.dropbox.com/s/422ugyhvfc0zg99/
   country_absolute_prize_numbers.png?raw=1" alt="">
</div>
<div class="chart">
   <img src="https://www.dropbox.com/s/n6rfr9kvuvir7gi/
   country_relative_prize_numbers.png?raw=1" alt="">
</div>
```

Построение диаграмм с помощью Plotly

Библиотека Matplotlib предоставляет исключительно гибкую настройку статических диаграмм в форматах PNG или SVG, хотя ее API нельзя назвать интуитивно понятным. Однако, если требуется добавить динамические/интерактивные элементы (например, кнопки или селекторы, позволяющие изменять или фильтровать набор данных) то понадобится другая библиотека — Plotly[1].

Plotly — это поддерживающая несколько языков (включая Python) библиотека для построения диаграмм, которую, как и Matplotlib, можно использовать в интерактивных сессиях Jupyter Notebook. Она предлагает больше типов

[1] Bokeh — достойная альтернатива.

диаграмм, чем стандартная Matplotlib, а ее настройка проще и интуитивнее. Этих преимуществ уже достаточно для выбора Plotly, но ее главная сила — возможность публикации в интернете диаграмм с интерактивными виджетами.

Хотя интерактивность требуется далеко не всегда, в любом случае Plotly предлагает полезные функции вроде подсказок с детальной информацией, всплывающих при наведении на элементы диаграммы.

Основные диаграммы

Посмотрим, как Plotly воспроизводит одну из диаграмм Matploblib из подраздела «Исторические тренды в распределении премии» на стр. 289. Сначала мы создадим DataFrame из набора данных о лауреатах Нобелевской премии, показывающий совокупные премии по трем географическим регионам:

```
new_index = pd.Index(np.arange(1901, 2015), name='year')

by_year_nat_sz = df.groupby(['year', 'country'])\
   .size().unstack().reindex(new_index).fillna(0)

# Наш список был составлен путем выбора двух-трех стран
# с наибольшим числом лауреатов на каждом из трех континентов.
regions = [
{'label':'N. America',
'countries':['United States', 'Canada']},
{'label':'Europe',
'countries':['United Kingdom', 'Germany', 'France']},
{'label':'Asia',
'countries':['Japan', 'Russia', 'India']}
]
# Создаем новый столбец с меткой региона для каждого
# словаря в списке регионов, суммируя страны региона.
for region in regions:
   by_year_nat_sz[region['label']] =\
   by_year_nat_sz[region['countries']].sum(axis=1)
# Создаем новый DataFrame, используя кумулятивную сумму
# столбцов нового региона.
df_regions = by_year_nat_sz[[r['label'] for r in regions]].\
   cumsum()
```

Получаем DataFrame `df_regions` с совокупной суммой по каждому столбцу.

```
df_regions
country  N. America  Europe   Asia
year
1901     0.0         4.0      0.0
1902     0.0         7.0      0.0
1903     0.0         10.0     0.0
1904     0.0         13.0     1.0
1905     0.0         15.0     1.0
...      ...         ...      ...
2010     327.0       230.0    36.0
2011     333.0       231.0    36.0
...
```

Модуль Express библиотеки Plotly

Plotly содержит в себе модуль `express`, ускоряющий создание набросков диаграмм, которые прекрасно подходят для экспресс-анализа в блокноте Jupyter. Этот модуль содержит высокоуровневые объекты для линейных, столбчатых и других диаграмм и может принимать DataFrame pandas, интерпретируя данные в столбцах[1]. Всего две строки кода, и Plotly Express выстроил линейную диаграмму, напрямую используя только что созданный нами DataFrame регионов. Полученную диаграмму см. на рисунке 14.2 (слева):

```
# импорт модуля express
import plotly.express as px
# используем метод line с нужным DataFrame
fig = px.line(df_regions)
fig.show()
```

Обратите внимание на метку индекса строки, которая по умолчанию используется для оси X, и на подсказку, всплывающую при наведении указателя мыши, которая в данном случае показывает информацию об участке линии.

[1] Можно легко привести DataFrame к необходимой форме столбцов с помощью оператора транспонирования `T`.

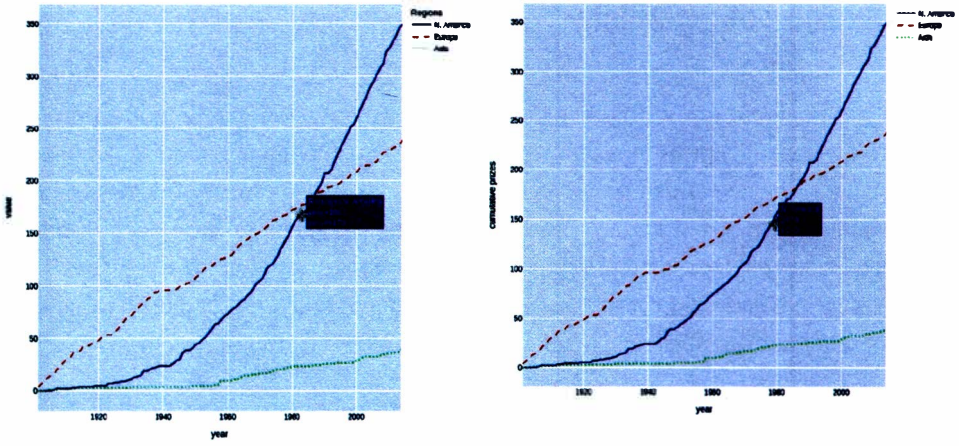


***Рис. 14.2.** Отображение кумулятивных данных о премиях с помощью Plotly*

Следует также отметить, что метка легенды берется из индекса группы, в данном случае — `country`. Мы можем легко переименовать ее во что-то более подходящее, в данном случае в `Regions`:

```
fig = px.line(df_regions, labels={'country': 'Regions'})
   line_dash='country', line_dash_sequence=['solid', 'dash', 'dot']) ❶
)
fig.show()
```

❶ Plotly по умолчанию использует цветовые различия линий, но для печатной версии книги мы изменим их стили (сплошная, пунктирная и др.). Для этого мы указываем в аргументе `line_dash` группировку по странам, а в `line_dash_sequence` — последовательность стилей линий[1].

Простой в использовании Plotly Express предлагает новые типы диаграмм[2]. По быстроте создания набросков данных он конкурирует с оберткой от pandas для Matplotlib, работающей непосредственно с DataFrame. Но если вы хотите получить больше контроля над своими графиками и в полной мере использовать преимущества Plotly, я рекомендую фигуры и графические объекты этой библиотеки. API этой библиотеки более сложный, но гораздо более мощный. Ее API зеркально повторяет JavaScript API — таким образом, вы изучаете сразу две библиотеки, что крайне полезно, как будет показано далее в главе.

[1] Доступные стили линий в Plotly см. на *https://oreil.ly/zUyxK*.

[2] Примеры демо см. на сайте Plotly.

Графические объекты Plotly

Использование графических объектов Plotly требует несколько больше boilerplate-кода, но принцип создания остается одинаковым для любых объектов: столбчатых и скрипичных диаграмм, карт и др. Основная идея — использовать массив графических объектов (точек, линий, столбиков, японских свечей, «ящиков с усами» и т. д.) в качестве данных для фигуры. Объект `layout` определяет параметры оформления графика.

Код ниже создает диаграмму, показанную на рисунке 14.2 (справа). Обратите внимание на настраиваемую подсказку, всплывающую при наведении:

```
import plotly.graph_objs as go


traces = [] ❶
for region in regions:
   name = region['label']
   traces.append(
      go.Scatter(
         x=df_regions.index, # years
         y=df_regions[name], # cum. prizes
         name=name,
         mode="lines", ❷
         hovertemplate=f"{name}<br>%{{x}}<br>$%{{y}}
                        <extra></extra>" ❸
         line=dict(dash=['solid', 'dash', 'dot'][len(traces)]) ❹
      )
   )
layout = go.Layout(height=600, width=600,\ ❺
   xaxis_title="year", yaxis_title="cumulative prizes")
fig = go.Figure(traces, layout) ❻
fig.show()
```

❶ Создадим массив объектов линейных графиков в качестве данных для фигуры.

❷ В линейном режиме точки рассеянных объектов соединяются.

❸ Зададим HTML-шаблон подсказки, которая будет отображаться при наведении мыши и сообщать координаты `x` и `y` для текущей точки диаграммы.

❹ Свойство *line* объекта `Scatter` позволяет задавать для линий цвет, стиль, форму и др[1]. Зададим стиль линий, чтобы они отличались друг от друга

[1] Подробнее см. *https://oreil.ly/8UDgA*.

в черно-белой печатной книге. Для этого мы используем размер (`len`) массива `traces` как индекс для последовательного получения стилей из массива.

❺ Помимо данных мы предоставляем объект `layout`, определяющий макет: размеры диаграммы, заголовки осей и др.

❻ Создаем фигуру, используя массив графических объектов и макет.

Создание карт с помощью Plotly

Еще одно достоинство Plotly — ее библиотеки карт и интеграция с экосистемой Mapbox, ведущей платформой для создания веб-карт на основе тайлов. Система тайлов Mapbox позволяет быстро и эффективно создавать карты для визуализаций.

Продемонстрируем работу с картами Plotly на примере распределения нобелевских лауреатов по странам мира.

Сначала создадим DataFrame с количеством премий по номинациям для стран, имеющих лауреатов, и добавим столбец `Total`:

```
df_country_category = df.groupby(['country', 'category'])\
      .size().unstack()
df_country_category['Total'] = df_country_category.sum(1)
df_country_category.head(3) # top three rows
#category  Chemistry  Economics  Literature Peace    Physics \
#country
#Argentina 1.0        NaN        NaN        2.0      NaN
#Australia NaN        1.0        1.0        NaN      1.0
#Austria   3.0        1.0        1.0        2.0      4.0
#
#category     Physiology or Medicine   Total
#country
#Argentina    2.0                      5.0
#Australia    6.0                      9.0
#Austria      4.0                      15.0
```

Мы используем столбец `Total` для фильтрации строк, оставив только страны с тремя и более нобелевскими лауреатами. Сделаем копию этого среза, чтобы избежать ошибок pandas DataFrame при модификации представления:

```
df_country_category = df_country_category.\
   loc[df_country_category.Total > 2].copy()
df_country_category
```

Теперь нам понадобятся координаты географических центров (центроидов) стран. Для решения этой задачи используем Geopy — замечательную маленькую Python-библиотеку.

Установите ее с помощью `pip` или аналогичного менеджера пакетов:

```
!pip install geopy
```

Теперь мы можем использовать модуль `Nominatim`, чтобы получить координаты на основе строк с названиями стран. Создадим геолокатор, передав строку в user-agent:

```
from geopy.geocoders import Nominatim

geolocator = Nominatim(user_agent="nobel_prize_app")
```

Используя геолокатор, мы можем пройти в цикле по странам в индексе DataFrame, чтобы получить доступные географические данные:

```
for name in  df_country_category.index[:5]:
   location = geolocator.geocode(name)
   print("Name: ", name)
   print("Coords: ", (location.latitude, location.longitude))
   print("Raw details: ", location.raw)
#Name: Argentina
#Coords: (-34.9964963, -64.9672817)
#Raw details: {'place_id': 284427148, 'licence': 'Data ©
#OpenStreetMap contributors, ODbL 1.0. https://osm.org/
#copyright', 'osm_type': 'relation', 'osm_id': 286393,
#'boundingbox': ['-55.1850761', '-21.7808568', '-73.5605371',
#[...]}
```

С помощью геолокатора добавим в DataFrame столбцы для географической широты (`Lat`) и долготы (`Lon`):

```
lats = {} lons = {}
for name in df_country_category.index:
   location = geolocator.geocode(name)
   if location:
      lats[name] = location.latitude
```

```
        lons[name] = location.longitude
    else:
        print("No coords for %s"%name)

df_country_category.loc[:,'Lat'] = pd.Series(lats) df_country_
category.loc[:,'Lon'] = pd.Series(lons) df_country_category
#category  Chemistry  Economics  Literature Peace   Physics \
#country
#Argentina 1.0        NaN        NaN        2.0     NaN
#Australia NaN        1.0        1.0        NaN     1.0
#
#category  Physiology or Medicine   Total   Lat         Lon
#country
#Argentina 2.0                      5.0     -34.996496 -64.967282
#Australia 6.0                      9.0     -24.776109 134.755000
```

Мы изобразим на карте маркеры, отражающие количество нобелевских премий по странам. Размер круга-маркера должен соответствовать количеству наград, поэтому нам потребуется функция для вычисления соответствующего радиуса. Для ручной настройки размера маркеров используем параметр `scale`:

```
def calc_marker_radius(size, scale=5):
    return np.sqrt(size/np.pi) * scale
```

Модуль Express библиотеки Plotly, наряду с созданием основных диаграмм, позволяет быстро создавать карты с использованием pandas DataFrame. В Express есть специализированный метод `scatter_map box`, который возвращает объект фигуры. Используем эту фигуру для настройки макета карты через один из бесплатных стилей Plotly (carto-positron), как показано на рисунке 14.3:

```
import plotly.express as px
init_notebook_mode(connected=True)

size = df_country_category['Total'].apply(calc_marker_radius,
        args=(16,)) ❶
fig = px.scatter_mapbox(df_country_category, lat="Lat", lon="Lon", ❷
                        hover_name=df_country_category.index, ❸
                        hover_data=['Total'],
                        color_discrete_sequence=["olive"],
```

```
                        zoom=0.7, size=size)
fig.update_layout(mapbox_style="carto-positron", width=800,
        height=450) ❹
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
fig.show()
```

❶ Создаем массив размеров радиусов для маркеров-кругов.

❷ Функция scatter_mapbox принимает массив широты и долготы для размещения маркеров, а также массив их вычисленных радиусов (`size`). Уровень масштабирования (zoom=0.7) определяет стандартное положение камеры над картой.

❸ `hover_name` задает заголовок всплывающей подсказки, а `hover_data` добавляет информацию из столбца `Total`.

❹ Plotly предоставляет ряд бесплатных стилей карт на основе тайлов.

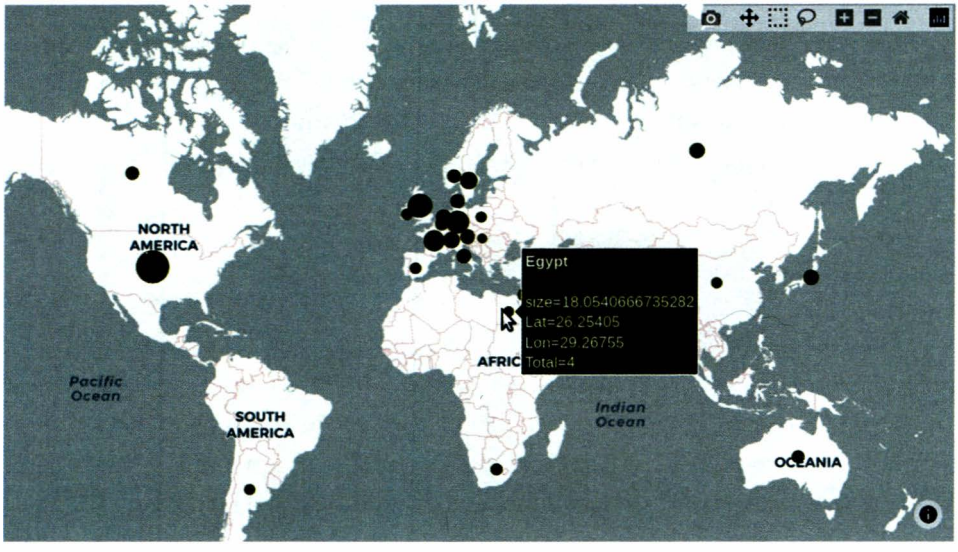


Рис. 14.3. *Быстрое создание карты с помощью Plotly Express*

Plotly предлагает похожий, но более мощный (чем для базовых диаграмм) вариант сопоставления данных и макета, который следует знакомому подходу: создается массив графиков *traces* и макет для настройки легенды, заголовков, масштаба карты и других параметров. Ниже показано, как создать карту для данных о нобелевских лауреатах:

SKLADCHIK.ORG

```
mapbox_access_token = "pk.eyJ1Ij...JwFsbg" ❶

df_cc = df_country_category

site_lat = df_cc.Lat ❷
site_lon = df_cc.Lon
totals = df_cc.Total
locations_name = df_cc.index

layout = go.Layout(
   title='Nobel prize totals by country',
   hovermode='closest',
   showlegend=False,
   margin ={'l':0,'t':0,'b':0,'r':0},
   mapbox=dict(
      accesstoken=mapbox_access_token,
      # здесь можно задать детали карты: центр, наклон
      # и ориентацию... поиграем.
#        bearing=0,
# #         center=dict(
# #            lat=38,
# #            lon= -94
# #         ),
#        pitch=0,
      zoom=0.7,
      style='light'
   ),
   width=875, height=450
)

traces = [

            go.Scattermapbox(
            lat=site_lat,
            lon=site_lon,
            mode='markers',
             marker=dict(
               size=totals.apply(calc_marker_radius, args=(7,)),
               color='olive',
```

```
            opacity=0.8
        ),
        text=[f'{locations_name[i]} won {int(x)} total prizes'\
            for i, x in enumerate(totals)], hoverinfo='text'
        )
]

fig = go.Figure(traces, layout=layout)
fig.show()
```

❶ Plotly предлагает ряд бесплатных наборов карт на основе OSM (OpenStreetMap), но для использования специфичных для Mapbox слоев требуется токен доступа. Для личного использования токены предоставляются бесплатно.

❷ Сохраняем столбцы и индекс DataFrame в более удобной форме.

Получившаяся карта показана на рисунке 14.4, слева. Обратите внимание на настраиваемую подсказку, всплывающую при наведении мыши. На правой части рисунка 14.4 показан результат взаимодействия с пользователем: изменение координат и масштаба для просмотра части карты с распределением премий по Европе.

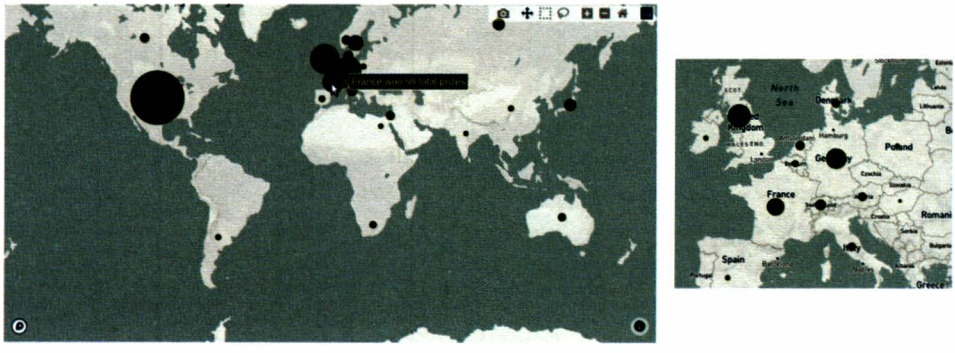


Рис. 14.4. *Картографирование с помощью графических объектов Plotly*

Давайте расширим возможности карты, добавив пользовательские элементы управления в виде кнопок для выбора номинаций премии.

Добавление пользовательских элементов управления с помощью Plotly

Одна из интересных функций интерактивных карт Plotly — возможность реализовать пользовательские элементы управления (custom controls) на Python с последующей их конвертацией в HTML+JS компоненты для веб-приложений. Управляющий API, на мой взгляд, обладает ограниченной гибкостью и поддерживает узкий набор элементов, однако сама возможность добавлять селекторы данных, ползунки, фильтры и прочее — большое преимущество. Добавим несколько кнопок на нашу карту, чтобы пользователь мог фильтровать набор данных по номинациям премии.

Прежде, чем продолжить, заменим в DataFrame пропущенные значения (NaN) премий по странам и номинациям на нули, чтобы исключить ошибки при маркировке (labeling errors). Эти изменения показаны в следующем коде:

```
df_country_category.head(2)
# Out:
# category    Chemistry  Economics  Literature Peace    Physics \
# country
# Argentina   1.0        NaN        NaN        2.0      NaN
# Australia   NaN        1.0        1.0        NaN      1.0
```

Одной строкой pandas заменяем все `NaN` на нули:

```
df_country_category.fillna(0, inplace=True)
```

Из-за этого потребуется немного другой паттерн Plotly, чем использовался до сих пор. Сначала мы создадим фигуру с макетом `layout`, а затем добавим трассировки данных с помощью `add_trace`, перебирая номинации премии и по ходу добавляя кнопки в массив `button`.

Затем с помощью метода `update` добавим эти кнопки к макету:

```
# ...
categories = ['Total', 'Chemistry', 'Economics', 'Literature',\
   'Peace', 'Physics','Physiology or Medicine',]
# …
colors = ['#1b9e77','#d95f02','#7570b3','#e7298a','#66a61e',
   '#e6ab02','#a6761d']
buttons = []
```

```
# ... ОПРЕДЕЛИТЬ МАКЕТ, КАК ПРЕЖДЕ
fig = go.Figure(layout=layout)
default_category = 'Total'

for i, category in enumerate(categories):
   visible = False
   if category == default_category: ❶
      visible = True
      fig.add_trace(
         go.Scattermapbox(
            lat=site_lat,
            lon=site_lon,
            mode='markers',
            marker=dict(
               size=df_cc[category].apply(calc_marker_radius,
                           args=(7,)),
               color=colors[i],
               opacity=0.8
            ),
            text=[f'{locations_name[i]} prizes for {category}: \
                  {int(x)}' for i, x in enumerate(df_cc[category])],
            hoverinfo='text',
            visible=visible
            ),
   )
   # Начинаем с массива mask из логических значений False для всех
   # номинаций (включая Total)
   # В Python [True] * 3 == [True, True, True]
   mask = [False] * len(categories)
   # Устанавливаем индекс маски текущей номинации в True
   # т. е. у кнопки 'Chemistry' маска [False, True, False, False,
   # False, False]
   mask[categories.index(category)] = True
   # теперь используем логическую маску, чтобы добавить кнопку
   # в список button
   buttons.append(
            dict(
                  label=category,
```

```
                method="update",
                args=[{"visible": mask}], ❷
            ),
    )

    fig.layout.update( ❸
        updatemenus=[
            dict(
                type="buttons",
                direction="down",
                active=0,
                x=0.0,
                xanchor='left',
                y=0.65,
                showactive=True, # показывает последнее нажатие кнопки
                buttons=buttons
            )
        ]
    )

    fig.show()
```

❶ Наборы маркеров номинаций изначально невидимы — по умолчанию отображается только `Total`.

❷ Используем маску, чтобы задать массив видимости, который обеспечивает отображение маркеров номинации, соответствующей выбранной кнопке.

❸ Настраиваем позиционирование группы кнопок вертикально (`x=0.0`, `y=0.65`) с фиксацией блока кнопок по левому краю.

При нажатии на кнопку становятся видны маркеры, соответствующие номинации (см. рисунок 14.5), благодаря маске видимости кнопки. Хотя способ выглядит немного неуклюжим, он обеспечивает надежную фильтрацию данных. В стиле этих кнопок мало что можно изменить в отличие от элементов управления на JavaScript+HTML, которые мы рассмотрим далее в этой главе.



Рис. 14.5. *Добавление пользовательских элементов управления к карте Plotly*

Из Notebook в веб-формат с помощью Plotly

Теперь рассмотрим, как перенести диаграммы Plotly из Notebook в компактное веб-представление. Для генерации встраиваемого HTML+JS кода используем функцию `plot` из модуля `offline` библиотеки Plotly. Импортируем ее:

```
from plotly.offline import plot
```

С помощью `plot` создадим из фигуры встраиваемую строку, которую можно будет напрямую отправить в браузер. Она содержит необходимые HTML-теги и JavaScript-код для подключения библиотеки Plotly.js и построения диаграммы:

```
embed_string = plot(fig, output_type='div', include_plotlyjs="cdn")
embed_string
#'<div> <script type="text/javascript">window.PlotlyConfig
#= {MathJaxConfig: \'local\'};</script>\n
#<script src="https://cdn.plot.ly/ #plotly-2.9.0.min.js"></script>
#<div id="195b2d71-f59d-4f8a-a40a-3b8c797a918b"
#class="plotly-graph-div" #style="height:600px; width:600px;">
#</div> <script type="text/ #javascript"> […] </script>
#</div>'
```

Приведем эту строку в порядок и увидим четыре компонента — HTML-тег `div` с ID диаграммы и код скрипта, содержащий вызов `newPlot` с данными и макетом, переданными в качестве параметров:

```
<div>
   <!– (1) Конфигурация Plotly.js, размещенная в файле JavaScript
      (.js) –>
   <script type="text/javascript">
      window.PlotlyConfig = {MathJaxConfig: 'local'};
   </script>
   <!– (2) Поместить в нижнюю часть HTML-файла для импорта
      библиотеки Plotly из облака (CDN) –>
   <script src="https://cdn.plot.ly/plotly-2.9.0.min.js"></script>
   <!– создать div контента для диаграммы, с id (для распаковки)
      (3)!! Поместим этот div в CodePen, в подраздел для HTML!! –>
   <div id="4dbeae4f-ed9b-4dc1-9c69-d4bb2a20eaa7"
   class="plotly-graph-div" style="height:100%; width:100%;"></div>

   <script type="text/javascript">
      // (4) Содержимое этого тега переносим в JavaScript-файл
      (.js) –>
      window.PLOTLYENV=window.PLOTLYENV || {};
      // Берем расположенный выше тег 'div' по ID и вызываем Plotly
      // JS API, используя встроенные данные и метаданные
      if (document.getElementById(
                  "4dbeae4f-ed9b-4dc1-9c69-d4bb2a20eaa7"))
      {     Plotly.newPlot("4dbeae4f-ed9b-4dc1-9c69-d4bb2a20eaa7",
         [{"mode":"lines","name":"Korea, South",
         "x": [0,1,2,3,4,5,6,7,8,9,10,11,12,...]}])
         };
   </script>
</div>
```

Хотя можно просто вставить HTML+JS в веб-страницу для отображения диаграммы, на практике лучше JS и HTML разделить. Сначала разместим div диаграммы на веб-странице с несколькими заголовками, добавим несколько контейнеров для информационных полей и др.:

```
<!– index.html –>
<div class="main">
   <h1 class='title'>The Nobel Prize</h1>
   <h2>From notebook to the web with Plotly</h2>

   <div class="intro">
      <p>Some nuggets of data mined [...]</p>
   </div>

   <div class="container" id="by-country-container">

      <div class="info-box">
         <p>This chart shows the cumulative Nobel prize wins by
         region, taking the two or three highest winning countries
         from each.[...]</p>
      </div>

      <div class="chart-wrapper" id="by-country">
         <div id="bd54c166-3733-4b20-9bb9-694cfff4a48e" ❶
         class="plotly-graph-div" style="height:100%;
         width:100%;"></div>
      </div>
   </div>
</div>

<script scr="scripts/plotly_charts.js"></script>
<script src="https://cdn.plot.ly/plotly-2.9.0.min.js"></script> ❷
```

❶ Контейнер `div` для контента, созданного Plotly, с ID, соответствующим JavaScript-диаграмме.

❷ Тег `script` со ссылкой для подключения библиотеки Plotly.js.

Переносим содержимое двух оставшихся тегов JavaScript в файл *plotly_charts.js*:

```
// scripts/plotly_charts.js
   window.PLOTLYENV=window.PLOTLYENV || {};
   if (document.getElementById(
               "bd54c166-3733-4b20-9bb9-694cfff4a48e")){ ❶
   Plotly.newPlot("bd54c166-3733-4b20-9bb9-694cfff4a48e", ❷
```

```
        [{"hovertemplate":"N. America<br>%{x}<br>$%{y}
                <extra></extra>", ❸
        "mode":"lines","name":"N. America",
        "x":[1901,1902,1903,1904,1905,1906,1907,1908,1909,1910,1911,
             1912,1913, 1914,1915,1916,1917,1918,1919,1920,1921,1922,
             ...]
        }])
}
```

❶ Проверяем наличие `div` с правильным идентификатором.

❷ С помощью метода `Plotly.newPlot` строим диаграмму в указанном контейнере.

❸ Список объектов диаграммы, в данном случае содержащий все данные (массивы `x` и `y`), необходимые для переноса диаграммы из блокнота на веб-страницу.

При загрузке страницы метод `newPlot` библиотеки Plotly принимает данные, макет и диаграмму, созданную на JS в контейнере `div` с указанным ID. Полученная веб-страница показана на рисунке 14.6.

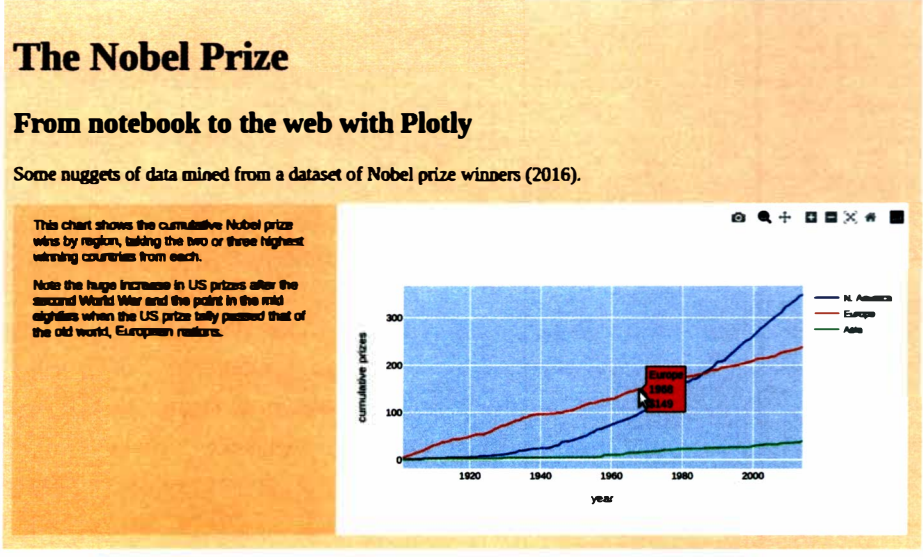


Рис. 14.6. *Из Notebook в веб-формат с помощью Plotly*

Таким способом можно передать на веб-страницу все диаграммы, созданные с помощью Plotly на языке Python. Если вы планируете использовать более двух

диаграмм, я рекомендую создавать для каждой свой JS-файл, так как встроенные данные могут значительно увеличить размер файлов.

Создание нативных JavaScript-диаграмм с помощью Plotly

Возможность легко перенести нужную диаграмму из Notebook в веб-приложение — это удобно, но для внесения изменений придется постоянно переключаться между ними. Через некоторое время это начнет раздражать. Один из плюсов Plotly — работая с ней, вы, можно сказать, бесплатно изучаете JS-библиотеку для построения диаграмм. Шаблоны фигур Python и JS очень похожи, поэтому преобразовать код диаграммы из Python в JS так же легко, как начать писать диаграммы на JS с нуля. Давайте для иллюстрации преобразуем диаграмму seaborn в диаграмму Plotly.js.

Мы уже создавали скрипичные диаграммы с помощью seaborn (см. «Возраст на момент получения премии» на стр. 373). Чтобы преобразовать ее для веб-страницы, нам потребуются данные. Удобный способ переноса небольшого набора данных: нужно преобразовать его в формат JSON, скопировать полученную строку и вставить ее в JS-файл, а затем преобразовать строку в объект JS. Сначала с помощью pandas создадим набор данных только со столбцами `award_age` и `gender`, а затем произведем требуемый массив JSON-объектов:

```
df_select = df[['gender', 'award_age']]
df_select.to_json(orient='records')
#'[{"gender":"male","award_age":57},{"gender":"male","award_age":80},
#{"gender":"male","award_age":49},{"gender":"male","award_age":59},
#{"gender":"male","award_age":49},{"gender":"male","award_age":46},…]'
```

Вставим строку JSON в JS-файл и применим встроенную JSON-библиотеку, чтобы преобразовать строку в массив JS-объектов:

```
let data = JSON.parse('[{"gender":"male","award_age":57}
',{"gender":"male","award_age":80},'
'{"gender":"male","award_age":49},{"gender":"male","award_age":59},'
'{"gender":"male","award_age":48}, ...]')
```

SKLADCHIK.ORG

Подготовив данные, создадим простой HTML-каркас с контейнером (с ID `award_age`) для диаграммы Plotly:

```
<div class="main">
   <h1 class='title'>The Nobel Prize</h1>
   <!- ... ->
      <div class="chart-wrapper">
         <div class='chart' id='award_age'></div> ❶
      </div>

</div>

<script src="https://cdn.plot.ly/plotly-2.9.0.min.js"></script>
```

❶ Используем ID этого контейнера, чтобы указать Plotly, где строить диаграмму.

Теперь построим нашу первую диаграмму Plotly.js. Паттерн совпадает с тем, что мы видели на графиках Python-библиотеки Plotly в Notebook. Сначала создадим массив данных (`traces`) с объектами диаграммы, а затем макет, чтобы задать заголовки, метки, цвет и др. После этого построим диаграмму с помощью `newPlot`. Главное отличие от Python — первым параметром метод `newPlot` принимает ID контейнера, внутри которого будет строиться диаграмма:

```
var traces = [{ ❶
   type: 'violin',
   x: data.map(d => d.gender), ❷
   y: data.map(d => d.award_age),
   points: 'none',
   box: {
      visible: true
   },
   line: {
      color: 'green',
   },
   meanline: {
      visible: true
   },
}]
```

```
var layout = {
   title: "Nobel Prize Violin Plot",
   yaxis: {
   zeroline: false
   }
}

Plotly.newPlot('award_age', traces, layout); ❸
```

❶ Как и принято в Plotly, сначала создаем массив объектов диаграммы.

❷ Для создания массивов пола и возраста лауреатов используем метод JS-массива `map` со стрелочными функциями.

❸ Plotly отрисует диаграмму в контейнере `div` с идентификатором `award_age`.

Полученную скрипичную диаграмму см. на рисунке 14.7.

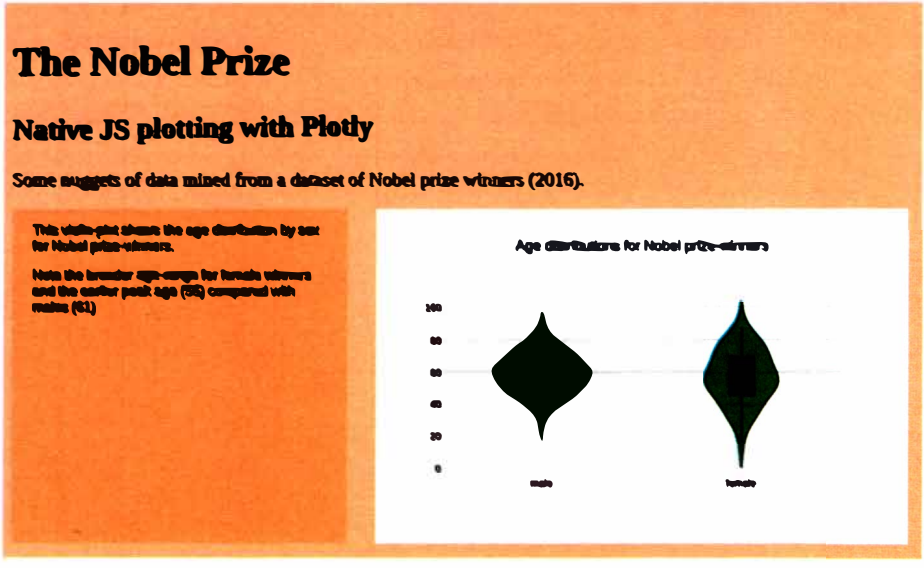


Рис. 14.7. *Скрипичная диаграмма, построенная с помощью Plotly.js*

Как видите, Plotly JS API соответствует Plotly Python API и, к слову, более лаконичен. Разделение доставки данных и построения диаграмм упрощает работу с кодом и позволяет выполнять настройку без возврата к Python API. Парсинг строк JSON оптимально применять для быстрой обработки небольших наборов данных.

Для работы с крупными наборами данных и полноценного использования веб-возможностей JS рекомендуется стандартный подход к доставке данных через JSON-файлы[1]. Это обеспечивает максимальную гибкость при создании визуализаций наборов данных мегабайтного масштаба.

Извлечение данных из файлов JSON

Другой способ загрузки данных на веб-страницы — экспортировать DataFrame в JSON, извлечь с помощью JavaScript, выполнить всю необходимую дальнейшую обработку, а затем передать данные в нативную библиотеку диаграмм JS (или в D3). Этот гибкий рабочий процесс обеспечивает максимальную свободу для визуализации данных с помощью JS.

Разделение функций — Python применяется для обработки данных, JavaScript для их визуализации — идеально подходит для создания масштабных веб-визуализаций и является стандартом индустрии.

Для начала с помощью специального метода сохраним DataFrame с данными о лауреатах Нобелевской премии в JSON. Как правило, мы хотим получить данные в виде массивов объектов, для чего используется параметр `orient='records'`:

```
df.to_json('nobel_winners.json', orient='records')
```

Используем полученный JSON, чтобы построить диаграмму Plotly с помощью JavaScript API, как в предыдущем подразделе. Напишем HTML-каркас, включающий в себя контейнер (с ID), в котором будет строиться диаграмма, и тег script для импорта JS:

```
<!- index.html ->
<link rel="stylesheet" href="styles/index.css">

<div class="main">
   <h1 class='title'>The Nobel Prize</h1>
   <!- ... ->
      <div class="chart-wrapper">
         <div class='chart' id='gender-category'> </div> ❶
      </div>
```

[1] Еще один путь для продвинутых визуализаций больших наборов данных — использование data-сервера с API.

```
    </div>
</div>

<script src="https://cdn.plot.ly/plotly-2.9.0.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/d3/7.4.2/
            d3.min.js"></script>
<script src="scripts/index.js"></script> ❷
```

❶ С помощью ID указываем Plotly, где строить диаграмму.
❷ Импортируем файл index.js из папки scripts.

В точке входа JS мы используем служебный метод `json` из библиотеки D3 для импорта набора данных о лауреатах Нобелевской премии и преобразования его в массив объектов JS. Затем мы передаем данные в функцию `makeChart`, и Plotly приступает к работе:

```
// scripts/index.js
d3.json('data/nobel_winners.json').then(data => {
   console.log("Dataset: ", data)
   makeChart(data)
})
```

В консоли видим массив лауреатов:

```
[
   {category: 'Physiology or Medicine', country: 'Argentina',
    date_of_birth: -1332806400000, date_of_death: 1016928000000,
    gender: 'male', ...},
   {category: 'Peace', country: 'Belgium',
    date_of_birth: -4431715200000, date_of_death: -1806278400000,
    gender: 'male', ...}
    [...]
]
```

В функции `makeChart` мы используем очень удобный метод d3.rollup для группирования набора данных лауреатов по полу и номинации, а затем получаем размеры групп на основе длины массивов из возвращенных элементов:

```
function makeChart(data) {
   let cat_groups = d3.rollup(data, v => v.length, ❶
           d=>d.gender, d=>d.category)
   let male = cat_groups.get('male')
   let female = cat_groups.get('female')
   let categories = [...male.keys()].sort() ❷

   let  traceM = {
      y: categories,
      x: categories.map(c => male.get(c)), ❸
      name: "male prize total",
      type: 'bar',
      orientation: 'h'
   }
   let traceF= {
      y: categories,
      x: categories.map(c => female.get(c)),
      name: "female prize total",
      type: 'bar',
      orientation: 'h'
   }
   let traces = [traceM, traceF]
   let layout = {barmode: 'group', margin: {l:160}} ❹

   Plotly.newPlot('gender-category', traces, layout)
}
```

❶ Сначала `rollup` группирует объекты из массива лауреатов по полу, затем по номинациям и наконец возвращает размеры/длину полученных групп как JS Map: `{male: {Physics: 199, Economics: 74, ...}, female: {...}}`.

❷ Используем JS-оператор spread (...), чтобы развернуть из ключей номинаций массив, который затем сортируется для получения значений Y горизонтальной столбчатой диаграммы.

❸ Сопоставляем отсортированные по алфавиту номинации с размерами их групп, чтобы получить высоту столбиков диаграммы.

❹ Увеличиваем внешний левый отступ (margin) диаграммы, чтобы поместились длинные подписи.

Полученная столбчатая диаграмма показана на рисунке 14.8. Имея полный набор данных о лауреатах Нобелевской премии, можно быстро создать серию диаграмм без переключения между Python и JS. Этот Plotly API интуитивно понятен, его просто освоить, поэтому он отлично дополняет набор инструментов для визуализации данных. Рассмотрим, как можно легко и просто его расширить с помощью пользовательских элементов управления на HTML+JS.

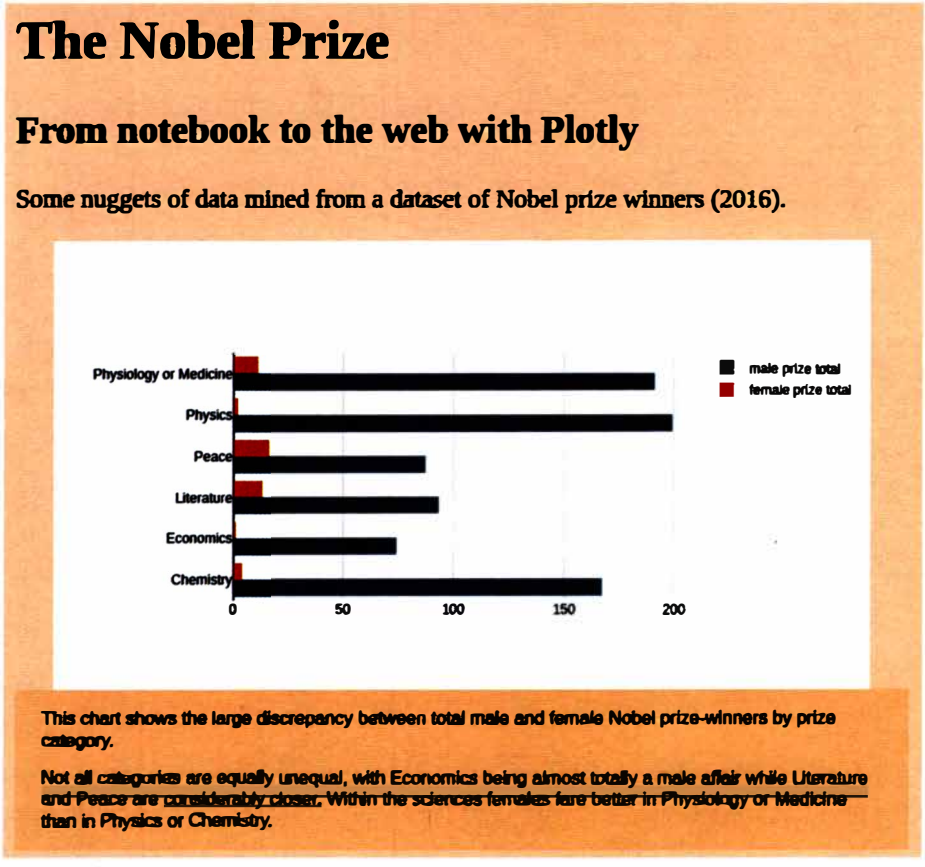


Рис. 14.8. *Столбчатая диаграмма, созданная с помощью Plotly.js*

Интерактивная визуализация Plotly с помощью JavaScript и HTML

Как мы узнали из «Добавления пользовательских элементов управления с помощью Plotly», библиотека позволяет добавлять кнопки и выпадающие списки на Python, которые экспортируются в веб-страницу как JS-управляемые

HTML-элементы. Однако у Plotly довольно ограниченные возможности для позиционирования и стилизации таких виджетов. Гораздо больше возможностей для изменения веб-диаграмм Plotly, фильтрации наборов данных или выбора стиля дает использование элементов управления, с нуля написанных на JS и HTML. Они представляют собой более гибкую и мощную структуру управления, а сделать их довольно просто — достаточно немного разбираться в JS.

Проиллюстрируем этот подход на примере диаграммы из подраздела «Создание нативных JavaScript-диаграмм с помощью Plotly». Добавим выпадающий список, который позволит пользователю сменить группу, отображаемую на оси X. Два очевидных варианта — это существующее распределение по полам и группирование возрастов по номинациям премии.

Сначала добавим выпадающий список HTML (`select`) на страницу и отцентрируем его с помощью CSS-свойства flex:

```
<div class="main">
   <h1 class='title'>The Nobel Prize</h1>
   <h2>From notebook to the web with Plotly</h2>

   <!- ... ->
   <div class="container">
   <!- ... ->
      <div class="chart-wrapper">
         <div class='chart' id='violin-group'> </div>
      </div>
   </div>

   <div id="chart-controls">

      <div id="nobel-group-select-holder">
         <label for="nobel-group">Group:</label>
         <select name="nobel-group" id="nobel-group"></select> ❶
      </div>

   </div>

</div>
```

❶ Тег `select` может содержать теги `option`, которые будут добавлены с помощью JS и D3.

Код CSS центрирует элементы управления в контейнере `controls` и задает стиль шрифта:

```
#chart-controls {
   display: flex;
   justify-content: center;
   font-family: sans-serif;
   font-size: 0.7em;
   margin: 20px 0;
}

select {
   padding: 2px;
}
```

Теперь, имея базовую HTML-структуру, мы используем JS и D3 для добавления тегов `select` в элементы управления группами. Перед этим модифицируем функцию построения диаграмм Plotly, чтобы она обновлялась с помощью новых групп. Данные JSON импортируются, как и раньше, но теперь они хранятся в локальной переменной и используются для обновления скрипичной диаграммы Plotly. Для создания диаграммы функция `updateChart` применяет метод `Plotly` update. Это метод работает подобно `newPlot`, но вызывается при изменении данных или макета и эффективно перерисовывает диаграмму, отражая любые изменения.

У нас также есть новая переменная `selectedGroup`, которая указывает, какое поле выпадающего списка `select` будет отображаться:

```
let data
d3.json("data/nobel_winners.json").then((_data) => {
   console.log(_data);
   data = _data
   updateChart();
});

let selectedGroup = 'gender' ❶

function updateChart() {
   var traces = [
      {
```

```
            type: "violin",
            x: data.map((d) => d[selectedGroup]), ❶
            y: data.map((d) => d.award_age),
            points: "none",
            box: {
               visible: true
            },
            line: {
               color: "green"
            },
            meanline: {
               visible: true
            }
         }
      ];

      var layout = {
         title: "Age distributions of the Nobel prizewinners",
         yaxis: {
            zeroline: false
         },
         xaxis: {
            categoryorder: 'category ascending' ❷
         }
      };

      Plotly.update("violin-group", traces, layout); ❸
   }
```

❶ Переменная `selectedGroup` позволяет пользователю изменить группу оси X.

❷ Чтобы призовые группы шли в алфавитном порядке (начиная с *Chemistry*), вносим изменение в макет.

❸ Вместо newPlot вызываем `update`, который имеет ту же сигнатуру, но отражает изменения данных (`traces`) или объекта макета.

Имея метод `updateChart`, добавим параметры отбора и коллбэк-функцию, которая будет вызываться при изменении пользователем группы премии:

```
let availableGroups = ['gender', 'category']
availableGroups.forEach((g) => {
   d3.select("#nobel-group") ❶
   .append("option")
   .property("selected", g === selectedGroup) ❷
   .attr("value", g)
   .text(g);
});

d3.select("#nobel-group").on("change", function (e) {
   selectedGroup = d3.select(this).property("value"); ❸
   updateChart();
});
```

❶ Для каждой доступной группы с помощью D3 находим выпадающий список по ID и добавляем в конец тег `<option>`, где текст и значение соответствуют названию группы.

❷ Это выражение гарантирует, что первоначальный выбор будет соответствовать значению `selectedGroup` с помощью установки атрибута `selected` в значение true.

❸ Используем D3, чтобы при выборе параметра вызвать коллбэк-функцию. Получаем значение параметра (*gender* или *category*) и присваиваем его переменной `selectedGroup`. Затем обновляем диаграмму, чтобы отразить это изменение.

После настройки связей выпадающий список позволяет менять отображение скрипичной диаграммы в соответствии с выбранной группой. На рисунке 14.9 показан результат выбора группы category. Обратите внимание, что Plotly автоматически поворачивает названия групп номинаций, чтобы они не накладывались друг на друга.

В следующих главах мы рассмотрим другие HTML-элементы управления в действии — кнопки и радиокнопки. Создание элементов управления на JS значительно гибче Python-решений, но требует некоторых навыков веб-разработки.

SKLADCHIK.ORG

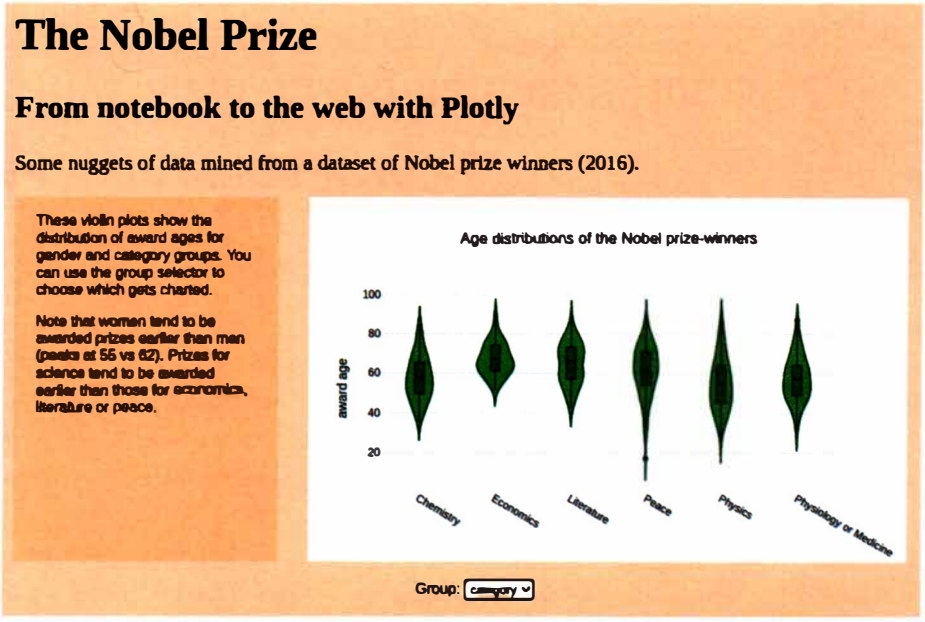


Рис. 14.9. *Добавление выпадающего списка для управления диаграммой Plotly*

Резюме

В этой главе мы рассмотрели, как превратить диаграммы, которые мы исследовали в Notebook, в веб-представление. Доступные опции варьируются от статических PNG со стилизацией Matplotlib до интерактивных Plotly-диаграмм с кастомными JavaScript-элементами управления. Данные можно интегрировать через вызовы Plotly (с использованием офлайн-библиотеки) или импортировать как JSON-строку (для набросков) либо файл.

Plotly — хорошая библиотека для построения графиков, к тому же есть дополнительный плюс: изучая ее Python API, вы параллельно получаете знания о JS API. Для обычных и некоторых специализированных (например, машинного обучения) диаграмм эта библиотека — отличный вариант. Однако для более сложных или уникальных задач стоит использовать библиотеку D3, предлагающую, как мы увидим в следующих главах, гораздо более мощные инструменты.

Разработка концепции визуализации Нобелевской премии

В главе 13 мы исследовали набор данных о лауреатах Нобелевской премии, пытаясь найти интересные аспекты данных, которые могли бы увлечь и просветить аудиторию. Мы выявили несколько интересных фактов, среди которых:

— всего две женщины получили Нобелевскую премию по физике — Мария Кюри и Мария Гепперт-Майер;
— после Второй мировой войны число Нобелевских лауреатов из США резко возросло, обогнав прежних лидеров: Великобританию, Германию и Францию;
— число полученных премий различается на разных континентах;
— если подсчитать процент премий на душу населения, то лидируют скандинавские страны.

Эти и другие нарративы требуют специфических типов визуализации. Сравнение количества Нобелевских премий по странам лучше всего отображать с помощью классической столбчатой диаграммы, а для отражения географического распределения премий потребуется карта. В этой главе мы попытаемся спроектировать современную интерактивную визуализацию, которая объединит ключевые истории, обнаруженные при анализе данных.

Для кого эта визуализация?

Первое, что следует определить при создании визуализации, — ее целевая аудитория. Визуализация для музейной экспозиции будет существенно отличаться от корпоративного дашборда, даже при использовании одного и того же набора данных. Особенность визуализации Нобелевской премии в этой книге в том, что она является учебным пособием по ключевым аспектам D3 и JavaScrip, необходимым для создания современной интерактивной веб-визуализации. Это неформальная визуализация, призванная развлекать и информировать. Для нее не требуется особой аудитории.

Выбор визуальных элементов

Первое условие — визуализация данных о нобелевских лауреатах должна быть достаточно простой для обучения ключевым навыкам работы с D3. Впрочем, масштаб охвата стоит ограничивать для любой визуализации. Охват во многом зависит от контекста[1], но в большинстве случаев чем проще обучение, тем лучше. Избыток интерактивных элементов может перегрузить пользователя и ослабить восприятие информации.

Не забывая об этом, определим основные элементы визуализации и их визуальную компоновку.

Обязательно должна присутствовать строка меню, позволяющая взаимодействовать с визуализацией и управлять данными. Функциональность меню будет зависеть от выбранных нами историй, но оно должно предоставлять инструменты для исследования или фильтрации данных.

Идеально, если визуализация будет отображать премии по годам и динамически обновляться при изменении данных через меню. Для анализа национальных/региональных тенденций включим в визуализацию карту с подсветкой стран, граждане которых получили Нобелевскую премию, и количественными показателями. Столбчатая диаграмма — оптимальный способ сравнения количества премий по странам. Она должна динамически адаптироваться к любым изменениям данных. Также должна быть возможность выбора между абсолютным числом премий и их количеством на душу населения.

Пользователи также смогут выбирать сведения о конкретных лауреатах с фотографией (при наличии) и краткой биографией, полученные путем скрейпинга из Википедии. Нам потребуется текущий отфильтрованный список лауреатов и окно для отображения деталей.

Перечисленных элементов достаточно для отражения ключевых сюжетов из предыдущей главы. После доработки они впишутся в типичный форм-фактор[2].

Визуализация использует фиксированный размер для всех устройств: мы жертвуем возможностью адаптации под большие экраны ради совместимости со смартфонами/планшетами последнего поколения. Я пришел к выводу, что фиксированный размер обеспечивает точный контроль над расположением визуальных блоков, информационных полей и др. Но для некоторых визуализаций

[1] Эксперты, для которых предназначаются специализированные дашборды, более терпимы к большому количеству функциональных элементов, чем пользователи универсальной образовательной визуализации.

[2] Важно учитывать изменение разрешения экранов устройств (размер в пикселях). На май 2022 года большинство устройств поддерживали визуализации размером 1000×800 пикселей.

требуется иной подход, особенно для многоэлементных дашбордов. Адаптивный веб-дизайн (RWD) динамически подстраивает визуализацию под размеры окна браузера. Некоторые популярные CSS-библиотеки типа Bootstrap адаптируют стили под размер экрана (например, 1280×800 пикселей) для оптимального использования пространства. Фиксированный размер визуализации и абсолютное позиционирование гарантируют точность расположения визуальных элементов. Однако надо помнить о сложностях RWD, особенно когда создаешь многокомпонентные дашборды и тому подобное.

Давайте попробуем определить внешний вид нашей визуализации и требования к отдельным ее элементам, начав с главного пользовательского элемента управления — строки меню.

Строка меню

Интерактивной визуализацией управляет пользователь, выбирая опции, нажимая кнопки, управляя ползунками и тому подобное. Эти элементы определяют область визуализации, поэтому рассмотрим их первыми. Пользовательские элементы управления размещаются на верхней панели инструментов.

Стандартный способ подтолкнуть пользователя к интересным открытиям — предоставить возможность фильтровать данные по ключевым параметрам. Для нашей визуализации это пол, номинация и страна. По этим критериям мы и анализировали данные в прошлой главе. Фильтры должны быть кумулятивными, например, если выбрать «женский пол + номинация "Физика"», то будут возвращены две женщины-лауреата. В дополнение к этим фильтрам нам нужны радиокнопки для переключения между абсолютным числом лауреатов и их количеством на душу населения.

На рисунке 15.1 показана строка меню, соответствующая перечисленным требованиям. Верхняя панель содержит выпадающие списки (номинация, пол, страна) и радиокнопки для переключения между абсолютным числом лауреатов и их количеством на душу населения.

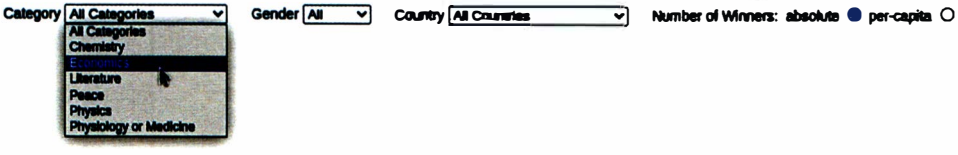


Рис. 15.1. *Пользовательские элементы управления*

Панель меню располагается над основным компонентом визуализации — диаграммой, отображающей историю вручения Нобелевских премий. Она будет описана далее.

Распределение премии по годам

В предыдущей главе были показаны значимые исторические тенденции распределения Нобелевских премий по странам. Мы также выяснили, что хотя в последнее время число женщин-лауреатов возросло, они существенно отстают в естественнонаучных номинациях. Для выявления этих трендов отобразим все Нобелевские премии на временной шкале с возможностью фильтрации по полу, стране и категории (с помощью строки меню).

При ширине визуализации в 1000 пикселей на каждый год 114-летней истории премии придется примерно 8 пикселей — этого достаточно. Максимальное количество премий за год (14 в 2000 году) определяет минимальную высоту элемента как 14×8 пикселей (~112px). Для изображения отдельных вручений премии подойдут кружки с цветовой кодировкой номинаций. В результате получаем диаграмму, как на рисунке 15.2.

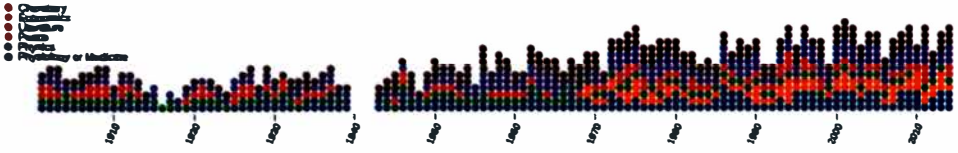


Рис. 15.2. *Число вручений Нобелевской премии по годам. Каждой номинации соответствует свой цвет*

Распределение отдельных премий — суть визуализации, поэтому мы разместим хронологическую диаграмму на видном месте над картой, которая подчеркивает международный характер премии и позволяет анализировать любые глобальные тенденции.

Карта, показывающая выборку стран нобелевских лауреатов

Картографирование — одна из ключевых возможностей D3, поддерживающая различные проекции (от Меркатора до 3D-сфер)[1]. Карты, несомненно, интересны, но используются излишне часто, так что советую применять их только к геоданным. Например (если вы не будете осторожны), географические области, занимающие большую площадь, такие как США или страны Европы, перевесят области с меньшей площадью, даже если в последних выше численность населения. При представлении демографических данных трудно избежать этого перекоса, что может привести к искажению информации[2].

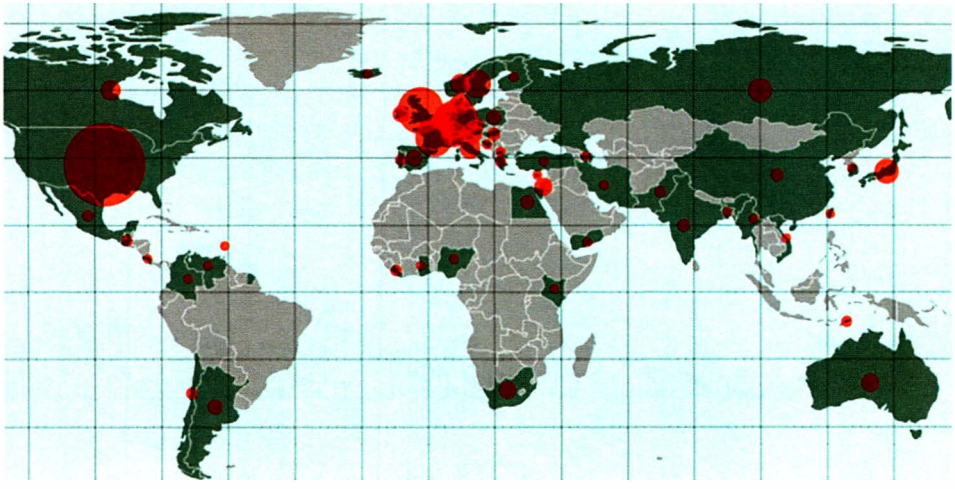


Рис. 15.3. *Распределение премий по миру*

Но поскольку Нобелевская премия является международной, то интересно узнать о распределении ее лауреатов по континентам, так что карта мира хорошо подходит для отображения отфильтрованных данных. Размещенные в центре стран маркеры-круги, чей размер отражает число лауреатов (абсолютное или на душу населения), минимизируют искажения данных в сторону более крупных массивов суши. В сравнительно небольших европейских странах эти маркеры

[1] Эти ортографические проекции трехмерного изображения на экран являются «подделками» в том смысле, что для них не используются технологии типа WebG L. Замечательные примеры см. на сайтах Джейсона Дэвиса (Jason Davies), observablehq и nullschool.

[2] Пример см. на xkcd.

будут пересекаться. Придав им легкую прозрачность[1], мы сможем видеть наложенные друг на друга круги, а добавив непрозрачности, создадим ощущение плотности премии, как показано на рисунке 15.3.

Добавим к карте всплывающую подсказку, чтобы продемонстрировать процесс создания этого удобного визуального компонента, а также отображать названия стран. На рисунке 15.4 показано, что мы собираемся сделать.



Рис. 15.4. *Простая всплывающая подсказка для карты Нобелевской премии*

Ниже карты будет размещен последний крупный элемент визуализации — столбчатая диаграмма, позволяющая пользователю наглядно сравнивать количество нобелевских премий в разных странах.

Столбчатая диаграмма, показывающая количество лауреатов по странам

Многочисленные исследования подтверждают, что столбчатые диаграммы идеально подходят для сравнения числовых данных[2]. Настраиваемая столбчатая диаграмма повышает гибкость визуализации, позволяя отражать результаты фильтрации данных, выбор метрик (например, абсолютные значения или на душу населения) и другие параметры.

На рисунке 15.5 показана столбчатая диаграмма, с помощью которой мы будем сравнивать выборку премий для указанных стран. На действия пользователя через строку меню (см. рисунок 15.1) должны динамически реагировать как деления осей, так и столбики диаграммы. Для этого хорошо было бы применить анимированный переход между состояниями столбчатой диаграммы и (как мы

[1] Это можно сделать, используя альфа-канал в коде RGBA и CSS-свойство opacity, выбирая значение от 0 (полная прозрачность) до 1 (полная непрозрачность).

[2] См. содержательный пост в блоге Стивена Фью (Stephen Few) https://www.perceptualedge.com/blog/?p=1492.

увидим в разделе «Переходы» на стр. 432) эта возможность практически бесплатно поставляется с D3.js. Помимо того, что такие переходы привлекательно выглядят, есть основания полагать, что они являются эффективным средством коммуникации. Подробнее об эффективности анимированных переходов при визуализации данных см. этот документ Стэнфордского университета.

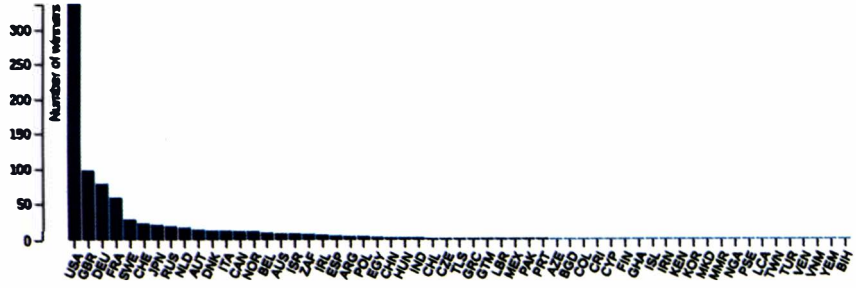


Рис. 15.5. *Компонент столбчатой диаграммы*

Сбоку от карты и столбчатой диаграммы мы разместим список выбранных на данный момент лауреатов и поле с биографией, что позволит пользователю кое-что узнать о личности лауреатов.

Список выбранных лауреатов

Мы хотим, чтобы пользователь мог выбирать отдельных лауреатов и просматривать краткую биографию и фотографию (при ее наличии). Проще всего использовать элемент управления списка (list box), отображающий текущую выборку лауреатов, отфильтрованных из полного набора данных с помощью строки меню. Разумный вариант по умолчанию — сортировка списка по году в порядке убывания. Хотя мы могли бы разрешить сортировку списка по столбцам, это представляется излишним.

Нам будет достаточно простой HTML-таблицы с заголовками столбцов. Как она выглядит, показывает рисунок 15.6.

Selected winners

Year	Category	Name
2014	Physics	Shuji Nakamura
2014	Chemistry	Eric Betzig
2014	Chemistry	William E. Moerner
2014	Physiology or Medicine	John O'Keefe
2014	Peace	Malala Yousafzai
2014	Physiology or Medicine	Edvard Moser
2014	Physiology or Medicine	May-Britt Moser
2014	Physics	Isamu Akasaki
2014	Physics	Hiroshi Amano
2014	Peace	Kailash Satyarthi
2014	Chemistry	Stefan Hell
2014	Literature	Patrick Modiano
2014	Economics	Jean Tirole
2013	Chemistry	Arieh Warshel
2013	Chemistry	Michael Levitt
2013	Chemistry	Martin Karplus
2013	Physiology or Medicine	Randy Schekman
2013	Physiology or Medicine	Thomas C. Südhof
2013	Physiology or Medicine	James Rothman
2013	Economics	Eugene F. Fama

Рис. 15.6. *Список выбранных лауреатов*

Строки списка кликабельные, так что пользователь может выбрать из списка любого лауреата, информация о котором появится в нашем последнем элементе — биографическом блоке.

Блок мини-биографии с фотографией

Нобелевская премия вручается личностям, о каждой из которых есть что рассказать. Чтобы сделать визуализацию более человечной и обогатить ее, используем мини-биографии и изображения, которые мы извлекли из Википедии (см. главу 6), для каждого лауреата из нашего списка.

На рисунке 15.7 показана область биографии с верхней границей, окрашенной в цвет номинации премии, с фотографией в правом верхнем углу и первыми абзацами из биографической записи в Википедии. Цвета номинаций совпадают с цветами на временной диаграмме (рисунок 15.2).

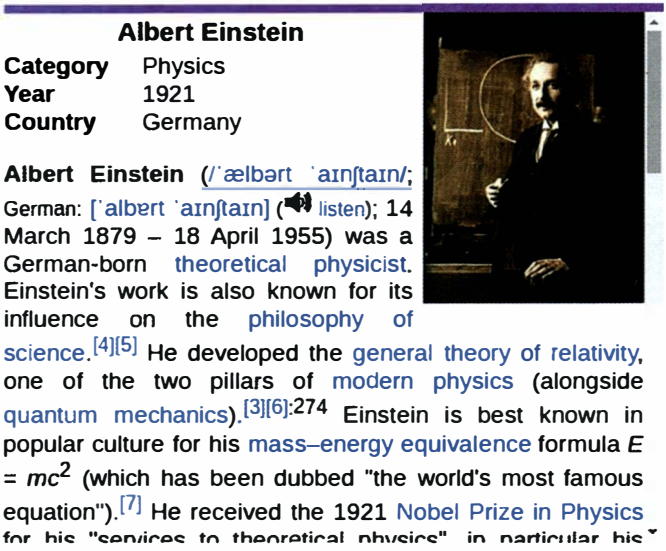


Рис. 15.7. *Краткая биография и фото выбранного лауреата*

Биографический блок — последний визуальный компонент из нашего набора. Теперь объединим компоненты в области окна браузера с указанном ранее размером 1000×800 пикселей.

Визуализация целиком

На рисунке 15.8 показана вся визуализация данных о лауреатах Нобелевской премии с пятью ключевыми элементами и размещенными на самом верху пользовательскими элементами управления. Все организовано так, чтобы поместилось в области 1000×800 пикселей. Поскольку мы решили, что временная шкала должна занять почетное место, а карте мира требуется отдать центр, остальные элементы мы упорядочили по их особенностям. Для столбчатой диаграммы нужна максимальная ширина, чтобы уместились маркированные столбики всех 58 стран, тогда как список выбранных лауреатов и мини-биография отлично помещаются справа.

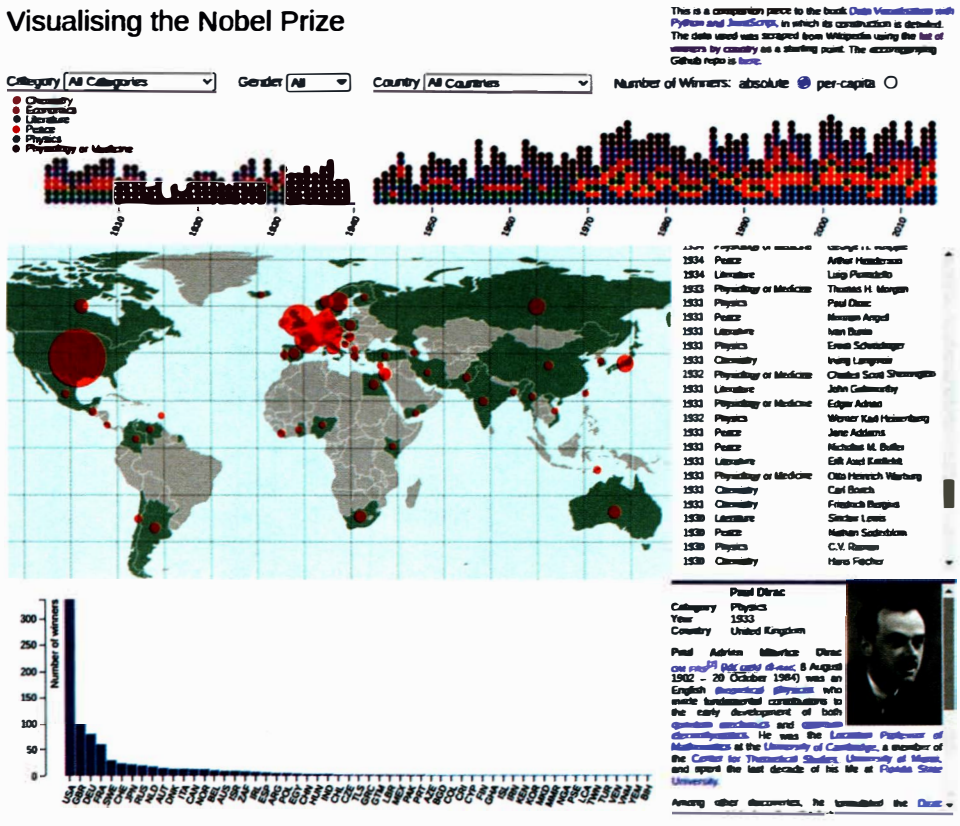


Рис. 15.8. *Вся визуализация данных о лауреатах Нобелевской премии*

Подведем итоги планирования визуализации, прежде чем перейти к ее реализации в следующей главе.

SKLADCHIK.ORG

Резюме

В этой главе мы представили себе нашу визуализацию, определили минимальный набор визуальных элементов, необходимых для рассказа о ключевых находках, сделанных в предыдущей главе. Все это сложилось в общую картину, показанную на рисунке 15.8. В следующих главах я покажу, как создавать отдельные элементы и как объединять их, чтобы сформировать современную интерактивную веб-визуализацию. По ходу простого рассказа о столбчатой диаграмме начнем понемногу знакомиться с библиотекой D3.

Создание визуализации

В главе 15 мы использовали результаты исследования набора данных о лауреатах Нобелевской премии (см. главу 11), чтобы составить план визуализации. На рисунке 16.1 показана задуманная нами визуализация, а в этой главе мы рассмотрим, как приступить к ее реализации, опираясь на язык JavaScript и библиотеку D3.

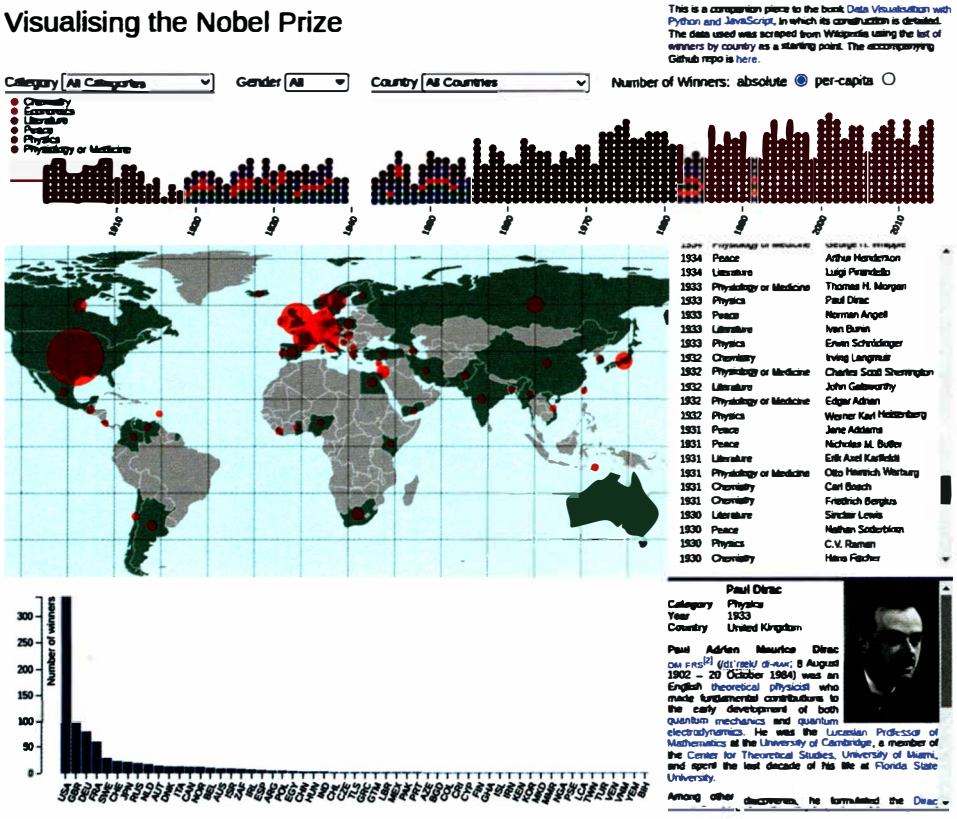


Рис. 16.1. *Наша цель — визуализация данных о лауреатах Нобелевской премии*

Я покажу, как объединить задуманные нами визуальные элементы для преобразования очищенного и обработанного набора данных в интерактивную

веб-визуализацию, которую можно одним движением отобразить на миллиардах устройств. Но прежде чем углубляться в детали, бегло ознакомимся с основными компонентами современной веб-визуализации.

Предварительные сведения

Перед тем, как приступить к созданию визуализации, рассмотрим основные компоненты, которые мы будем использовать, и то, как нам организовать файлы.

Основные компоненты

Как было показано в «Базовой странице с плейсхолдерами» на стр. 142, для современной веб-визуализации нужны четыре ключевых компонента:

HTML-каркас, на который будут опираться созданные нами JS-скрипты;

— один или несколько файлов CSS для управления внешним видом визуализации;

— JS-файлы, включая любые сторонние библиотеки, которые могут понадобиться (D3 — самая большая зависимость);

— и наконец, данные для преобразований, в идеале в формате CSV (для полностью статических данных) или JSON.

Прежде всего создадим файловую структуру для нашего проекта визуализации (Nobel-viz) и определим, как передавать данные в визуализацию.

Организация файлов

В примере 16-1 показана структура каталога нашего проекта. Согласно общепринятой практике, в корневом каталоге располагаются файл *index.html* и каталог *static*, содержащий все используемые для визуализации библиотеки и ресурсы (изображения, данные).

***Пример 16-1.** Файловая структура проекта Nobel-viz*

```
nobel_viz
├── index.html
└── static
    ├── css
    │   └── style.css
    ├── data                    ❶
    │   ├── nobel_winners_biopic.json
```

```
│   ├── winning_country_data.json
│   ├── world-110m.json
│   └── world-country-names-nobel.csv
├── images
│   └── winners ❷
│   └── full
│   ├── 002b4f05aa3758e2d6acadde4ed80aa991ed6357.jpg
│   ├── 00d7ed381db8b5d18edc84694b7f9ce14ee57c5b.jpg
│   ├── ...
└── js              ❸
├── nbviz_bar.mjs
├── nbviz_core.mjs
├── nbviz_details.mjs
├── nbviz_main.mjs
├── nbviz_map.mjs
├── nbviz_menu.mjs
└── nbviz_time.mjs
└── libs
├── crossfilter.min.js
├── d3.min.js
└── topojson.min.js
```

❶ В файлах со статическими данными хранятся, в частности, карта мира в формате TopoJSON (см. главу 19) и данные стран, полученные из интернета (см. «Получение данных о странах для визуализации нобелевских лауреатов» на стр. 181).

❷ Фотографии лауреатов, собранные с помощью Scrapy.

❸ В подкаталоге *js* содержатся файлы JavaScript-модулей (*.mjs*) для основных элементов проекта Nobel-viz. Название каждого модуля начинается с *nbviz_*.

Передача данных

Объем полного набора данных о лауреатах Нобелевской премии, который мы скрейпили в главе 6, составляет около 3 МБ, а при сжатии для веб-передачи значительно меньше. По современным меркам, для веб-страницы это не слишком большой объем данных. Средний размер веб-страницы составляет около 2–3 МБ[1]. Тем не менее, мы близки к моменту, когда следует рассмотреть возможность разбить данные на более мелкие фрагменты для загрузки по требованию. Также можно передавать данные динамически веб-сервером (см. главу 13) из базы данных, например SQLite. В таком случае небольшая задержка в начале компенсируется высокой скоростью работы в дальнейшем, так как браузер кеширует все данные. Это также значительно упрощает задачу, поскольку требуется только одна начальная загрузка данных с сервера.

[1] См. аналитику о среднем размере веб-страниц в материалах SpeedCurve и Web Almanac.

Для нашей визуализации мы будем передавать все данные из каталога data (см. пример 16-1), загружая их при инициализации приложения.

HTML-каркас

Хотя в нашей визуализации будет использоваться ряд динамических компонентов, ее HTML-каркас на удивление простой. Это в очередной раз подтверждает основной постулат книги: нужно совсем немного знаний в области традиционной веб-разработки, чтобы приступить к программированию визуализации данных.

Файл *index.html*, который при загрузке создает визуализацию, показан в примере 16-2. Он содержит три компонента:

1. Импортированную CSS-таблицу *style.css* для стилей шрифтов, позиционирования блоков контента и др.
2. HTML-плейсхолдеры для визуальных элементов с ID вида `nobel-[foo]`.
3. JavaScript — сначала сторонние библиотеки, затем наши собственные скрипты.

В следующих главах мы детально рассмотрим отдельные фрагменты HTML, но я хотел бы, чтобы вы обратили внимание, — эта часть визуализации данных о лауреатах Нобелевской премии в основном не требует программирования. Этот каркас позволяет сосредоточиться на креативном программировании — сильной стороне D3. По мере того, как вы привыкаете определять блоки контента в HTML и задавать размеры и позиционирование с помощью CSS, вы замечаете, что все больше времени остается на самое интересное: манипулирование данными с помощью программного кода.



Я считаю, что полезно рассматривать плейсхолдеры с идентификаторами как «рамки-обертки» соответствующих элементов. Например, плейсхолдер `<div id="nobel-map"></div>` *обрамляет* карту. В основном файле CSS или JS[1] мы задаем размер и относительное позиционирование этих «рамок», а элементы, такие как динамическая карта, адаптируются к размеру своей «рамки». Это позволяет не умеющему

[1] Я бы порекомендовал в основном использовать стили CSS, а JavaScript-стили — только в особых случаях.

программировать дизайнеру изменять внешний вид визуализации с помощью стилей CSS.

Пример 16-2. *Файл index.html — входная точка нашей одностраничной визуализации*

```
<! DOCTYPE html>
<meta charset="utf-8">
<title>Visualizing the Nobel Prize</title>
<!— 1. ИМПОРТ файла CSS для визуализации —>
<link rel="stylesheet" href="static/css/style.css" media="screen" />
<body>
   <div id='chart'>
      <!— 2. БЛОК С ЗАГОЛОВКОМ И БЛОК INFO —>
         <div id='title'>Visualizing the Nobel Prize</div>
         <div id="info">
            This is a companion piece to the book <a href='http://'>
            Data visualization with Python and JavaScript</a>, in which
            its construction is detailed. The data used was scraped
            Wikipedia using the <a href='https://en.wikipedia.org/wiki
            /List_of_Nobel_laureates_by_country'> list of winners by
            country</a> as a starting point. The accompanying GitHub
            repo is <a href= 'http://github.com/Kyrand/
            dataviz-with-python-and-js-ed-2'> here</a>.
         </div>
         <!— 3. ПРЕЙСХОЛДЕРЫ ДЛЯ ВИЗУАЛЬНЫХ КОМПОНЕНТОВ —>
         <div id="nbviz">
            <!— НАЧАЛО СТРОКИ МЕНЮ —>
            <div id="nobel-menu">
            <div id="cat-select">
               Category
            <select></select>
            </div>
            <div id="gender-select">
               Gender
               <select>
                  <option value="All">All</option>
                  <option value="female">Female</option>
                  <option value="male">Male</option>
```

```
        </select>
      </div>
      <div id="country-select">
        Country
        <select></select>
      </div>
      <div id='metric-radio'>
        Number of Winners: 
        <form>
          <label>absolute
            <input type="radio" name="mode" value="0" checked>
          </label>
          <label>per-capita
            <input type="radio" name="mode" value="1">
          </label>
        </form>
      </div>
    </div>
    <!— КОНЕЦ СТРОКИ МЕНЮ —>
    <!— НАЧАЛО КОМПОНЕНТОВ NOBEL-VIZ —>
    <div id='chart-holder' class='_dev'>
      <!— ВРЕМЕННАЯ ШКАЛА ПРЕМИЙ —>
      <div id="nobel-time"></div>
      <!— КАРТА И ВСПЛЫВАЮЩАЯ ПОДСКАЗКА —>
      <div id="nobel-map">
        <div id="map-tooltip">
          <h2></h2>
          <p></p>
        </div>
      </div>
      <!— СПИСОК ЛАУРЕАТОВ —>
      <div id="nobel-list">
        <h2>Selected winners</h2>
        <table>
          <thead>
            <tr>
              <th id='year'>Year</th>
              <th id='category'>Category</th>
              <th id='name'>Name</th>
```

```
                </tr>
            </thead>
            <tbody>
            </tbody>
        </table>
    </div>
    <!-- БИОГРАФИЧЕСКИЙ БЛОК -->
    <div id="nobel-winner">
        <div id="picbox"></div>
        <div id='winner-title'></div>
        <div id='infobox'>
            <div class='property'>
                <div class='label'>Category</div>
                <span name='category'></span>
            </div>
            <div class='property'>
                <div class='label'>Year</div>
                <span name='year'></span>
            </div>
            <div class='property'>
                <div class='label'>Country</div>
                <span name='country'></span>
            </div>
        </div>
        <div id='biobox'></div>
        <div id='readmore'>
            <a href='#'>Read more at Wikipedia</a>
        </div>
    </div>
    <!-- СТОЛБЧАТАЯ ДИАГРАММА -->
    <div id="nobel-bar"></div>
  </div>
  <!-- КОНЕЦ КОМПОНЕНТОВ NOBEL-VIZ -->
 </div>
</div>
<!-- 4. ФАЙЛЫ JAVASCRIPT -->
<!-- СТОРОННИЕ JS-БИБЛИОТЕКИ, В ОСНОВНОМ D3 -->
<script src="libs/d3.min.js"></script>
```

```
    <!- ... ->
    <!- ГЛАВНЫЙ JAVASCRIPT-МОДУЛЬ ДЛЯ ЭЛЕМЕНТОВ ВИЗУАЛИЗАЦИИ ->
    <script src="static/js/nbviz_main.mjs" ></script>
</body>
```

HTML-каркас (пример 16-2) определяет иерархическую структуру компонентов Nobel-viz, а визуальное оформление, в том числе размеры и позиционирование блоков задаются в файле *style.css*. В следующем подразделе мы рассмотрим, как это реализовано, и изучим общий стиль визуализации.

Стили CSS

Мы уже рассматривали стили отдельных компонентов диаграмм в соответствующих главах (см. диаграмму на рисунке 16.1). Сейчас мы рассмотрим неспецифичные стили CSS, самое главное — размеры и позиционирование блоков контента (*панелей*).

Над выбором размеров визуализации порой приходится поломать голову. Сегодня существует множество устройств: смартфоны, планшеты и другие мобильные гаджеты с разными разрешениями, такими как Retina[1] и Full HD (1920×1080). Размеры пикселей стали разнообразнее из-за появления экранов с повышенной плотностью, которая стала ключевым параметром. Большинство устройств масштабируют пиксели для компенсации этого явления, поэтому текст на смартфоне остается читабельным, даже если на нем столько же пикселей, сколько на большом настольном мониторе. Большинство мобильных устройств поддерживает жесты масштабирования (pinch-to-zoom) и перемещения по экрану (pan), что позволяет пользователю фокусироваться на отдельных областях большой визуализации. Мы выбираем компромиссный размер 1280×800 пикселей, при котором наша визуализация будет хорошо выглядеть на большинстве настольных мониторов и подойдет (включая верхние элементы управления высотой 50 пикселей) для горизонтальной ориентации экрана мобильных устройств.

Сначала зададим базовые стили документа с помощью селектора `body`: шрифт без засечек (sans-serif) и цвет фона (белый с желтоватым оттенком). Также укажем параметры ссылок. Еще мы зададим ширину визуализации в целом и ее наружные отступы (margin):

[1] В настоящее время около 2560×1600 пикселей.

```
body {
    font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;
    background: #fefefe; ❶
    width: 1000px;
    margin: 0 auto; /* сверху и снизу 0, слева и справа auto */
}

a: link {
    color: royalblue;
    text-decoration: none; ❷
}
a: hover {
    text-decoration: underline;
}
```

❶ Этот цвет чуть темнее чисто белого `#ffffff`, что делает страницу менее яркой и более комфортной для глаз.

❷ Стандартное подчеркивание гиперссылок, на мой взгляд, выглядит аляповато, так что мы уберем его.

В Noble-viz три основных блока контента , позиционируемых абсолютно внутри с селектором `#chart` (их относительного родителя). Это основной заголовок (`#title`), краткая информация о визуализации (`#info`) и основной контейнер (`#nbviz`). Основной контейнер, занимающий 100 % ширины, устанавливаем на 90 пикселей ниже верхнего края, чтобы оставить место для заглавия и информации (которые размещаем «на глазок»). Для вышеперечисленного используем такой CSS-код:

```
#nbviz {
    position: absolute;
    top: 90px;
    width: 100%;
}

#title {
    position: absolute;
    font-size: 30px;
    font-weight: 100;
    top: 20px;
}
```

SKLADCHIK.ORG

```
#info {
   position: absolute;
   font-size: 11px;
   top: 18px;
   width: 300px;
   right: 0px;
   line-height: 1.2;
}
```

Для `chart-holder` задаем высоту 750 пикселей и ширину 100 % от его родителя. Поскольку свойству `position` мы присваиваем значение `relative`, то абсолютное позиционирование дочерних панелей будет отсчитываться от его верхнего левого угла. Нижний отступ для диаграмм равен 20 пикселям:

```
#chart-holder {
   width: 100%;
   height: 750px;
   position: relative;
   padding: 0 0 20px 0; /* верх справа низ слева */
}

#chart-holder svg { ❶
   width: 100%;
   height: 100%;
}
```

❶ SVG-контекст компонентов будет расширяться в соответствии с контейнерами.

Учитывая ограничение высоты Nobel-viz в 750 пикселей, соотношение ширины и высоты равнопромежуточной проекции карты 2:14[1], а также необходимость разместить на временной диаграмме круглые индикаторы Нобелевских премий за более чем 100 лет, оптимальным компромиссом для размеров визуальных элементов стал вариант, показанный рисунке 16.2.

[1] Сравнение различных геометрических проекций см. в разделе «Проекции» на стр. 558. Для отображения всех стран, гражданами которых были нобелевские лауреаты, наиболее эффективной оказалась равнопромежуточная проекция.

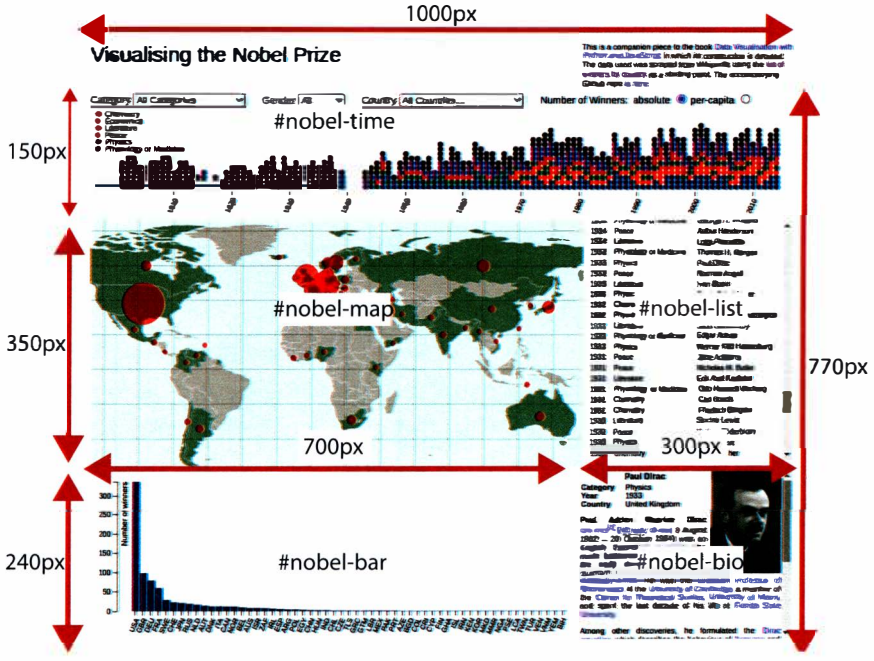


Рис. 16.2. *Размеры Nobel-viz*

Результат задания размеров и позиционирования компонентов с помощью CSS показан на рисунке 16.2:

```
#nobel-map, #nobel-winner, #nobel-bar, #nobel-time, #nobel-list{
   position: absolute; ❶
}

#nobel-time {
   top: 0;
   height: 150px;
   width: 100%; ❷
}

#nobel-map {
   background: azure;
   top: 160px;
   width: 700px;
   height: 350px;
}
```

```
#nobel-winner {
   top: 510px;
   left: 700px;
   height: 240px;
   width: 300px;
}

#nobel-bar {
   top: 510px;
   height: 240px;
   width: 700px;
}

#nobel-list {
   top: 160px;
   height: 340px;
   width: 290px;
   left: 700px;
   padding-left: 10px; ❸
}
```

❶ Задаем абсолютное позиционирование относительно родительского контейнера `chart-holder`.
❷ Временная шкала занимает всю ширину визуализации.
❸ Внутренние отступы позволяют компонентам «дышать».

Остальные стили CSS специфичны для отдельных компонентов, мы рассмотрим их в соответствующих главах. HTML и CSS каркас нашей визуализации теперь можно наполнить содержимым с помощью JavaScript.

Движок JavaScript

Внедрять модульность на раннем этапе полезно для визуализации любого размера. Многие примеры в интернете, созданные с помощью D3[1], — это решения, где на одной веб-странице объединяются HTML, CSS, JS и даже данные. Хотя это удобно для обучающих примеров, с ростом кодовой базы поддержка становится

[1] См. коллекцию на D3 GitHub.

сложнее: изменения требуют больше усилий, возрастает вероятность конфликтов пространства имен и подобных проблем.

Импорт скриптов

Как показано в примере 16.2, мы импортируем содержимое внешних JavaScript-файлов, необходимых для нашей визуализации, с помощью тегов `<script>`, размещенных в самом низу тега `<body>` файла *index.html*:

```
<! DOCTYPE html>
<meta charset="utf-8">
...
<body>
...
   <!— СТОРОННИЕ JS-БИБЛИОТЕКИ, В ОСНОВНОМ НА БАЗЕ D3 —> ❶
   <script src="static/libs/d3.min.js"></script>
   <script src="static/libs/topojson.min.js"></script>
   <script src="static/libs/crossfilter.min.js"></script>
   <!— JAVASCRIPT ДЛЯ ЭЛЕМЕНТОВ ВИЗУАЛИЗАЦИИ —>
   <script src="static/js/nbviz_main.mjs"></script> ❷
</body>
```

❶ Используем локальные копии сторонних библиотек.

❷ Главная точка входа приложения, в которой оно запрашивает первые наборы данных и запускает процесс отображения. Этот модуль импортирует все остальные модули, используемые в визуализации.

Сети CDN

В тегах script на страницах *index.html* (как правило) можно часто увидеть использование сетей доставки контента (CDN) для получения кешируемых библиотек, что позволяет эффективно использовать центральное хранилище. Для импорта D3:app-name вы можете воспользоваться CDN Cloudflare:

```
<script src=  "https://cdnjs.cloudflare.com/ajax/libs/d3/7.3.1/
   d3.js"></script>
```

Если вы используете много относительно ресурсоемких библиотек, то эффект от применения CDN получите очень быстро. Однако в процессе разработки полезно использовать локальную копию библиотеки, это мы и делаем: три библиотеки, используемые в визуализации, извлекаются из каталога *libs*, находящегося в каталоге *static*.

Импорт модулей JS

В первом издании я использовал распространенный, но костыльный шаблон для создания пространства имен `nbviz`, включающий функции, переменные и константы компонентов визуализации данных о нобелевских лауреатах. Приведу пример, поскольку вы вполне можете столкнуться с похожими шаблонами в чужом коде:

```
/* js/nbviz_core.js
/* global $, _, crossfilter, d3  */ ❶
(function(nbviz) {
   //... ПРИВАТНЫЕ ПЕРЕМЕННЫЕ МОДУЛЕЙ И Т.Д.
   nbviz.foo = function(){//... ❷
};
}(window.nbviz = window.nbviz || {})); ❸
```

❶ Определение глобальных переменных для предотвращения ошибок JSLint.

❷ Предоставляет эту функцию другим скриптам как часть общего пространства имен `nbviz`.

❸ Использует объект `nbviz`, если он доступен, в противном случае создает его.

Каждый JS-скрипт был инкапсулирован с помощью данного шаблона, а все необходимые скрипты были включены в тег `<script>` в основной точке входа — *index.html*. С появлением кросс-браузерной поддержки модулей JS мы получили современный способ импорта, знакомый питонистам (см. «Модули JavaScript» на стр. 53).

Итак, нам осталось включить наш основной модуль JS в *index.html*, а он импортирует все остальные необходимые модули:

```
// static/js/nbviz_main.mjs
import nbviz from './nbviz_core.mjs'
import {initMenu} from './nbviz_menu.mjs'
```

```
import {initMap} from './nbviz_map.mjs'
import './nbviz_bar.mjs' ❶
import './nbviz_details.mjs'
import './nbviz_time.mjs'
```

❶ Эти модули импортируются для инициализации их коллбэков обновления. Чуть дальше в этой главе мы рассмотрим их работу.

В следующих главах будет подробно описано применение JavaScript/D3 для создания элементов визуализации. Сначала мы рассмотрим поток данных через Nobel-viz: от сервера (данных) к клиентскому браузеру, а также внутри него при взаимодействии с пользователем.

Основной поток данных

Есть много способов работать с данными в проектах любой сложности. Для интерактивных приложений, особенно визуализаций, я считаю оптимальным использовать для кеширования текущих данных центральный объект данных. Помимо кешированных данных в основном объекте данных хранятся подмножества набора данных, или его активные отражения. В визуализации данных о нобелевских лауреатах пользователь может выбирать подмножества (например, только лауреатов в номинации«Физика»).

Когда пользователь меняет представление данных, например, вместо абсолютного количества премий выбирает метрику «премии на душу населения», то флаг[1] (в данном случае `valuePerCapita` устанавливается в 0 или 1). Затем все визуальные компоненты обновляются, в том числе и те, что зависят от `valuePerCapita`. Изменяется размер маркеров на карте, и перестраивается столбчатая диаграмма.

Ключевая идея — синхронизировать визуальные элементы с изменениями данных, которые инициирует пользователь. Для этого надежнее всего иметь единый метод обновления (здесь — `onDataChange`), который вызывается при любом пользовательском изменении данных. Этот метод оповещает все активные визуальные элементы, и они реагируют на изменение соответствующим образом.

Теперь рассмотрим, как организован код приложения. Начнем с общих базовых утилит.

[1] Наше приложение реализовано максимально просто: при расширении набора UI-опций рекомендуется хранить флаги, диапазоны и т. д. в отдельном объекте.

Основной код

Первым из JavaScript-файлов загружается *nbviz_core.js*. Этот скрипт содержит код, который будет использоваться другими скриптами. Например, функция `categoryFill` возвращает конкретный цвет для каждой номинации. Возвращенный результат используется как для компонентов временной диаграммы, так и для верхней границы биографического блока. Основной код включает функции, которые изолированы для удобства тестирования или упрощения других модулей.



Строковые константы часто используются в программировании в качестве ключей словаря, в операциях сравнения и в генерируемых метках. Легко поддаться дурной привычке использовать строки по мере необходимости, но гораздо лучше объявить константу. Например, вместо того, чтобы писать `'if option === "All Categories"'` лучше использовать `'if option === nbviz.ALL_CATS'`. Если в строке `'All Categories'` будет опечатка, то это не вызовет ошибки, но будет получен неверный результат. Использование const позволяет изменить все вхождения строки однократным редактированием. В JavaScript в свое время появилось ключевое слово const, которое немного упрощает обеспечение константности, хотя всего лишь предотвращает переназначение переменных. Подробности и примеры const см. в документации Mozilla.

В примере 16-3 показан код, общий для остальных модулей. Все, что предназначено для использования остальными модулями, присоединяется к общему пространству имен `nbviz`.

Пример 16-3. *Общая кодовая база в файле nbviz_core.js*

```
let nbviz = {}
nbviz.ALL_CATS = 'All Categories'
nbviz.TRANS_DURATION = 2000 // время визуальных переходов в мс
nbviz.MAX_CENTROID_RADIUS = 30
nbviz.MIN_CENTROID_RADIUS = 2
nbviz.COLORS = {palegold: '#E6BE8A'} // любые именованные цвета
```

```
nbviz.data = {} // our main data store
nbviz.valuePerCapita = 0 // metric flag
nbviz.activeCountry = null
nbviz.activeCategory = nbviz.ALL_CATS

nbviz.CATEGORIES = [
   "Chemistry", "Economics", "Literature", "Peace", "Physics",
   "Physiology or Medicine"
];
// принимает номинацию, например, Physics и возвращает цвет для нее
nbviz.categoryFill = function(category){
   var i = nbviz.CATEGORIES.indexOf(category);
   return d3.schemeCategory10[i]; ❶
};

let nestDataByYear = function(entries) { ❷
//...
};

nbviz.makeFilterAndDimensions = function(winnersData){
//...
};

nbviz.filterByCountries = function(countryNames) {
//...
};

nbviz.filterByCategory = function(cat) {
//...
};

nbviz.getCountryData = function() {
// ...
};

nbviz.callbacks = [] nbviz.onDataChange = function () { ❸
   nbviz.callbacks.forEach((cb) => cb())
}

export default nbviz ❹
```

❶ Используем для получения цветов номинаций премии одну из встроенных цветовых схем D3. `schemeCategory10` — это массив из 10 hex-кодов цветов (`['#1f77b4', '#ff7f0e',...]`), к которым мы получаем доступ с помощью индексов номинаций.

❷ Подробное объяснение этой и следующих пустых функций будет дано в контексте их использования в следующих главах.

❸ Эта функция вызывается при изменении набора данных (после инициализации приложения, по инициативе пользователя) для обновления элементов Nobel-viz. Коллбэки обновлений, установленные модулями компонентов и сохраненные в массиве `callbacks`, вызываются в свою очередь, запуская необходимые визуальные изменения.

❹ Объект `nbviz` с утилитами, константами и переменными для этого модуля экспортируется по умолчанию и импортируется другими модулями через `import nbviz from ./nbviz_core`.

Итак, у нас есть основной код, теперь рассмотрим, как инициализируется наше приложение при извлечении статических ресурсов с помощью служебных методов D3.

Инициализация визуализации Нобелевской премии

Для запуска приложения нам потребуются данные. Используем вспомогательные методы `d3.json` и `d3.csv`, чтобы загрузить данные и преобразовать их в объекты и массивы JavaScript. Метод `Promise.all`[1] одновременно запускает промисы для выборок данных, ожидает завершения всех четырех, а затем передает данные указанной функции-обработчику, в данном случае — `ready`:

```
// static/js/nbviz_main.mjs
//...
   Promise.all([ ❶
      d3.json('static/data/world-110m.json'),
      d3.csv('static/data/world-country-names-nobel.csv'),
      d3.json('static/data/winning_country_data.json'),
         d3.json('static/data/nobel_winners_biopic.json'),
   ]).then(ready)
```

[1] Подробнее о Promise.all см. в документации Mozilla.

```
function ready([worldMap, countryNames, countryData,
                winnersData]) { ❷
  // ХРАНИТ НАБОР ДАННЫХ COUNTRY-DATA
  nbviz.data.countryData = countryData
  nbviz.data.winnersData = winnersData
  //...
}
```

❶ Одновременный вызов запросов для четырех файлов с данными. Статические файлы содержат карту мира (110m) и данные о странах, которые мы будем использовать в визуализации.

❷ К массиву, возвращенному `ready`, применяем деструктуризацию JavaScript, чтобы «распаковать» массивы в соответствующие переменные.

Если запросы были успешными, функция `ready` получает запрошенные данные, и мы готовы начать передачу данных визуальным элементам.

Готовность к работе

После того, как отложенные запросы данных, созданные методом `Promise.all`, будут выполнены, он вызовет функцию `ready`, передавая ей наборы данных в качестве аргументов в том порядке, в котором они были добавлены.

Функция `ready` показана в примере 16-4. Если загрузка прошла без ошибок, используем данные о лауреатах для создания активного фильтра (предоставленного библиотекой Crossfilter), позволяющего выбирать подмножества по номинации, полу и стране. Затем мы вызываем методы инициализации и, наконец, используем `onDataChange` для запуска отрисовки визуальных элементов визуализации, обновления столбчатой диаграммы, карты, временной диаграммы и др. На рисунке 16.3 схематично показано, как распространяются изменения данных.

***Пример 16-4.** Функция ready вызывается после успешного выполнения первоначальных запросов данных*

```
//...
function ready([worldMap, countryNames, countryData,
                winnersData]) {
  // ХРАНИТ НАБОР ДАННЫХ COUNTRY-DATA
  nbviz.data.countryData = countryData
  nbviz.data.winnersData = winnersData
```

SKLADCHIK.ORG

```
    // СОЗДАНИЕ ФИЛЬТРА И ЕГО ИЗМЕРЕНИЙ
    nbviz.makeFilterAndDimensions(winnersData) ❶
    // ИНИЦИАЛИЗАЦИЯ МЕНЮ И КАРТЫ
    initMenu()
    initMap(worldMap, countryNames)
    // ЗАПУСК ОБНОВЛЕНИЯ С ПОЛНЫМ НАБОРОМ ДАННЫХ О ЛАУРЕАТАХ
    nbviz.onDataChange()
}
```

❶ Этот метод использует только что загруженный набор данных о нобелевских лауреатах для создания фильтра, позволяющего пользователю выбирать подмножества данных для визуализации.

Когда мы будем рассматривать библиотеку Crossfilter в подразделе «Фильтрация данных с помощью Crossfilter» на странице 400, то увидим, как работает метод `makeFilterAndDimensions` (пример 16-4 ❶). Пока предположим, что у нас есть способ получать данные, выбранные пользователем через элементы меню (например, всех женщин-лауреатов).

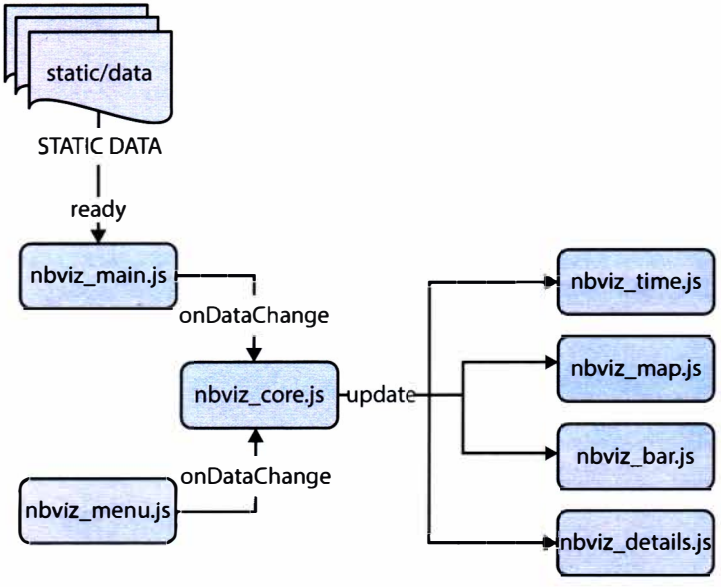


Рис. 16.3. *Основной поток данных приложения*

Обновление на основе данных

После инициализации меню и карты в функции `ready` (мы увидим, как это работает в главе 19 для карты и главе 21 для меню), мы запускаем обновление визуальных элементов с помощью метода `onDataChange` из файла *nbviz_core.js*. Метод `onDataChange` (см. пример 16-5) — это общая функция, которая вызывается при каждом изменении отображаемых данных или смене метрики (например, подсчета на душу населения вместо абсолютных значений).

***Пример 16-5.** Функция, которая вызывается для обновления визуальных элементов в ответ на изменение выбранных данных*

```
// nbviz_core.js
nbviz.callbacks = [] ❶

nbviz.onDataChange = function () {
   nbviz.callbacks.forEach((cb) => cb()) ❷
}
```

❶ Каждый модуль компонента, требующий обновления, добавляет свой коллбэк к этому массиву.

❷ При изменении данных поочередно вызываются коллбэки компонентов, запуская необходимые изменения визуализации.

При первом импорте модулей они добавляют свои коллбэки в массив `callbacks` в основном модуле. Пример для столбчатой диаграммы:

```
// nbviz_bar.mjs
import nbviz from './nbviz_core.mjs'
// ...
nbviz.callbacks.push(() => {
   let data = nbviz.getCountryData()
   updateBarChart(data) ❶
})
```

❶ Когда основная функция обновления вызывает эту коллбэк-функцию, локальная функция обновления использует данные по странам для изменения столбчатой диаграммы.

Основной набор данных (для временной шкалы, карты и столбчатой диаграммы) создается методом `getCountryData`. Он группирует лауреатов по странам и добавляет информацию о стране — численность населения и международный буквенный код. В примере 16-6 рассматривается этот метод.

***Пример 16-6.** Создание основного набора данных по странам*

```
nbviz.getCountryData = function() {
   var countryGroups = nbviz.countryDim.group().all(); ❶

   // создать основной набор данных
   var data = countryGroups.map(function(c) { ❷
      var cData = nbviz.data.countryData[c.key]; ❸
      var value = c.value;
      // если значение на душу населения, то разделить
      // на численность населения
      if(nbviz.valuePerCapita){
         value = value / cData.population; ❹
      }
      return {
         key: c.key, // например, Japan
         value: value, // например, 19 (премий)
         code: cData.alpha3Code, // например, JPN
      };
   })
      .sort(function(a, b) { ❺
         return b.value - a.value; // по убыванию
      });

   return data;
};
```

❶ Одно из измерений Crossfilter — `countryDim`, предоставляющий здесь ключ группы и количество премий (например, `{key: Argentina, value:5}`).

❷ Используем метод `map` для создания нового массива с компонентами, добавленными из набора данных по странам.

❸ Извлекает данные о стране, используя ключ группы (например, Australia).

❹ Если радиокнопка `valuePerCapita` включена, то делим число премий на численность населения страны, что дает более *справедливое* представление о распределении премий.

❺ Используем метод массивов `sort`, чтобы отсортировать элементы массива по убыванию значений.

Все методы для обновления Nobel-viz используют данные, отфильтрованные библиотекой Crossfilter. Давайте посмотрим, как это делается

Фильтрация данных с помощью Crossfilter

Crossfilter[1] разработали создатели D3 Майк Бостоk и Джейсон Дэвис (Jason Davies). Это высокооптимизированная JavaScript-библиотека для анализа больших многомерных наборов данных. Она очень быстро работает и легко справляется с наборами данных, объемы которых намного превосходят объем нашего набора. Мы будем использовать эту библиотеку для фильтрации набора данных о лауреатах по номинациям, полу и странам.

Возможно, для наших задач подошло бы что-то поскромнее библиотеки Crossfilter, но я хотел показать ее в действии, поскольку считаю, что она очень полезна. Еще одно свидетельство эффективности — Crossfilter лежит в основе *dc.js*, популярной библиотеки для построения D3-диаграмм. Хотя Crossfilter порой сложна для понимания, особенно когда дело касается фильтров по пересекающимся измерениям, большинство сценариев использования следуют базовому шаблону, который быстро усваивается. Когда приходится делать срезы больших наборов данных, оптимизация Crossfilter — настоящее спасение.

Создание фильтра

При инициализации Nobel-viz функция `ready`, определенная в файле *nbviz_main.js*, вызывает метод `makeFilterAndDimensions` из файла *nbviz_core.js*. Он создает Crossfilter-фильтр и измерения (например, номинации) на основе загруженных данных о нобелевских лауреатах.

Создаем для начала фильтр, используя полученный при инициализации набор данных о лауреатах Нобелевской премии. Давайте вспомним, как выглядит набор:

[1] Пример многомерной фильтрации см. на странице Square https://square.github.io/crossfilter/.

```
[{
   name:"C\u00e9sar Milstein",
   category:"Physiology or Medicine",
   gender:"male",
   country:"Argentina",
   year: 1984
 },
 {
   name:"Auguste Beernaert",
   category:"Peace",
   gender:"male",
   country:"Belgium",
   year: 1909
 },
 ...
}];
```

Чтобы создать фильтр, вызовем функцию `crossfilter` с массивом объектов winnersData:

```
nbviz.makeFilterAndDimensions = function(winnersData){
   // ДОБАВИМ ФИЛЬТР И СОЗДАДИМ ИЗМЕРЕНИЯ
   nbviz.filter = crossfilter(winnersData);
   //...
};
```

Crossfilter позволяет вам создавать многомерные фильтры для данных, применяя функцию к объектам. В простейшем случае это создает измерение на основе одной категории — например, по полу. Создаем гендерное измерение, по которому будем фильтровать данные о лауреатах Нобелевской премии:

```
nbviz.makeFilterAndDimensions = function(winnersData){
//...
   nbviz.genderDim = nbviz.filter.dimension(function(o) {
      return o.gender;
   });
//...
}
```

Это измерение теперь будет упорядочивать наш набор данных по полю gender. Получим с его помощью все объекты со значением «пол женский»:

```
nbviz.genderDim.filter('female'); ❶
var femaleWinners = nbviz.genderDim.top(Infinity); ❷
femaleWinners.length // 47
```

❶ `filter` принимает одно значение или, где это нужно, диапазон значений (например [5, 21] — все значения от 5 до 21). Он также может принимать булеву функцию значений.

❷ После применения фильтра `top` возвращает указанное количество упорядоченных объектов. Если указано `Infinity`[1], то возвращаются все отфильтрованные объекты данных.

Возможности Crossfilter раскрываются в полной мере при работе с несколькими различными фильтрами, позволяющими разбивать данные на любые подмножества с впечатляющей скоростью[2].

Удалим измерение по полу и добавим новое, отфильтровав данные по номинации премии. Чтобы сбросить измерение[3], нужно применить метод `filter` без аргументов:

```
nbviz.genderDim.filter();
nbviz.genderDim.top(Infinity) //   весь массив [858] объектов
```

Теперь создадим новое измерение по номинации премии:

```
nbviz.categoryDim = nbviz.filter.dimension(function(o) {
   return o.category;
});
```

Теперь мы последовательно можем фильтровать по измерениям пола и номинации, чтобы, например найти всех женщин-лауреатов по физике:

[1] Значение `Infinity` в JavaScript — это значение типа number, обозначающее бесконечность.

[2] Библиотека Crossfilter предназначена для обновления миллионов записей в режиме реального времени в ответ на команды пользователя.

[3] Таким образом очищаются все фильтры данного измерения.

```
nbviz.genderDim.filter('female'); nbviz.categoryDim.
filter('Physics'); nbviz.genderDim.top(Infinity);
// Вывод:
// [
//   {name:"Marie Sklodowska-Curie", category:"Physics",…
//  {name:"Maria Goeppert-Mayer", category:"Physics",…
//]
```

Обратите внимание, что можно выборочно включать и выключать фильтры. Например, удалим фильтр по номинации Physics, тогда сработает только гендерный фильтр, и мы получим всех женщин-лауреатов:

```
nbviz.categoryDim.filter();
nbviz.genderDim.top(Infinity); // Массив из [47] объектов
```

В нашем проекте Nobel-viz пользователь, делающий выбор в верхней строке меню, осуществляет операции фильтрации данных.

Crossfilter может не только возвращать отфильтрованные подмножества, но и группировать данные. Воспользуемся этой возможностью, чтобы получить национальные сводные показатели по премии для столбчатой диаграммы и карты:

```
nbviz.genderDim.filter(); // reset gender dimension
var countryGroup = nbviz.countryDim.group(); ❶
countryGroup.all(); ❷

// Вывод:
// [
// {key:"Argentina", value:5}, ❸
//  {key:"Australia", value:9},
//  {key:"Austria",  value:14},
// ...]
```

❶ group может принимать необязательную функцию как аргумент, но обычно достаточно значения по умолчанию.

❷ Возвращает все группы по ключу и значению. Не изменяйте возвращаемый массив[1].

❸ Здесь `value` — это общее количество нобелевских лауреатов из Аргентины.

[1] См. страницу Crossfilter на GitHub.

Для создания фильтра и измерений Crossfilter мы используем метод `makeFilterAndDimensions`, определенный в файле *nbviz_core.js*. В примере 16-7 метод показан целиком. Примечание: порядок создания фильтров не имеет значения, он не влияет на их пересечение.

***Пример 16-7.** Создание фильтров и измерений Crossfilter*

```
nbviz.makeFilterAndDimensions = function(winnersData){
   // ДОБАВИМ ФИЛЬТР И СОЗДАДИМ ИЗМЕРЕНИЯ
   nbviz.filter = crossfilter(winnersData);
   nbviz.countryDim = nbviz.filter.dimension(function(o){ ❶
      return o.country;
   });

   nbviz.categoryDim = nbviz.filter.dimension(function(o) {
      return o.category;
   });

   nbviz.genderDim = nbviz.filter.dimension(function(o) {
      return o.gender;
   });
};
```

❶ Для наглядности мы используем полную форму функций JavaScript, но сейчас чаще применяют компактную форму записи — стрелочную функцию, например `o => o.country`.

Запуск приложения для визуализации данных о нобелевских лауреатах

Для запуска нашей визуализации требуется веб-сервер, имеющий доступ к файлу *index.html*. В процессе разработки для запуска сервера можно использовать встроенный в Python модуль `http`. Из корневого каталога, где хранится *index.html*, запускаем:

```
$ python -m http.server 8080
Serving HTTP on 0.0.0.0 port 8080 …
```

Откроем окно браузера и перейдем по адресу *http: localhost:8080*. Мы должны увидеть то, что показано на рисунке 16.4.

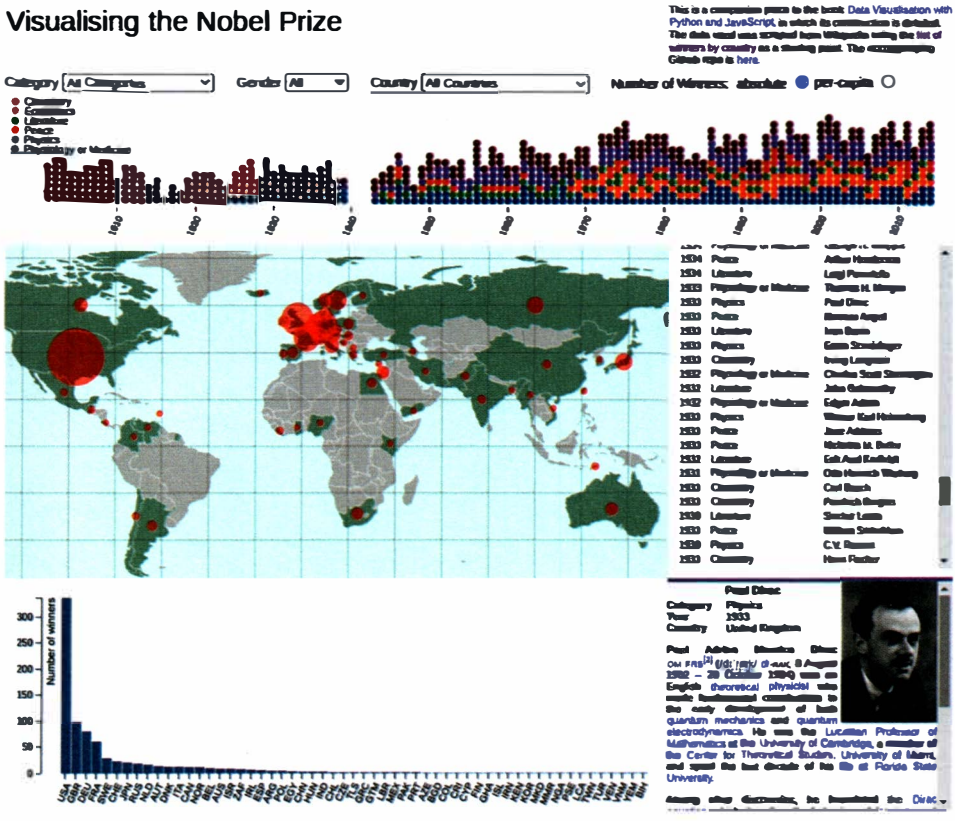


Рис. 16.4. *Приложение Nobel-viz готово*

Резюме

В этой главе кратко описано, как реализовать визуализацию, план создания которой мы составили в главе 15. Было показано, как построить основу из HTML, CSS и JavaScript, а также описана передача данных в приложение и их поток внутри него. В следующих главах мы увидим, как отдельные компоненты Nobel-viz создают интерактивную визуализацию на основе отправленных им данных. Начнем с большой главы, в которой ознакомимся с основами D3 и узнаем, как создать компонент столбчатой диаграммы нашего приложения. Это будет подготовкой к последующим главам, посвященным D3.

Введение в D3 на примере столбчатой диаграммы

В главе 16 мы представили себе визуализацию данных о лауреатах Нобелевской премии, разбив ее на составные элементы. В этой главе я познакомлю вас с D3 в процессе построения нужной нам столбчатой диаграммы (рисунок 17.1).

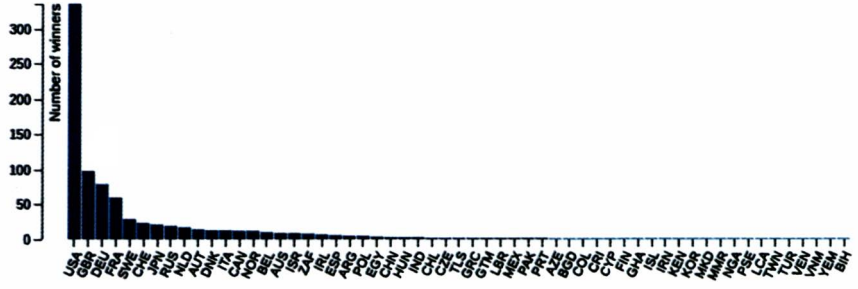


Рис. 17.1. *Цель этой главы: столбчатая диаграмма*

D3 — это нечто большее, чем просто библиотека для построения диаграмм. Помимо прочего, это библиотека для создания других библиотек диаграмм. Так почему же я знакомлю вас с D3 на примере самой обычной, ультраконсервативной визуализации — столбчатой диаграммы? Во-первых потому, что немного волнуешься, когда впервые создаешь диаграмму с нуля, полностью управляя ее внешним видом и функциональностью, безо всяких ограничений, присущих конкретным библиотекам построения диаграмм. Во-вторых, это удачный способ охватить фундаментальные элементы D3, в частности, привязку/объединение данных и паттерн обновления enter-exit-remove, инкапсулированные в методе `join`. Усвоив основы, вы существенно продвинетесь по пути к использованию всей мощи и выразительности D3 и созданию более оригинальных визуализаций, чем простая столбчатая диаграмма.

Мы будем использовать некоторые элементы веб-разработки, рассмотренные в главе 4, в частности графику SVG, на которой специализируется D3. Вы можете проверять фрагменты кода в онлайн-редакторе, например: CodePen или

SKLADCHIK.ORG

VizHub (где также имеется множество тщательно отобранных примеров визуализации данных).

Прежде чем построить столбчатую диаграмму, рассмотрим ее элементы.

Формулирование задачи

Ключевые компоненты столбчатой диаграммы — оси, легенды и метки, а также, естественно, столбики. Поскольку мы создаем современные интерактивный компоненты, нам нужно, чтобы оси и столбики трансформировались в ответ на действия пользователя, который отфильтровывает выборку из данных о лауреатах с помощью выпадающих списков строки меню (см. рисунок 15.1).

Мы будем строить диаграмму шаг за шагом, заканчивая переходами (transitions) D3, которые могут сделать ваши творения более интересными и визуально привлекательными. Но сначала мы рассмотрим основы D3:

— выбор DOM-элементов на веб-странице;
— получение и установка их атрибутов, свойств и стилей;
— вставка и добавление DOM-элементов.

Усвоив эти основы, перейдем к связыванию данных, где D3 начнет показывать свои силы.

Работа с выборкой

Выборки (Selections) — это становой хребет D3. Используя основанные на CSS jQuery-селекторы, D3 может выбирать отдельные и сгруппированные DOM-элементы и манипулировать ими. Все цепочки операции D3 начинаются с выбора элемента или набора элементов DOM. Для этого используются методы `select` и `selectAll`. Метод `select` возвращает первый отвечающий требованию элемент, а `selectAll` — набор элементов.

На рисунке 17.2 приведены примеры выборок D3, как методом `select`, так и `selectAll`. Эти выборки меняют атрибут высоты `height` одного или нескольких столбиков. Метод `select` возвращает первый элемент `rect` (ID `barL`) с классом `bar`, а `selectAll` может возвращать любую комбинацию элементов `rect` в зависимости от запроса.

Помимо настройки атрибутов (именованных строк в элементах DOM, например, `id` или `class`), D3 позволяет задавать стили элементов CSS, свойства (например, выбран ли чекбокс), возвращать текст и HTML.

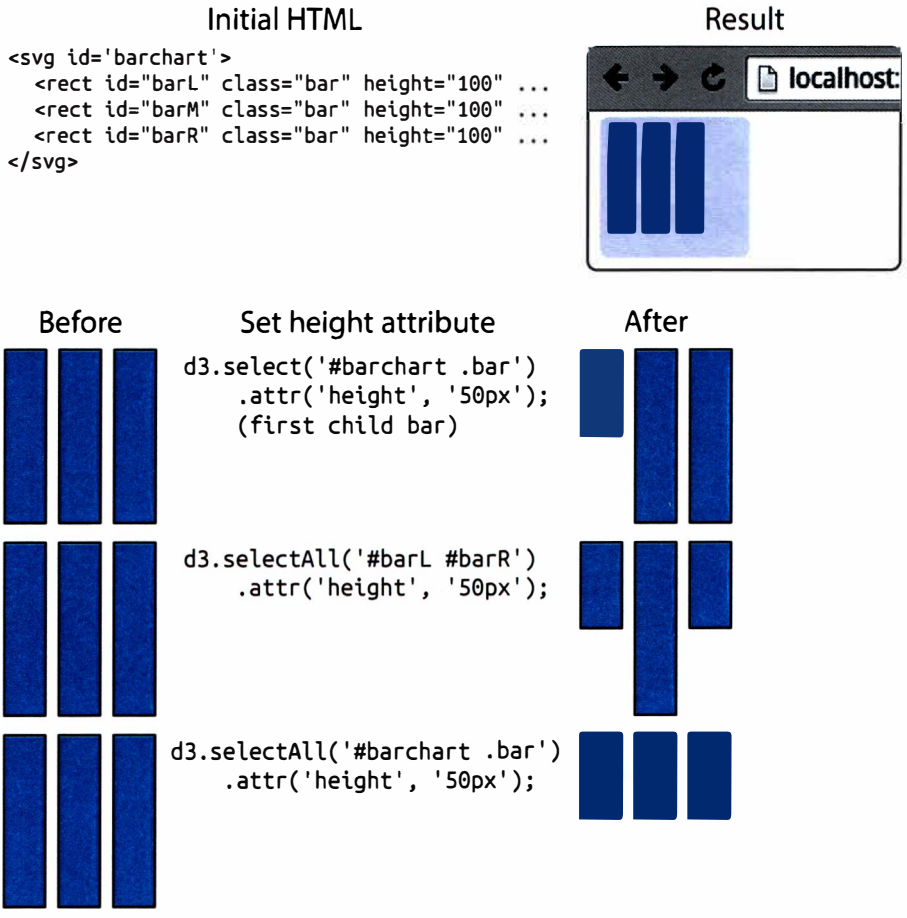


Рис. 17.2. *Выборки элементов и изменение их атрибутов: с помощью первоначального кода HTML созданы три прямоугольника. Из них делаем выборки, соответственно меняется атрибут высоты одного или нескольких столбиков.*

На рисунке 17.3 показаны все способы изменения элементов DOM с помощью D3. Используя эти методы, уже можно реализовать практически любой внешний вид и поведение элементов.

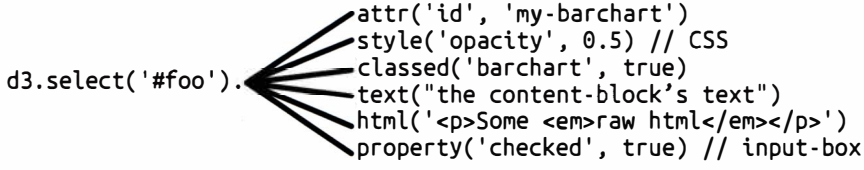


Рис. 17.3. *Изменение элемента DOM с помощью D3*

На рисунке 17.4 показано, как можно применить стиль CSS, добавив класс к элементу, или задав стиль напрямую. Сначала мы выбираем средний столбик по его идентификатору `barM`. Затем используем метод `classed` для применения желтой подсветки (см. CSS) и задаем для атрибута высоты `height` значение в 50 пикселей. Затем с помощью метода `style` красная заливка применяется непосредственно к столбику.

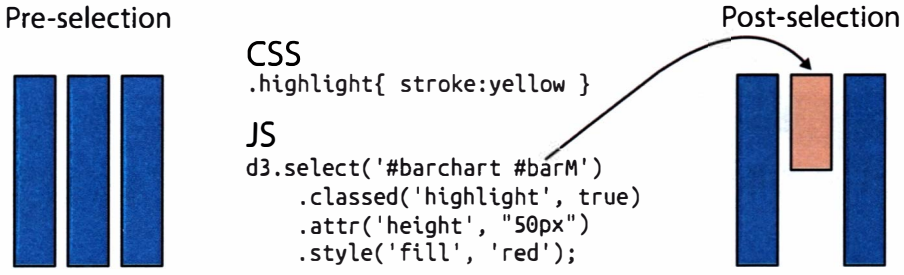


Рис. 17.4. *Установка атрибутов и стиля*

Метод D3 `text` задает текстовое содержимое для таких тегов DOM-элементов, как `div`, `p`, заголовки `h*` и текстовые элементы SVG. Чтобы увидеть метод `text` в действии, создадим документ HTML с плейсхолдером заголовка:

```
<! DOCTYPE html>
<meta charset="utf-8">

<style>font-family: sans-serif;</style>

<body>
   <h2 id="title">title holder</h2>
</body>
```

На рисунке 17.5 (Before) показано, как выглядит полученная страница в браузере до изменений.

Теперь создадим CSS класс `fancy-title` с крупным полужирным шрифтом:

```
.fancy-title {
   font-size: 24px;
   font-weight: bold;
}
```

Теперь с помощью D3 выбираем заголовок title, добавляем к нему класс `fancy-title` и присваиваем тексту заголовка значение My Bar Chart:

```
d3.select('#title')
   .classed('fancy-title', true)
   .text('My Bar Chart');
```

На рисунке 17.5 (After) показан увеличенный и выделенный полужирным шрифтом заголовок.



Рис. 17.5. *Настройка текста и стиля с помощью D3*

С помощью выборок можно не только задавать свойства элементов DOM, но и получать информацию об этих свойствах. Исключив второй аргумент любого из перечисленных на рисунке 17.3 методов, мы получим информацию о настройке веб-страницы.

На рисунке 17.6 показано, как получить основные свойства прямоугольника SVG. Как мы увидим, для программной адаптации и настройки полезно получать такие атрибуты SVG-элементов, как `width` и `height`.



Рис. 17.6. *Получение информации о столбике rect*

На рисунке 17.7 показаны методы `html` и `text`, которые работают как геттеры. После создания небольшого списка (`silly-list`) мы используем D3, чтобы выбрать его и получить различные свойства. Метод `html` возвращает HTML-код дочерних элементов списка — тегов <li>. Метод `text` возвращает текст внутри списка без HTML-тегов. Обратите внимание, что для тегов родителя форматирование любого возвращаемого текста несколько запутано, но достаточно хорошо подходит для поиска одной-двух строк.

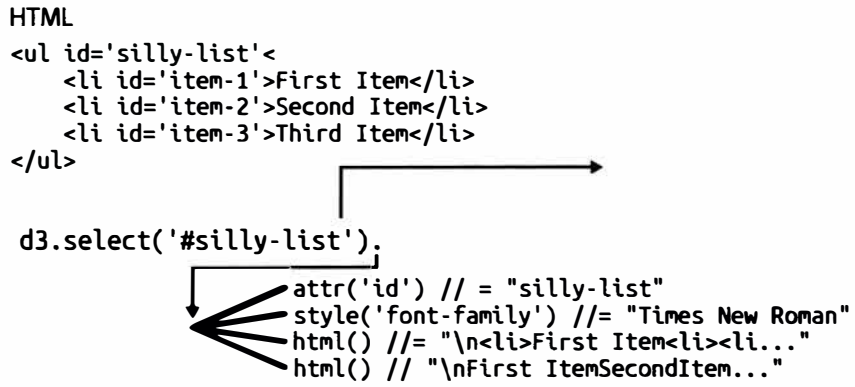


Рис. 17.7. *Получение HTML и текста из списка тегов*

До сих пор мы работали с атрибутами, стилями и свойствами существующих элементов DOM. Это полезный навык, но D3 окажется в своей стихии, когда мы начнем создавать элементы DOM программно, используя методы `append` и `insert`. Попробуем сделать это сейчас.

Добавление элементов DOM

Мы рассмотрели, как выбирать атрибуты, стили и свойства элементов DOM и управлять ими. Теперь узнаем, как можно с помощью D3 вставлять и добавлять элементы, программно изменяя DOM-дерево.

Начнем с HTML-каркаса, содержащего плейсхолдер `nobel-bar`:

```
<! DOCTYPE html>
<meta charset="utf-8">
<link rel="stylesheet" href="style.css" />

<body>
   <div id='nobel-bar'></div>

   <script
      src="https://cdnjs.cloudflare.com/ajax/libs/d3/7.3.1/d3.min.js">
   </script>
   <script type="text/javascript" src="script.js"></script> ❶
</body>
```

❶ *script.js* — это файл, куда мы добавим JavaScript-код нашей столбчатой диаграммы.

С помощью CSS-кода, хранящегося в файле *style.css*, зададим размер элемента `nobel-bar`:

```
#nobel-bar {
   width: 600px;
   height: 400px;
}

.bar {
   fill: blue; /* синие столбики для этой главы */
}
```

Обычно при создании D3-диаграммы первым делом обеспечивают для нее рамку SVG. Для этого добавляют элемент области рисования `<svg>` к `div`-контейнеру диаграммы. Затем к `<svg>` добавляют группу `<g>`для размещения элементов диаграммы (в нашем случае — столбиков). У этой группы есть внешние отступы для размещения осей, меток осей и заголовков.

Получение размеров элемента

При программировании в D3 или любой другой JS-библиотеке визуализации очень часто ширина и высота SVG или HTML-элемента используются как базис для установки размера элементов компонентов. Один из способов — использовать метод D3 `style`, принимающий CSS-свойства (размеры), а затем с помощью функции `parseInt` получить целочисленные значения:

```
/* CSS: #nobel-bar {width: 600px}; */

var width = parseInt(d3.select('#nobel-bar')
.style('width'), 10); ❶
```

❶ Метод `style` возвращает строку `'600px'`, которую мы с помощью функции `parseInt` преобразуем в число 600.

Как правило, метод хорошо работает, хотя `parseInt` действует довольно кустарно, превращая строку `'600px'` в число 600. Если ширина указана в процентах относительно контейнера родителя, то метод не сработает.

Более надежный и, возможно, лучший подход — получить размеры ограничивающей рамки (bounding box) интересующего вас элемента HTML или SVG. Метод getBBox дает ширину, высоту и относительную позицию элемента, используя метод `node` для получения элемента DOM:

```
let ele = d3.select("#chart-holder")
// Для HTML элемента, начиная с D3 +ele+ selection
let bRect = ele.node().getBoundingClientRect();
// например, размеры bRec {width: 600, height: 400, …}
// Для элемента SVG используется метод getBBox:
let bBox = eleSVG.node().getBBox();
// например, размеры bBox {width: 600, height: 400, x: 100,
// y: 100}
```

Традиционно указывается внешний отступ диаграммы в объекте `margin`, а затем из величины отступа и указанных в CSS ширины и высоты контейнера диаграммы выводится ширина и высота группы диаграммы. Нужный для этого код JavaScript приведен в примере 17-1.

***Пример 17-1.** Получение размеров столбчатой диаграммы*

```
var chartHolder = d3.select("#nobel-bar");

var margin = {top:20, right:20, bottom:30, left:40};

var boundingRect = chartHolder.node()
   .getBoundingClientRect(); ❶
var width = boundingRect.width - margin.left - margin.right,
height = boundingRect.height - margin.top - margin.bottom;
```

❶ Получаем прямоугольник, ограничивающий панель нашей столбчатый диаграммы, и на его основе задаем ширину и высоту контейнера для группы столбиков диаграммы.

Получив ширину и высоту группы столбиков, используем D3 для построения рамки диаграммы, добавляя необходимые теги <svg> и <g>, указывая размер области для рисования SVG и перемещение группы столбцов:

```
d3.select('#nobel-bar').append("svg")
    .attr("width", width + margin.left + margin.right)
    .attr("height", height + margin.top + margin.bottom)
    .append("g").classed('chart', true)
    .attr("transform", "translate(" + margin.left + ","
                                    + margin.top + ")");
```

В результате изменяется HTML-контент блока `nobel-bar`:

```
...
    <div id="nobel-bar">
        <svg width="600" height="400">
            <g class="chart" transform="translate(40, 20)"></g>
        </svg>
    </div>
...
```

Полученная SVG-рамка показана на рисунке 17.8. Ширину и высоту элемента <svg> составляет сумма размеров его дочерней группы и окружающих внешних отступов. Свойство `transform` позволяет перемещать дочернюю группу с помощью функции translate на количество пикселей `margin.left` вправо и на количество пикселей `margin.top` вниз (по соглашению SVG положительное направление Y уходит вниз).

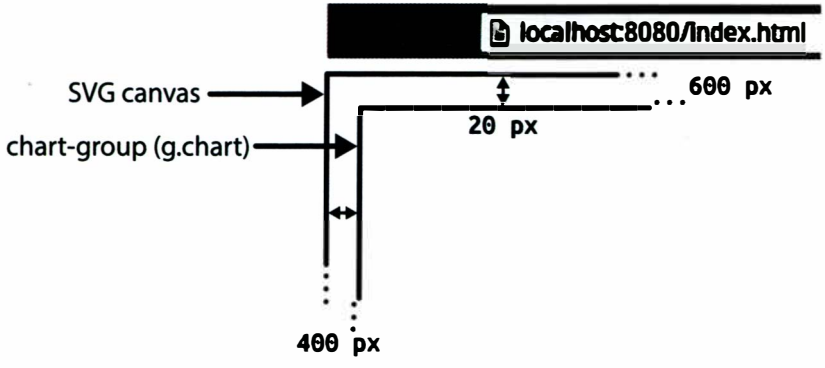


Рис. 17.8. *Построение рамки столбчатой диаграммы*

Построив рамку, добавим в нее столбики с помощью append. Мы используем тестовые данные — массив объектов верхнего среза стран с наибольшим количеством нобелевских лауреатов:

```
var nobelData = [
   {key:'United States', value:336},
   {key:'United Kingdom', value:98},
   {key:'Germany', value:79},
   {key:'France', value:60},
   {key:'Sweden', value:29},
   {key:'Switzerland', value:23},
   {key:'Japan', value:21},
   {key:'Russia', value:19},
   {key:'Netherlands', value:17},
   {key:'Austria', value:14}
];
```

Чтобы построить черновую столбчатую диаграмму[1], пройдем в цикле по массиву nobelData, добавляя столбики по очереди в конец группы диаграммы. Код см. в примере 17-2. После того, как создана базовая рамка диаграммы, просматриваем массив nobelData, используя поля value для установки высоты и положения столбца по оси Y. На рисунке 17.9 показано, как значения объекта используются при добавлении столбиков в группу диаграммы. Обратите внимание: у SVG нисходящая ось Y, поэтому, чтобы расположить столбчатую диаграмму правильно, нужно сместить каждый столбик на расстояние равное высоте диаграммы минус высота столбика. Далее мы увидим, как с помощью функций D3 scale можно свести к минимуму такие геометрические расчеты.

Пример 17-2. *Построение черновика столбчатой диаграммы с помощью append*

```
var buildCrudeBarchart = function() {

   var chartHolder = d3.select("#nobel-bar");

   var margin = {top:20, right:20, bottom:30, left:40};
   var boundingRect = chartHolder.node().getBoundingClientRect();
```

[1] Чуть позже в этой главе, когда включим D3 на полную мощность и начнем привязку данных, мы займемся осями, метками и др.

```
    var width = boundingRect.width - margin.left - margin.right,
    height = boundingRect.height - margin.top - margin.bottom;
    var barWidth = width/nobelData.length;

    var svg = d3.select('#nobel-bar').append("svg")
       .attr("width", width + margin.left + margin.right)
       .attr("height", height + margin.top + margin.bottom)
       .append("g").classed('chart', true)
       .attr("transform", "translate(" + margin.left + ","
                                       + margin.top + ")");

    nobelData.forEach(function(d, i) { ❶
       svg.append('rect').classed('bar', true)
          .attr('height', d.value)
          .attr('width', barWidth)
          .attr('y', height - d.value)
          .attr('x', i * (barWidth));
    });
};
```

❶ Итерируем по каждому объекту `nobelData`. Метод `forEach` передает объект и индекс массива в анонимную функцию.

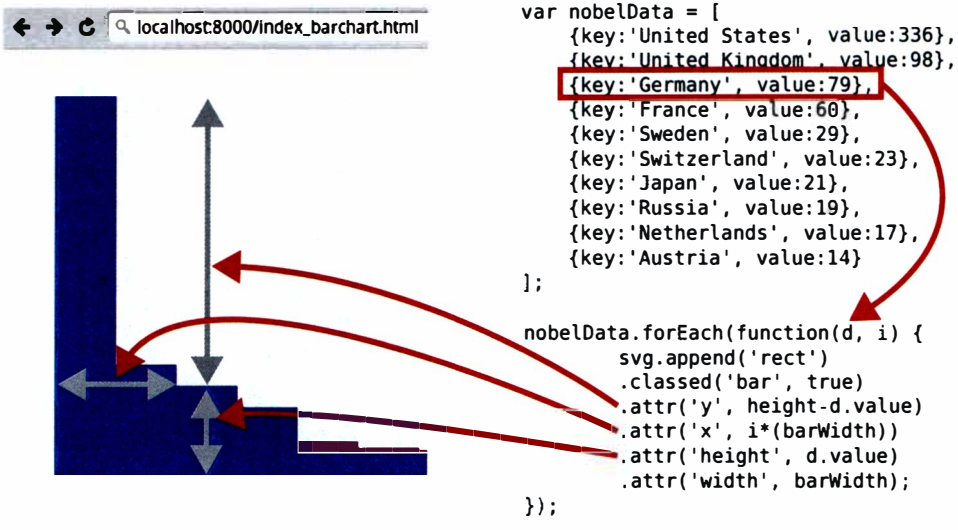


Рис. 17.9. *Программирование базовой столбчатой диаграммы с помощью D3*

SKLADCHIK.ORG

Также D3 может добавлять элементы в дерево DOM с помощью метода `insert`. Этот метод похож на `append`, но принимает второй селекторный аргумент для указания, перед какой позицией в последовательности тегов разместить элементы, например, в начале упорядоченного списка. На рисунке 17.10 показано использование `insert`: список элементов `silly-list` выбирается так же, как и в случае с `append`, а затем второй аргумент (например, `': first-child'`) задает позицию вставки.

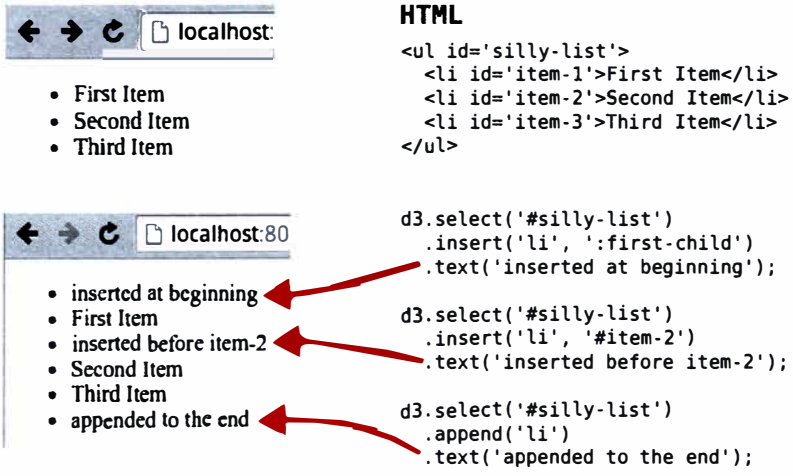


Рис. 17.10. *Добавление списка элементов с помощью метода insert из D3.js*

Может показаться, что для элементов SVG, расположенных непосредственно внутри родительской группы с использованием координат X и Y, метод `insert` не нужен. Но, как говорилось в подразделе «Создание слоев и прозрачность» на стр. 115, для SVG важен порядок DOM, поскольку элементы располагаются слоями, то есть последний добавленный элемент перекрывает предыдущие. Пример этого мы увидим в главе 19, где показано наложение сети координат (или `graticule`) на карту мира. Чтобы эта сетка отображалась поверх остальных элементов карты, нужно использовать функцию `insert`.

Черновую столбчатую диаграмму (см. рисунок 17.9) требуется усовершенствовать. Рассмотрим, как можно ее улучшить, сначала с помощью объектов `scale`, а затем с помощью грандиозной идеи D3 — привязки данных.

Использование D3

В примере 17-2 с помощью D3 мы построили базовую, без всяких излишеств, столбчатую диаграмму. У диаграммы есть ряд недостатков. Во-первых, цикл по массиву данных выглядит громоздко. А что, если мы захотим адаптировать набор данных для нашей диаграммы? Нам понадобится какой-то механизм добавления/удаления столбиков и последующего обновления и перерисовки столбиков в соответствии с новыми данными. Нам также потребуется масштабировать размеры столбиков по осям X и Y, чтобы отобразить различное число столбиков и их максимальные значения. Для этого потребуется немало подсчетов, в которых можно быстро запутаться. Кроме того, где-то нужно хранить измененные наборы данных. Каждое изменение диаграммы, вызванное изменением данных, потребовало бы передачи набора данных и создания цикла для обработки элементов. Получается, что данные существуют вне цепочки рабочего процесса D3, хотя на самом деле должны быть его неотъемлемой частью.

Изящное решение для интеграции набора данных обеспечивает концепция привязки данных — ключевая идея D3. Проблемы масштабирования решаются с помощью одной из самых полезных служебных библиотек D3 — d3-scale. Сейчас мы рассмотрим ее, а затем раскроем потенциал D3, используя привязку данных.

Шкалы в D3: от данных к их визуальному представлению

Основная идея шкал D3 (scale) — отображение интервала входных значений, который называется доменом (domain), в выходной диапазон (range). Эта простая процедура устраняет множество сложных аспектов при построении диаграмм и визуализаций. Чем лучше вы будете разбираться в шкалах, тем в большем числе ситуаций сможете их применять. Их освоение — ключ к эффективной работе с D3.

D3 предоставляет множество типов шкал, подразделяющихся на три основные категории: количественные, порядковые и временные[1]. Хотя существуют экзотические варианты для особых случаев, чаще всего используются линейные и порядковые шкалы.

При использовании D3-конструкция scale может показаться немного странной, поскольку это одновременно и объект, и функция. После создания шкалы

[1] Полный список см. на странице D3 GitHub.

вы можете вызывать различные методы, чтобы задать ее свойства (например, domain, который определяет интервал ее входных значений), но также можете вызывать ее как функцию с аргументом domain для возврата значения выходного диапазона range. Пример ниже должен прояснить эти особенности:

```
var scale = d3.scaleLinear(); // создание линейной шкалы
scale.domain([0, 1]).range([0, 100]); ❶
scale(0.5) // возвращается 50 ❷
```

❶ Используем методы шкалы domain и range, чтобы отобразить интервал 0 → 1 в диапазон 0 → 100.

❷ Вызываем scale как функцию с аргументом domain 0.5, функция возвращает значение range 50.

Рассмотрим еще два основных вида шкал — количественную и порядковую, которые мы применим к нашей столбчатой диаграмме.

Количественные шкалы

Количественная шкала (например, linear) отображает непрерывный домен (domain) в непрерывный диапазон (range) и обычно используется при построении линейных и столбчатых диаграмм, диаграмм рассеивания и др. Предположим, что высота столбиков и значения массива nobelData находятся в линейной зависимости. Сопоставляем диапазон высот столбиков (400–0 пикселей) с доменом между наименьшим возможным значением лауреатов в стране (0) и наибольшим значением в массиве (336 в США). В коде ниже сначала с помощью метода d3.max получим наибольшее значение в массиве nobelData, которое будет конечной границей домена:

```
let maxWinners = d3.max(nobelData, function(d){
                    return +d.value; ❶
                });

let yScale = d3.scaleLinear()
              .domain([0, maxWinners]) /* [0, 336] */
              .range([height, 0]);
```

❶ Префикс + преобразует строку в число (что часто требуется для JSON-данных).

Маленькая хитрость: диапазон range идет от максимума к минимуму, потому что мы будем использовать его для указания положительного смещения вниз вдоль оси Y в SVG, чтобы ось Y столбчатой диаграммы была направлена вверх (чем меньше высота столбика, тем большее смещение по Y потребуется). И наоборот, самый большой столбик (число лауреатов из США) вообще не смещен (см. рисунок 17.11).

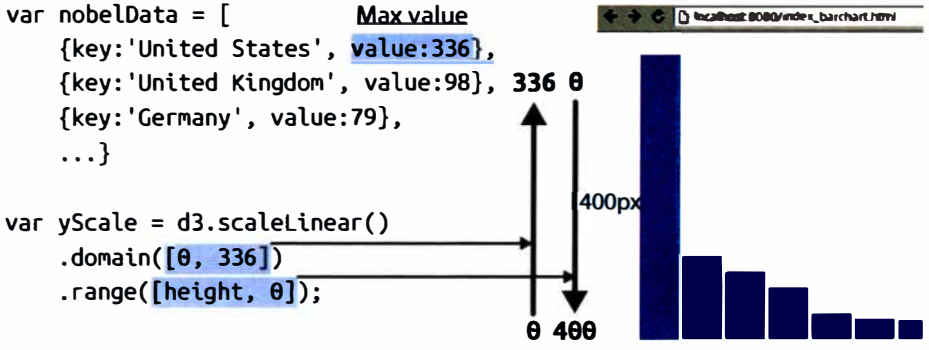


Рис. 17.11. *Применение линейной шкалы для фиксации домена и диапазона столбчатой диаграммы.*

Мы используем простейшую линейную шкалу для оси Y столбчатой диаграммы, отображая один числовой диапазон в другой, но линейные шкалы в D3 способны сделать гораздо больше. Ключом к их пониманию является метод `d3.interpolate`[1], который принимает два значения и возвращает интерполятор между ними. Итак, для диапазона `yScale` на рисунке 17.11 метод `interpolate` возвращает числовой интерполятор для значений `400` и `0`:

```
var numInt = d3.interpolate(400, 0);

numInt(0); // 400 ❶
numInt(0.5); // 200
numInt(1); // 0
```

❶ Интерполяторы по умолчанию имеют домен [0,1].

Метод `interpolate` работает не только с числами, успешно обрабатывая также строки, цветовые коды и даже объекты. Можно указать для массива

[1] Подробнее см. в документации D3.

domain более двух чисел, просто чтобы убедиться, что у него и массива range одинаковый размер[1]. Скомбинируем обе названные возможности, чтобы создать цветовую карту[2].

```
var color = d3.scaleLinear()
   .domain([-1, 0, 1])
   .range(["red", "green", "blue"]);

color(0) // "#008000" hex-код зеленого
color(0.5) // "004080" сине-голубой
```

Линейные шкалы в D3 имеют множество полезных методов и богатую функциональность. Скорее всего, основным вашим инструментом будут числовые карты (numeric maps), но я рекомендую прочесть документацию D3, чтобы в полной мере оценить гибкость линейной шкалы. Там же представлены другие количественные шкалы практически для всех случаях количественных исследований:

— степенные d3.scale.pow() — возвращают определенную степень числа (например, квадратный корень `sqrt`);
— логарифмические d3.scale.log() — позволяют использовать логарифмическую интерполяцию;
— шкалы квантования d3.scale.quantize() — преобразуют непрерывные входные числовые данные в дискретный диапазон. Результат разделен на сегменты (например, [1, 2, 3, 4, 5]);
— квантильные d3.scale.quantile() — часто используются для создания цветовых палитр. Отображают домен, указанный как дискретный набор выборочных значений, в дискретный диапазон;
— d3.scaleIdentity() — разновидность линейных шкал, где домен и диапазон идентичны (довольно экзотичная функция).

Количественные шкалы прекрасно подходят для манипулирования значениями из непрерывной области, но часто нам требуется получать значения из дискретной области (например, имена или категории). Для этого в D3 имеется специализированный набор порядковых шкал.

---

[1] D3 обрежет тот, который больше.

[2] В D3 есть множество встроенных цветовых карт и усовершенствованная обработка цветовых моделей RGB, HSL и др. В следующих главах мы некоторые из них применим.

Порядковые шкалы

Порядковые шкалы (ordinal scales) принимают массив значений в качестве домена и отображают в дискретные или непрерывные диапазоны, создавая для каждого из них единственное преобразованное значение. Чтобы явно создать однозначное отображение, используем метод `range`:

```
var oScale = d3.scaleOrdinal()
               .domain(['a', 'b', 'c', 'd', 'e'])
               .range([1, 2, 3, 4, 5]);

oScale('c'); // 3
```

Чтобы получить координаты по оси X для столбиков нашей диаграммы, отобразим массив индексов в непрерывный диапазон. Для этого подойдут метод `scaleBand` или метод `rangeRound`, который привязывает выходные значения к пикселям. Здесь мы используем `rangeRound`, чтобы преобразовать массив чисел в непрерывный диапазон, с округлением до целых значений пикселей:

```
var oScale = d3.scaleBand()
               .domain([1, 2, 3, 4, 5])
               .rangeRound([0, 400]);

oScale(3) // 160
oScale(5) // 320
```

Когда мы строили черновую столбчатую диаграмму (пример 17-2), то для определения размера столбиков использовали переменную `barWidth`. Для создания внутренних отступов (padding) между столбиками потребовалась бы переменная для них, а также корректировка `barWidth` и позиции столбиков. Используя нашу новую порядковую ленточную шкалу (Band Scale), мы легко реализуем эти задачи автоматически, без утомительных ручных вычислений. Метод `bandwidth` для шкалы X вернет подсчитанную ширину столбиков. Мы также можем использовать метод `padding`, чтобы указать отступы между столбиками как долю от пространства, занимаемого каждым столбиком. Значение `bandwidth` поменяется соответственно. Примеры:

```
var oScale = d3.scaleBand()
               .domain([1, 2]); ❶
```

```
oScale.rangeRound([0, 100]); ❷
oScale(2); // 50
oScale.bandwidth(); // 50


oScale.rangeRound([0, 100]); oScale.padding(0.1) // pBpBp ❸
oScale(1); // 5
oScale(2); // 52
oScale.bandwidth(); // 42, ширина столбика без отступов
```

❶ Сохраняем шкалу с фиксированным доменом. Это полезно, если ожидаются изменения диапазона.

❷ `rangeRound` округляет выходные значения до целочисленных.

❸ Задаем коэффициент отступа (`p`) = `0.1` * выделенное для столбика пространство(`B`).

На рисунке 17.12 показана шкала X столбчатой диаграммы с коэффициентом `0.1` для внутренних отступов. Непрерывный диапазон, равный 600 (пикселям), — ширина столбчатой диаграммы, а домен — массив целых чисел, соответствующих отдельным столбикам. Как показано на рисунке, переданное `xScale` значение с числовым индексом столбика возвращает его позицию на оси X.

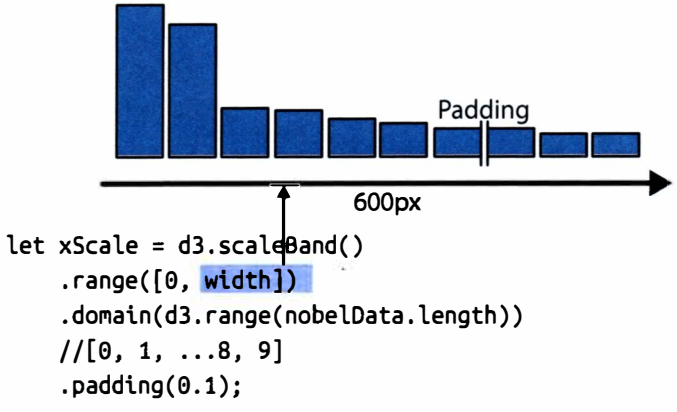


Рис. 17.12. *Установка домена и диапазона для шкалы X столбчатой диаграммы, с использованием коэффициента отступов 0.1*

Ознакомившись с D3-шкалами, перейдем к ключевой концепции — привязке данных к DOM для управления изменениями.

Привязка данных к элементам DOM — главное преимущество D3

D3 — это сокращение от *Data-Driven Documents* («документы, управляемые данными»), а мы до сих пор по-настоящему не работали с нашими данными. Полный потенциал D3 раскрывается при использовании его основной идеи — привязки данных (англ. binding/joining) к DOM-элементам и обновления веб-страницы (документа) на основе этой интеграции. В комбинации с методами `enter` и `exit` этот шаг открывает огромные возможности.

Начиная с версии 5, в D3 появился метод `join`, который существенно упрощает использование `enter` и `exit`. В этой главе методу `join` мы уделим особое внимание.

Чтобы разобраться в тысячах примеров в интернете, использующих старые паттерны обновления `enter-exit-remove`, важно знать внутреннюю логику D3 при объединении данных. Подробнее см. Приложение A.

Обновление DOM при изменении данных

Надо отметить, что раньше было непросто понять, как D3 обновляет DOM при добавлении новых данных (вы убедитесь в этом, если попробуете написать об этом главу книги или обучить студентов). Было несколько реализаций, таких как общий паттерн обновлений (general update pattern), которые сами по себе прошли через ряд несовместимых форм. Так что имейте в виду — популярные примеры в интернете, в которых используется старая версия D3, направят вас по неверному пути.



Даже если вы планируете создавать на скорую руку диаграммы с одним процессом привязки данных, полезно спрашивать себя: «А что, если потребуется изменять данные динамически?» Если неясно, как такие изменения осуществить, то вы выбрали неудачную конструкцию D3. Вовремя это обнаружив, вы сможете провести аудит кода и внести необходимые изменения до того, как ситуация ухудшится. Это будет хорошим стимулом избавиться от привычки использовать устаревшие конструкции D3. Поскольку работа с D3 требует ремесленного подхода, постоянное соблюдение лучших практик окупится.

К счастью, последние версии D3 закрепили основные методы и сделали их намного проще. Теперь триаду ключевых методов D3 для объединения данных, — `enter`, `exit` и `remove`— можно заменить единственным методом `join`, передавая ему только необходимые аргументы. Давайте рассмотрим, как использовать эти четыре метода для обновления столбчатой диаграммы в ответ на новые данные, в данном случае данные о странах лауреатов Нобелевской премии.

По всей видимости, концепция привязки (объединения) данных — основная концепция D3. Набор данных (обычно массив объектов данных) используется для создания визуальных элементов, например, прямоугольных столбиков столбчатой диаграммы. Любое изменение данных приводит к изменению визуализации, например, меняется высота или позиционирование существующих столбиков, или их количество. Можно подразделить эту операцию на три этапа:

1. Создание визуального элемента для любых данных, к которым еще не применялся `enter`;
2. Обновление атрибутов и стилей этого элемента, а если потребуется, то и других уже существующих элементов;
3. Удаление с помощью методов `exit` и `remove` всех старых элементов, к которым больше не привязаны данные.

В старых версиях D3 требовалось самостоятельно реализовывать паттерн обновления, используя методы `enter`, `exit`, и (недолго) `merge`, теперь метод `join` комбинирует эти методы в одном удобном для пользователя пакете. Можно вызывать join только с одним аргументом, определяющим, какой визуальный элемент будет привязан к данным (например, прямоугольник или круг SVG), но он также позволяет передавать коллбэк-функции `enter`, `exit` и `update`, обеспечивая более гибкое управление.

Посмотрим, насколько просто теперь объединить данные и визуальные элементы, привязав несколько горизонтальных столбиков, созданных из прямоугольников SVG, к нашему фиктивному набору данных о лауреатах Нобелевской премии. Присоединим следующий набор данных к группе прямоугольников и используем его для создания нескольких горизонтальных столбиков. На CodePen вы можете найти рабочий пример кода.

```
let nobelData = [
   {key: "United States", value: 336},
   {key: "United Kingdom", value: 98},
   {key: "Germany", value: 79},
   {key: "France", value: 60},
```

```
    {key: "Sweden", value: 29},
    {key: "Switzerland", value: 23},
    {key: "Japan", value: 21}
];
```

С помощью HTML и CSS создадим группу SVG для размещения столбцов и зададим классу `bar` синюю заливку:

```
<div id="nobel-bars">
  <svg width="600" height="400">
    <g class="bars" transform="translate(40, 20)"></g>
  </svg>
</div>

<style>
  .bar {
    fill: blue;
  }
  </style>
```

Используя данные и HTML-каркас, посмотрим, как работает D3 `join`. Создадим функцию `updateBars`, которая принимает массив данных стран в виде пар «ключ-значение», и привяжем массив к SVG-прямоугольникам.

Итак, функция `updateBars` принимает массив данных и сначала с помощью метода `data` добавляет его к выборке класса `'bar'`. Как показано в примере 17-3, затем она методом `join` привязывает выборку столбиков `bars` к SVG-прямоугольникам.

***Пример 17-3.** Объединение данных стран с SVG bars*

```
function updateBars(data) {
// выбираем и храним группу SVG bars
let svg = d3.select("#nobel-bars g");
let bars = svg.selectAll(".bar").data(data);

bars
  .join("rect") ❶
  .classed("bar", true) ❷
  .attr("height", 10)
```

SKLADCHIK.ORG

```
    .attr("width", d => d.value)
    .attr("y", function (d, i) {
      return i * 12;
    });
}
```

❶ Привязывает все данные существующих столбиков (bars) к SVG-элементам `rect`.

❷ Метод `join` возвращает все существующие `rect`, которые мы затем обновим с помощью привязанных к ним данных.

После вызова метода `join` D3 выполняет все, что нужно, используя `enter`, `exit` и `remove` для синхронизации данных с визуальными элементами. Давайте несколько раз вызовем функцию `updateBars` с разными данными. Сначала возьмем первые четыре элемента из набора данных нобелевских лауреатов и обновим столбики:

```
updateBars(nobelData.slice(0, 4));
```

Получим такую картину:



Теперь обновим привязку данных, используя только первые два элемента массива:

```
updateBars(nobelData.slice(0, 2));
```



При вызове метода на этот раз мы получили два столбика, показанные на изображении выше. D3 автоматически удаляет избыточные элементы, которые больше не связаны с данными.

А теперь проверим, что произойдет, если на этот раз мы укажем первые шесть членов массива данных о лауреатах Нобелевской премии:

```
updateBars(nobelData.slice(0, 6));
```



И снова D3 выполнила то, на что мы рассчитывали — на этот раз добавила прямоугольники, привязанные к новым объектам данных (см. изображение выше).

Убедившись, что D3 синхронизирует данные и визуальные элементы, добавляя и удаляя прямоугольники по мере необходимости, мы можем создать основу для столбчатой диаграммы нобелевских лауреатов.

Сборка столбчатой диаграммы

Собрав воедино то, что мы узнали в этой главе, построим основные элементы нашей столбчатой диаграммы. Здесь нам также помогут D3-шкалы.

Сначала выбираем контейнер для столбчатой диаграммы по идентификатору *#nobel-bar* и используем его размеры (из `boundingClientRectangle`) и настройки отступов margin, чтобы получить ширину и высоту диаграммы:

```
let chartHolder = d3.select('#nobel-bar')
let margin = {top: 20, right: 20, bottom: 35, left: 40}
let boundingRect = chartHolder.node().getBoundingClientRect()
let width = boundingRect.width - margin.left - margin.right,
height = boundingRect.height - margin.top - margin.bottom
// левый padding метки оси y
var xPaddingLeft = 20
```

Зададим шкалы, используя ширину и высоту:

```
let xScale = d3.scaleBand()
   .range([xPaddingLeft, width]) // left-padding for y-label
   .padding(0.1)

let yScale = d3.scaleLinear().range([height, 0])
```

Создадим SVG-группу диаграммы, используя ширину, высоту и отступы margin, и сохраним ее в переменной:

```
var svg = chartHolder
   .append('svg')
   .attr('width', width + margin.left + margin.right)
   .attr('height', height + margin.top + margin.bottom)
```

```
    .append('g')
    .attr('transform', 'translate(' + margin.left + ','
                                    + margin.top + ')')
```

Подготовив HTML- и SVG-каркас, адаптируем функцию `updateBars` (см. пример 17-3) для реагирования на изменение наших реальных данных о лауреатах Нобелевской премии. Функция обновления получает массив данных в следующей форме:

```
[{key: 'United States', value: 336, code: 'USA'}
{key: 'United Kingdom', value: 98, code: 'GBR'}
{key: 'Germany', value: 79, code: 'DEU'} ...]
```

Вызванная с новыми данными, функция `updateBarchart` сначала отфильтровывает страны без лауреатов, затем обновляет домены шкал X и Y, чтобы отобразить нужное количество столбиков/стран и максимальное число премий, как показано в примере 17-4.

***Пример 17-4.** Обновление столбчатой диаграммы*

```
let updateBarChart = function (data) {
   // отфильтровываем страны, где нет лауреатов премии (value =0)
   data = data.filter(function (d) {
      return d.value > 0
   })
   // изменяем домены, чтобы отразить отфильтрованные данные
   // создаем массив кодов стран: ['USA', 'DEU', 'FRA' …]
   xScale.domain(
      data.map(d => d.code)
   )
   // определим шкалу, равную наибольшему числу полученных премий –
   //США: 336
   yScale.domain([
      0,
      d3.max(data, d => d.value)
   ])
// ...
}
```

Обновив шкалы, используем привязку данных для создания столбиков в соответствии с предоставленными данными. В целом это та же функция из примера 17-3, но с использованием шкал для определения размера столбиков, а также пользовательского метода `enter` для добавления класса и левого внутреннего отступа к вновь созданным столбикам:

```
let bars = svg
      .selectAll('.bar')
      .data(data)
      .join(
         (enter) => { ❶
            return enter
               .append('rect')
               .attr('class', 'bar')
               .attr('x', xPaddingLeft)
      }
   )
   .attr('x', d => xScale(d.code))
   .attr('width', xScale.bandwidth())
   .attr('y', d => yScale(d.value))
   .attr('height', d => height - yScale(d.value))
```

❶ Кастомизируем метод `enter`, чтобы добавить класс `'bar'` к прямоугольнику. Обратите внимание, что необходимо вернуть объект `enter` для использования после вызова после вызова `join`.

Теперь наша диаграмма реагирует на изменения данных, в данном случае инициированное пользователем. Результат фильтрации данных по всем Нобелевским премиям в области химии показан на рисунке 17.13. Хотя нескольких ключевых элементов еще не хватает, основную работа по созданию столбчатой диаграммы мы выполнили. Нанесем последние штрихи — добавим оси и несколько интересных эффектов перехода.



***Рис. 17.13.** Готовая столбчатая диаграмма*

Оси и метки

Теперь, когда у нас есть работающий паттерн обновления, добавим оси и метки осей, без которых не обходится ни одна уважающая себя диаграмма.

В D3 не так уж много высокоуровневых элементов диаграмм, что побуждает веб-разработчиков создавать собственные. Но удобный объект `axis` в D3 имеется, так что нам не придется мастерить элементы SVG. Объект просто использовать и, как и следовало ожидать, он отлично сочетается с нашими паттернами обновления данных, позволяя изменять деления осей и метки в соответствии с представленными данными.

Оси D3

Оси из библиотеки D3 сначала могут сбивать с толку, создавая ощущение некой магии. Лучше всего рассматривать их как плагин[1], который генерирует необходимый HTML для осей (линии, деления, подписи к делениям и др.) и надлежащим образом реагирует на изменения данных — при изменении диапазона шкал оси могут меняться плавно.

Обычно при использовании D3-оси нужно создать группу SVG, в которой она будет находиться, а затем вызывать ось на ней, задав шкалу, которую она будет представлять. При выполнении `call` объект оси *добавит* точный HTML в DOM. Следующий код демонстрирует простую настройку диаграммы с помощью SVG-группы `x-axis` и оси D3. Вызов оси в группе осей генерирует строки HTML и текст, необходимые для оси (см. рабочий пример CodePen).

```
<! DOCTYPE html>
<meta charset="utf-8">
<script src="static/libs/d3.min.js">
</script>
<style>
   svg {
      width: 600px;
      height: 400px
   }
</style>
```

[1] Оси следуют паттерну, схожему с предложенным Майком Бостоком в книге *Towards Reusable Charts*: для построения HTML-документа на основе выбранных элементов DOM используется JS-метод `call`.

```
<body>
   <svg>
      <g id='chart' transform='translate(20,20)'>
         <g id='x-axis'></g>
      </g>
   </svg>
   <script>
      var scale = d3.scaleLinear()
         .domain([0, 10]).range([0, 400]) ❶
      var xaxis = d3.axisBottom().scale(scale) ❷
      d3.select('#x-axis').call(xaxis) ❸
   </script>
</body>
```

❶ Создаем шкалу с доменом от 0 до 10 и интервалом из 400 (пикселей).

❷ Создаем нижнюю ось D3 с использованием заданной шкалы.

❸ Вызов D3-оси в группе `x-axis` создает HTML-ветвь оси, код которой показан на рисунке 17.5. Выглядит ось, как на рисунке 17.14.

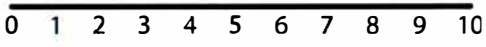


Рис. 17.14. *Простая D3-ось*

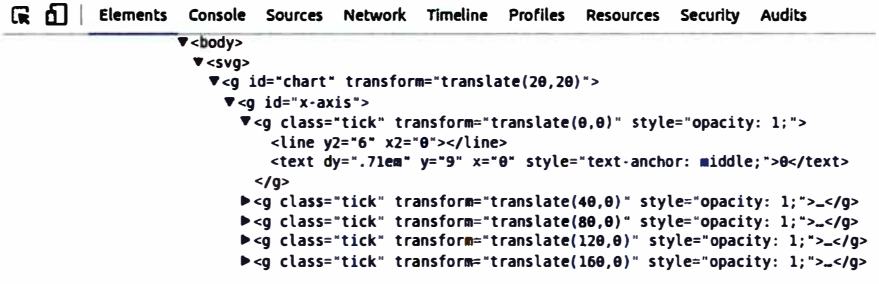


Рис. 17.15. *HTML-ветвь оси, созданная экземпляром D3-оси*

Использование метода call для выборки — обычная практика D3 при создании плагинов, например для кистей и всплывающих подсказок. На *bl.ocks.org* есть немало примеров, демонстрирующих этот паттерн. К примеру, такой, от Карла Бота (Charl Botha). Главное понимать, что это не магия, и что важно разобраться в базовых процессах, происходящих при вызове метода `call`.

Чтобы определить оси X и Y, нужно знать, какие диапазоны и домены эти оси должны представлять. В нашем случае они совпадают с диапазонами и доменами наших шкал X и Y, поэтому мы передаем их методу осей `scale`. D3 также позволяет указать ориентацию осей, что дает возможность фиксировать относительное позиционирование делений и подписей к ним. Мы хотим, чтобы ось X нашей столбчатой диаграммы располагалась снизу, а ось Y — слева. На порядковой оси X будут иметься метки для каждого столбика, но выбор числа делений оси Y может быть произвольным. Десять кажется подходящим числом, так что установим его с помощью метода `ticks`. Код ниже показывает, как мы объявляем оси столбчатой диаграммы:

```
let xAxis = d3.axisBottom().scale(xScale)

let yAxis = d3
   .axisLeft()
   .scale(yScale)
   .ticks(10)
   .tickFormat(function (d) {
      if (nbviz.valuePerCapita) { ❶
         return d.toExponential()
      }
      return d
   })
```

❶ Формат подписей делений должен меняться в зависимости от выбранного показателя: на душу населения или абсолютного. На душу населения приходится число, которое лучше всего представить в экспоненциальной форме (например, 0.000005 → 5e-6). Метод `tickFormat` принимает значение данных на каждом делении и возвращает для него нужную строку.

Добавим немного CSS для правильного оформления осей: удалим заливку (`fill`) по умолчанию, зададим черный цвет обводки и обеспечим четкую отрисовку формы. Попутно укажем размер и семейство шрифта:

```
.axis {font: 10px sans-serif;}
.axis path,.axis line {
   fill: none;
   stroke: #000;
```

```
    shape-rendering: crispEdges;
}
```

Получив генераторы объектов `axis`, мы создадим пару групп SVG, которые будут содержать в себе полученные оси. Добавим их в наш основной `svg` селектор как группы с понятными именами классов:

```
svg.append("g")
   .attr("class", "x axis")
   .attr("transform", "translate(0," + height + ")"); ❶

svg.append("g")
      .attr("class", "y axis");
```

❶ По умолчанию в SVG ось Y идет вниз от верхнего левого угла с координатами (0, 0). Поскольку мы хотим разместить ось x *снизу*, нам нужно опустить ее от верхнего края на высоту диаграммы в пикселях (*height*).

У осей нашей столбчатой диаграммы фиксированные диапазоны (ширина и высота диаграммы), но домены будут меняться в зависимости от того, какие фильтры пользователь применяет к набору данных. Например, пользователь выбрал номинацию «Экономика» — количество столбиков (стран) уменьшилось, следовательно, изменились домен порядковой шкалы X (количество столбиков) и домен количественной шкалы Y (максимальное количество лауреатов). Мы хотим, чтобы оси динамически обновлялись вместе с доменами, с плавными переходами для лучшей визуализации.

В примере 17-5 показано, как обновлять оси. Сначала обновим домены шкал с помощью новых данных (A). Новые домены шкал отображаются при вызове связанных генераторов осей на соответствующих группах.

Пример 17-5. *Обновление осей столбчатой диаграммы*

```
let updateBarChart = function(data) {
   // A. Обновим домены шкал новыми данными
   xScale.domain(data.map(d => d.code));
   yScale.domain([0, d3.max(data, d => +d.value)])
   // B. Используем генераторы осей с новыми доменами
   svg.select('.x.axis')
      .call(xAxis) ❶
```

```
        .selectAll("text") ❷
        .style("text-anchor", "end")
        .attr("dx", "-.8em")
        .attr("dy", ".15em")
        .attr("transform", "rotate(-65)");

    svg.select('.y.axis')
    .call(yAxis);
    // ...
}
```

❶ Вызов метода D3-осей `axis` для элемента группы оси X создает все необходимые SVG-элементы оси, включая деления и подписи к ним. Метод `axis` использует внутренний паттерн обновления для активации переходов к только что привязанным данным.

❷ После создания оси X вносим изменения в SVG сгенерированных текстовых меток. Сначала выбираем элементы `text` оси, то есть подписи делений. Затем размещаем якоря текста в конце элемента и слегка меняем позиционирование. Это необходимо, так как текст вращается относительно своей точки привязки: мы хотим повернуть метки стран, расположенные под линиями делений, вокруг их конца. Результат этих изменений см. на рисунке 17.16. Примечание: без вращения метки накладывались бы друг на друга.

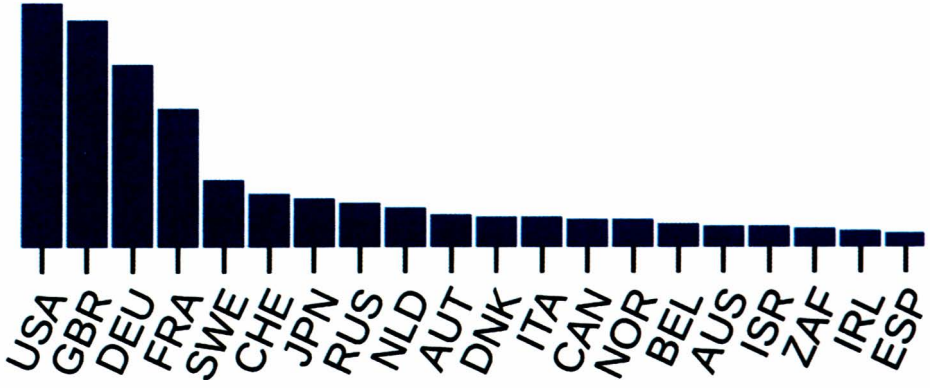


Рис. 17.16. *Переориентированные подписи делений на оси X*

Теперь добавим подпись к оси X и проверим работу диаграммы с реальными данными:

```
let xPaddingLeft = 20 ❶

let xScale = d3.scaleBand()
   .range([xPaddingLeft, width])
   .padding(0.1)
//...
svg.append("g")
   .attr("class", "y axis")
   .append("text")
   .attr('id', 'y-axis-label')
   .attr("transform", "rotate(-90)") ❷
   .attr("y", 6)
   .attr("dy", ".71em") ❸
   .style("text-anchor", "end")
   .text('Number of winners');
```

❶ Константа левого отступа (в пикселях), обеспечивающая место для подписи оси Y.

❷ Поворачиваем текст против часовой стрелки до вертикального положения.

❸ `dy` — относительная координата [относительно только что заданной координаты y (6)]. Используя единицу `em` (относительно размера родительского шрифта), мы можем гибко настраивать внешние отступы и базовую линию текста.

На рисунке 17.17 показан результат фильтрации набора данных о лауреатах Нобелевской премии по номинации «Химия». Ширина столбцов увеличивается пропорционально уменьшению числа стран, а обе оси адаптируются под новый набор данных.

Теперь наша столбчатая диаграмма использует паттерн обновления, самостоятельно реагируя на изменение пользователем набора данных. Хотя функционал работает, переход между состояниями выглядит резким и диссонирующим. Один из способов сделать изменения более интересными и даже информативными — добиться, чтобы диаграмма плавно обновлялась в течение короткого периода времени, при этом сохраненные столбцы стран перемещались бы из старого положения в новое, одновременно меняя свою высоту и ширину. Такие непрерывные переходы оживляют визуализацию, их можно увидеть во многих самых впечатляющих работах с использованием D3. Преимущество D3 в глубокой интеграции переходов: замечательные визуальные эффекты достигаются буквально несколькими строками кода.

SKLADCHIK.ORG

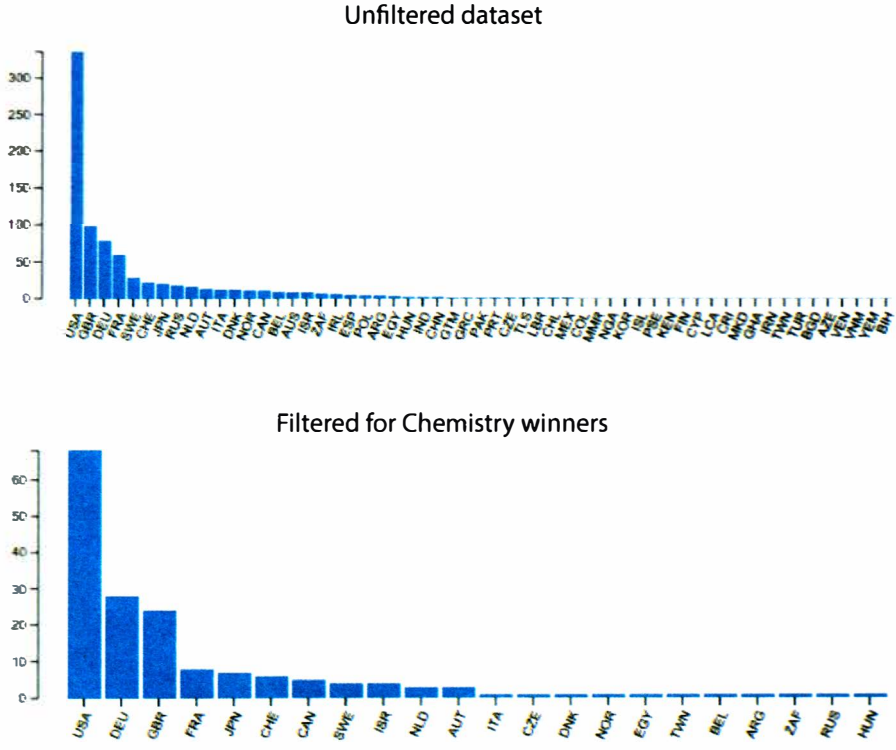


Рис. 17.17. *Столбчатая диаграмма до и после применения фильтра по номинации «Химия»*

Переходы

В нынешнем виде наша столбчатая диаграмма полностью функциональна. Она реагирует на изменения данных, добавляя или удаляя столбики, а затем обновляя их размеры в соответствии с новыми данными. Однако скачкообразный переход от одного отображения данных к другому выглядит резким и раздражающим.

Переходы, предоставляемые D3, сглаживают визуальное обновление элементов, заставляя их плавно меняться в течение заданного периода времени. Эстетичная плавность переходов не только улучшает восприятие[1], но и повышает вовлеченность пользователя — это веская причина их освоить.

[1] Например, когда мы переключаемся с абсолютных показателей премий по странам на показатели на душу населения, то большой объем перемещений, отображаемых при изменении порядка столбиков стран, подчеркивает разницу между двумя показателями.

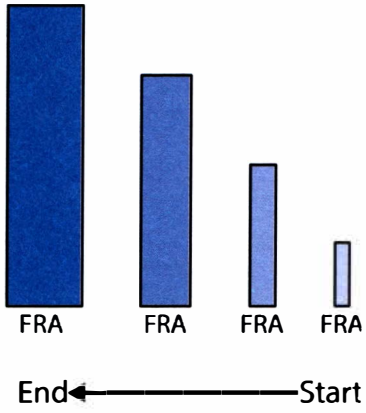


Рис. 17.18. *Плавные переходы столбиков при обновлении*

На рисунке 17.18 показан эффект, к которому мы стремимся. Когда столбчатая диаграмма обновляется в соответствии с только что выбранным набором данных, столбики всех стран, которые были до и после перехода, должны плавно морфировать от старых позиций и размеров в новые[1]. На рисунке 17.18 столбик, изображающий Францию, постепенно растет в течение перехода (скажем, за пару секунд), а промежуточные столбики демонстрируют увеличение ширины и высоты. Деления и подписи осей также будут адаптироваться по мере изменения масштабов X и Y.

```
bars = svg.selectAll(".bar")
        .data(data);
```

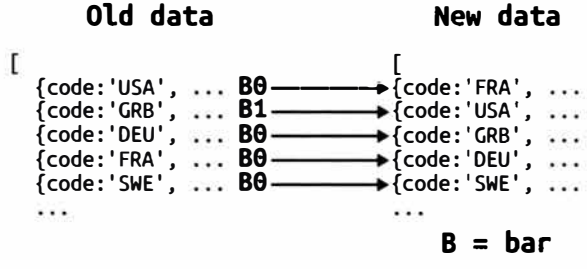


Рис. 17.19. *Новые данные по умолчанию привязываются по индексу*

Эффекта, показанного на рисунке 17.18, достичь на удивление легко, но требуется точно понимать, как объединяются данные в D3. По умолчанию новые

[1] Специалистам по анимации и компьютерной графике этот эффект известен как твининг/tweening (см. страницу в Википедии).

данные привязываются к существующим элементам DOM по индексам массива. На рисунке 17.19 на примере выбранных столбиков показано, как работает привязка. Первый столбик (B0), первоначально привязанный к данным США, теперь привязан к данным Франции. Столбик остается на первой позиции, но обновляет размер и метку деления. То есть столбик США превращается в столбик Франции[1].

Чтобы обеспечить непрерывность во время переходов (то есть, чтобы столбик США перемещался в новое положение, одновременно меняя высоту и ширину) новые данные следует привязать не по индексу, а по уникальному ключу. D3 позволяет передать функцию в качестве второго аргумента методу `data`, возвращающему ключ объекта data, который используется для привязки новых данных к соответствующим столбцам (предполагается, что они все еще существуют). На рисунке 17.20 показано, как это происходит. Теперь столбик (B0) привязан к новым данным США, поменяв свою позицию по индексу, ширину и высоту в соответствии с новым американским столбиком.

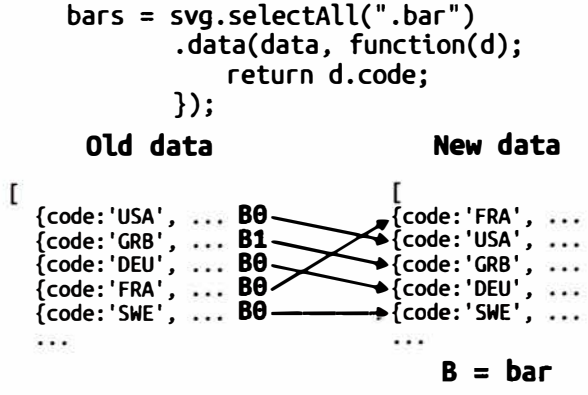


Рис. 17.20. *Использование ключа объекта для привязки новых данных*

Привязка данных по ключу дает нам правильные начальные и конечные точки для столбиков стран. Теперь остается найти способ создания плавного перехода между ними. Используем для этого два замечательных метода D3: `transition` и `duration`. Вызовем их до того, как изменим атрибуты размера и позиции столбика, и D3 чудесным образом выполнит плавный переход между состояниями, как показано на рисунке 17.18. Несколько строк кода добавят переходы в процесс обновления столбчатой диаграммы:

[1] См. замечательную демонстрацию константности восприятия объекта от Майка Бостока на его сайте.

```
// nbviz_core.mjs
nbviz.TRANS_DURATION = 2000 // время в миллисекундах
// nbviz_bar.mjs
import nbviz from ./nbviz_core.mjs
//...
svg.select('.x.axis')
   .transition().duration(nbviz.TRANS_DURATION) ❶
   .call(xAxis) //...
//...
svg.select('.y.axis')
   .transition().duration(nbviz.TRANS_DURATION)
   .call(yAxis);
//...
var bars = svg.selectAll(".bar")
   .data(data, d => d.code) ❷
//...
let bars = svg.selectAll('.bar')
   .data(data, (d) => d.code)
   .join(
//...
)
.classed('active', function (d) {
   return d.key === nbviz.activeCountry
})
.transition()
.duration(nbviz.TRANS_DURATION)
.attr("x", (d) => xScale(d.code)) ❸
.attr("width", xScale.bandwidth())
.attr("y", (d) => yScale(d.value))
.attr("height", (d) => height - yScale(d.value));
```

❶ Константа `TRANS_DURATION` со значением 2000 (мс) задает для перехода длительность 2 секунды.

❷ Используем свойство `code` объекта данных для создания непрерывной привязки данных.

❸ Атрибуты `x`, `y`, `width` и `height` будут плавно морфировать от исходных значений до значений, определенных этим выражением.

Переходы работают с наиболее очевидными атрибутами и стилями существующего элемента DOM[1].

Рассмотренные переходы обеспечивают плавное изменение атрибутов от начальной точки до конечной цели, но можно сделать их эффектнее: D3 предлагает для этого множество настроек. Например, с помощью метода `delay` можно задать период задержки перед выполнением перехода. Задержка также может быть функцией данных.

Пожалуй, самый полезный дополнительный метод перехода — `ease`, позволяющий задавать алгоритм обновления атрибутов элементов в течение времени перехода. `CubicInOut` — функция плавности (easing-функция) используемая по умолчанию, но вы можете выбрать `Quad` для ускорения процесса перехода или `Bounce` и `Elastic`, которые в ответ на изменения выполняют примерно то, что обещают названия: bounce (англ. упругость), elastic (эластичность). Есть также синусоидальные функции плавности, ускоряющие переход в начале и замедляющие к концу. Неплохое описание различных easing-функций см. на сайте *easings.net*, а на observablehq — всесторонний обзор easing-функций D3 с использованием интерактивных диаграмм.

При необходимости вы можете создавать собственные функции плавности для реализации специфических требований. Метод `tween` предоставляет тонкий контроль доступа к данным, который может вам понадобиться.

Итак, применив паттерн обновления на основе `join` и некоторые интересные переходы, мы завершили создание столбчатой диаграммы Нобелевских премий. Всегда остается место для улучшений, но наша диаграмма отлично справляется со своей задачей. Прежде чем перейти к другим компонентам визуализации данных о лауреатах Нобелевской премии, подведем итог тому, что мы узнали из этой довольно большой главы.

Обновление столбчатой диаграммы

При импорте модуля столбчатой диаграммы к массиву коллбэков в основном модуле добавляется соответствующая коллбэк-функция. При обновлении данных в ответ на действия пользователя эта коллбэк-функция вызывается, и столбчатая диаграмма обновляется в соответствии с новыми данными по странам:

[1] Переходы применяются только к уже существующим элементам — например, нельзя плавно создать элемент DOM. Однако, применив CSS-свойсто opacity, можно сделать так, чтобы он постепенно появлялся и исчезал.

```
nbviz.callbacks.push(() => { ❶
   let data = nbviz.getCountryData() updateBarChart(data)
})
```

❶ При обновлении данных эта анонимная функция вызывается из основного модуля.

Резюме

Это была большая и довольно непростая глава. D3 — не самая простая для изучения библиотека, но я постарался облегчить процесс обучения, разбив материал на удобоваримые части. Не спеша усваивайте основные идеи и — главное — начинайте ставить перед собой небольшие задачи, чтобы расширить знания D3 на практике. Я считаю, что работа с D3 — это своего рода искусство, здесь, в отличие от большинства библиотек, учишься в процессе работы.

Самое главное, что надо усвоить в D3, чтобы эффективно ее применять, — паттерн обновления и привязка данных. Когда вы поймете их на базовом уровне, большинство других возможностей D3 логично встанут на свои места. Сосредоточьтесь на методах `data`, `enter`, `exit` и `remove` и убедитесь, что вы полностью понимаете, как они работают. Только так вы перейдете от копирования примеров к осознанному программированию на D3: начальная продуктивность этого подхода (благо примеров много) со временем сменяется разочарованием. Чтобы узнать узнать, какие данные привязаны к элементам DOM через переменную `__data__`, используйте консоль разработчика браузера (на данный момент лучшие инструменты — у Chrome и Chromium). Если что-то идет вопреки вашим ожиданиям, вы можете многому научиться, выясняя, почему так получилось.

Теперь у вас должно быть достаточно хорошее представление об основных техниках D3. В следующей главе мы опробуем эти новые навыки на более сложной визуализации — хронологии Нобелевских премий.

Визуализация отдельных премий

В главе 17 вы узнали об основах D3: как выбирать и изменять элементы DOM, как добавлять новые и как применять паттерн обновления данных — основу интерактивности в D3. В этой главе я расширю ваши знания и покажу, как создать уникальный визуальный элемент — хронологию всех отдельных Нобелевских премий по годам (рис. 18.1). Работая с временной диаграммой, мы расширим знания, полученные в предыдущей главе, и познакомимся с рядом новых техник, в том числе с более продвинутым манипулированием данными.

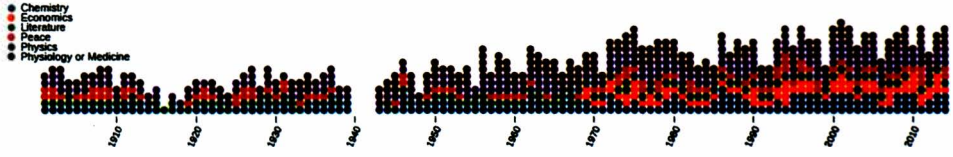


Рис. 18.1. *Цель этой главы — временная диаграмма вручения Нобелевской премии*

Начнем с создания HTML-структуры для временной диаграммы.

Создание структуры

Построение временной диаграммы аналогично построению столбчатой диаграммы, которую мы подробно рассмотрели в предыдущей главе. Сначала используем D3, чтобы выбрать контейнер `<div>` с id `nobel-time`, затем создадим группу диаграмм `svg`, используя ширину и высоту контейнера вместе с указанными нами внешними отступами:

```
import nbviz from './nbviz_core.mjs'

let chartHolder = d3.select('#nobel-time');

let margin = {top:20, right:20, bottom:30, left:40};
```

```
let boundingRect = chartHolder.node()
   .getBoundingClientRect();
let width = boundingRect.width - margin.left
- margin.right,
height = boundingRect.height - margin.top - margin.bottom;

let svg = chartHolder.append("svg")
      .attr("width", width + margin.left + margin.right)
      .attr("height", height + margin.top
      + margin.bottom)
      .append('g')
         .attr("transform",
               "translate(" + margin.left + ","
               + margin.top + ")");
   // ...
})
```

Получив `svg` группу диаграммы, добавим шкалы и оси.

Шкалы

Для размещения круглых индикаторов используем две порядковые scaleBand-шкалы (пример 18-1). Шкала X использует метод `rangeRoundBands`, задающего 10% отступ между кругами. Поскольку мы используем шкалу X, чтобы задать диаметр кругов, высота диапазона шкалы Y регулируется вручную, чтобы поместились все индикаторы с учетом отступов между ними. Для округления координат в пикселях до целочисленных значений используем `rangeRoundPoints`.

Пример 18-1. *Две ленточные шкалы диаграммы для осей X и Y*

```
let xScale = d3.scaleBand()
   .range([0, width])
   .padding(0.1) ❶
   .domain(d3.range(1901, 2015))

let yScale = d3.scaleBand()
   .range([height, 0]).domain(d3.range(15)) ❷
```

❶ Указываем padding 0.1, получая размер отступа приблизительно равный 10 % от диаметра индикатора.

❷ Домен [0, …, 15], где 15 — исторический максимум премий, присужденных за один год.

В отличие от столбчатой диаграммы из предыдущей главы, диапазоны и домены данной диаграммы фиксированные. Домен шкалы `xScale` — годы существования Нобелевской премии, а домен `yScale` — от нуля до максимального количества лауреатов в любом конкретном году (14 в 2000 году). Поскольку ни один из них не будет меняться в ответ на действия пользователя, мы определяем их вне метода `update`.

Оси

Поскольку в год награждается максимум 14 лауреатов, а каждому из них соответствует один круглый индикатор, то легко подсчитать количество лауреатов визуально. Учитывая это, а также акцент на отображении относительного распределения премий (например, резкого роста в США количества нобелевских лауреатов в области естественных наук после Второй мировой войны) и большую длину диаграммы, ось Y для нашей диаграммы избыточна.

Что касается оси X, то указывать временные метки с точностью до десятилетий представляется более-менее правильным. Так ось выглядит аккуратнее, к тому же такая нумерация часто используется в графиках, отображающих исторические процессы. В примере 18-2 показано создание оси X с помощью объекта D3 `axis`. Мы переопределяем значения делений с помощью метода `tickValues`, который возвращает только значения домена (1900–2015), заканчивающиеся на ноль.

***Пример 18-2.** Создание оси X с метками делений по десятилетиям*

```
let xAxis = d3.axisBottom()
   .scale(xScale)
   .tickValues(
      xScale.domain().filter(function (d, i) {
         return!(d % 10) ❶
      })
   )
```

❶ Чтобы ставить метки только на каждый десятый год, получим значения делений, фильтруя значения домена x-шкалы и используя их индексы для выбора значений, кратных 10. Если при делении по модулю 10 (`%10`) получаем 0, то логический оператор (`!`) возвращает `true`, то есть фильтр пройден.

Итак, если данное условие возвращает `true`, устанавливаем метку в начале каждого десятилетия.

Как и шкалы, оси не будут меняться[1], поэтому мы добавим их до получения набора данных в функции `updateTimeChart`:

```
svg.append("g") // группа для размещения оси
   .attr("class", "x axis")
   .attr("transform", "translate(0," + height + ")")
   .call(xAxis) ❶
   .selectAll("text") ❷
   .style("text-anchor", "end")
   .attr("dx", "-.8em")
   .attr("dy", ".15em")
   .attr("transform", "rotate(-65)");
```

❶ Вызываем D3 axis из группы `svg`, где объект `axis` автоматически строит элементы оси.

❷ Как и в секции «Оси и метки» на стр. 426, мы поворачиваем метки делений оси, размещая их по диагонали.

Мы завершили работу с осями и шкалами, остается только добавить небольшую легенду с цветовыми обозначениями номинаций и затем перейти к интерактивным элементам диаграммы.

Метки номинаций

Последний из *статических* компонентов — это легенда, содержащая наименования номинаций, показанные на рисунке 18.2.

[1] В D3 есть несколько удобных кистей, которые упрощают выбор участков оси X или Y. Применив их в сочетании с переходами, можно привлекательным и интуитивно понятным способом повысить разрешение больших наборов данных.

SKLADCHIK.ORG

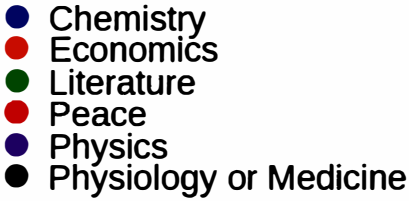


Рис. 18.2. Легенда номинаций

Чтобы создать легенду, сначала создадим для названий группу с классом `labels`. Привязываем данные `nbviz.CATEGORIES` к выборке `label` из группы `labels`, вводим привязанные данные и присоединяем группу для каждой номинации, смещенную по оси Y с помощью индекса:

```
let catLabels = chartHolder.select('svg').append('g')
      .attr('transform', "translate(10, 10)")
      .attr('class', 'labels')
      .selectAll('label').data(nbviz.CATEGORIES) ❶
      .join('g')
      .attr('transform', function(d, i) {
         return "translate(0," + i * 10 + ")"; ❷
      });
```

❶ Привязываем массив номинаций (`["Chemistry", "Economics", ...]`) к группе `label`, используя метод `data` и следующий за ним метод `join`.

❷ Создаем для каждой номинации группу, и размещаем по вертикали с интервалами 10 пикселей.

Получив выборку `catLabels`, добавим круглый индикатор (похожий на индикаторы на временной диаграмме) и текстовую метку к каждой из групп:

```
catLabels.append('circle')
   .attr('fill', (nbviz.categoryFill)) ❶
   .attr('r', xScale.bandwidth()/2); ❷

catLabels.append('text')
   .text(d => d)
   .attr('dy', '0.4em')
   .attr('x', 10);
```

❶ Используем общий метод `categoryFill`, возвращающий цвет, соответствующий привязанной номинации.

❷ Метод шкалы X `bandwidth` возвращает расстояние между двумя метками номинаций. Радиус круглых маркеров будет равняться половине этого значения.

Функция `categoryFill` (пример 18-3) определена в файле *nbviz_core.js* и используется приложением для предоставления цветов номинаций. В D3 есть ряд цветовых схем в форме массивов hex-кодов цветов, которые можно использовать в качестве цветов заливки SVG. См. демонстрацию цветовых схем на Observable. Мы выбираем набор Category10.

***Пример 18-3.** Установка цветов номинаций*

```
// nbviz_core.js
nbviz.CATEGORIES = [
   "Physiology or Medicine", "Peace", "Physics",
   "Literature", "Chemistry", "Economics"];

nbviz.categoryFill = function(category){
   var i = nbviz.CATEGORIES.indexOf(category);
   return d3.schemeCategory10[i]; ❶
};
```

❶ Цветовая схема D3 `schemeCategory10` — это массив из 10 hex-кодов цветов (`['#1f77b4', '#ff7f0e',...]`), которые мы применим с помощью индекса номинаций премии.

Мы рассмотрели все статические элементы временной диаграммы, давайте придадим ей удобную для использования форму с помощью библиотеки D3 *nest*.

Вложенные данные

Для создания такого компонента временной диаграммы нам придется реорганизовать наш плоский массив объектов лауреатов в структуру, пригодную для привязки к отдельным Нобелевским премиям на временной шкале. Чтобы привязка данных к D3 прошла максимально гладко, нам потребуется массив объектов премии по годам, причем группы лет также доступны как массивы.

Продемонстрируем процесс преобразования на наборе данных о лауреатах Нобелевской премии.

Ниже показан упорядоченный по годам набор данных о лауреатах Нобелевской премии, с которого мы начнем:

```
[
 {"year":1901,"name":"Wilhelm Conrad R\\u00f6ntgen",...},
 {"year":1901,"name":"Jacobus Henricus van \'t Hoff",...},
 {"year":1901,"name":"Sully Prudhomme",...},
 {"year":1901,"name":"Fr\\u00e9d\\u00e9ric Passy",...},
 {"year":1901,"name":"Henry Dunant",...},
 {"year":1901,"name":"Emil Adolf von Behring",...},
 {"year":1902,"name":"Theodor Mommsen",...},
 {"year":1902,"name":"Hermann Emil Fischer",...},
 ...
];
```

Преобразуем эти данные в следующий вложенный формат: массив объектов. Каждый объект состоит их ключа — года и значения — число лауреатов за год:

```
[
 {"key":"1901",
  "values":[
   {"year":1901,"name":"Wilhelm Conrad R\\u00f6ntgen",...},
   {"year":1901,"name":"Jacobus Henricus van \'t Hoff",...},
   {"year":1901,"name":"Sully Prudhomme",...},
   {"year":1901,"name":"Fr\\u00e9d\\u00e9ric Passy",...},
   {"year":1901,"name":"Henry Dunant",...},
   {"year":1901,"name":"Emil Adolf von Behring",...}
  ]
 },
 {"key":"1902",
  "values":[
   {"year":1902,"name":"Theodor Mommsen",...},
   {"year":1902,"name":"Hermann Emil Fischer",...},
   ...
  ]
 },
 ...
];
```

Мы можем проходить в цикле по этому вложенному массиву и привязывать по очереди группы годов, каждая из которых представлена на временной диаграмме столбиком индикаторов.

Чтобы сгруппировать Нобелевские премии по году, мы можем использовать вспомогательный метод D3 `group`. Он принимает массив данных и возвращает объект, сгруппированный по свойству, указанному в коллбэк-функции, в нашем случае — по году вручения премии. Сгруппируем наши записи следующим образом:

```
let nestDataByYear = function (entries) {
   let yearGroups = d3.group(entries, d => d.year) ❶
      // ...
}
```

❶ Используем краткую форму записи — стрелочную функцию JavaScript, эквивалентную записи `function(d) {return d.year}`.

Группирование возвращает массив записей в следующей форме:

```
[// yearGroups
   {1913: [{year: 1913, ...}, {year: 1913, ...}, ...]},
   {1921: [{year: 1921, ...}, {year: 1921, ...}, ...]},
   ...
]
```

Чтобы преобразовать коллекцию в массив объектов ключ-значение, используем JS-объект `Array` и его метод `from`. Мы передаем нашу коллекцию `yearGroups` и функцию-конвертер, которая принимает отдельные группы в виде массива [ключ, значения] и преобразует их в объект {ключ: ключ, значения: значения}. Мы снова применим синтаксис деструктуризации для сопоставления ключа и значений:

```
let keyValues = Array.from(yearGroups,
        [key, values] => {key, values})
```

Теперь у нас есть необходимая функция для группирования отфильтрованных по году записей о лауреатах в нужной форме ключ-значение:

```
nbviz.nestDataByYear = function (entries) {
   let yearGroups = d3.group(entries, (d) => d.year);
```

```
    let keyValues = Array.from(yearGroups, ([key, values]) => {
      let year = key;
      let prizes = values; prizes = prizes.sort(
        (p1, p2) => (p1.category > p2.category? 1: -1)); ❶
      return {key: year, values: prizes};
    });
    return keyValues;
};
```

❶ Используем JavaScript метод массивов `sort`, чтобы отсортировать премии в алфавитном порядке по номинациям, что упростит сравнение года с годом. Метод `sort` ожидает положительное или отрицательное числовое значение, которое мы получаем при логическом сравнении буквенно-цифровых строк.

Добавление лауреатов с помощью вложенных объединений данных

В предыдущей главе, в секции «Сборка столбчатой диаграммы» на стр. 424 мы рассмотрели, как новый метод D3 `join` упрощает синхронизацию изменений данных, тогда это было количество лауреатов по странам, с визуализацией этих данных с помощью столбиков столбчатой диаграммы. Временная диаграмма, показывающая распределение лауреатов по годам, по сути тоже столбчатая, где столбики составлены из круглых маркеров, представляющих собой вручение премии отдельным лауреатам. Теперь рассмотрим, как использовать два объединения данных и вложенный набор данных, созданный в предыдущей секции.

Сначала вложенные данные передаются из `onDataChange` в метод `updateTime Chart` временной диаграммы. Затем мы используем наше первое объединение данных для создания групп годов, позиционируя их с помощью порядковой шкалы X, которая сопоставляет годы с позициями пикселей (см. пример 18-1), и именует их по году:

```
nbviz.updateTimeChart = function (data) {

  let years = svg.selectAll('.year').data(data, d => d.key) ❶

  years
    .join('g') ❷
    .classed('year', true)
```

```
        .attr('name', d => d.key)
        .attr('transform', function (year) {
          return 'translate(' + xScale(+year.key) + ',0)'
        })
        //...
}
```

❶ Привязываем данные года к соответствующему столбцу по ключу года, а не по индексу массива, как было бы по умолчанию, (индекс изменится, если во вложенном массиве будут пропуски по годам, как это часто бывает в наборах данных, выбранных пользователем).

❷ Первое объединение данных использует массив ключ-значения для создания групп столбиков (составленных из кружков) по годам.

Воспользуемся вкладкой Elements в браузере Chrome, чтобы увидеть изменения, которые мы внесли в результате первого объединения данных. На рисунке 18.3 показаны группы годов, вложенные в родительскую для них группу диаграммы.

HTML before

```
▼ <div id="chart-holder" class="_dev">
  ▼ <div id="nobel-time">
    ▼ <svg width="1000" height="150">
      ▼ <g class="chart" transform="translate(40,20)">
        ▶ <g class="x axis" transform="translate(0,100)">…
          </g>
        ▶ <g class="labels" transform="translate(10, 10)">…
          </svg>
```

HTML after

```
  ▼ <div id="nobel-time">
    ▼ <svg width="1000" height="150">
      ▼ <g class="chart" transform="translate(40,20)">
        ▶ <g class="x axis" transform="translate(0,100)">…</g>
          <g class="year" name="1901" transform="translate(18,0)
          <g class="year" name="1902" transform="translate(26,0)
          <g class="year" name="1903" transform="translate(34,0)
          <g class="year" name="1904" transform="translate(42,0)
          <g class="year" name="1905" transform="translate(50,0)
```

Рис. 18.3. *Результат создания групп годов в процессе первого объединения данных*

Давайте проверим, правильно ли привязаны вложенные данные к соответствующим группам годов. На рисунке 18.4 выбираем элемент группы по его году

и изучаем его. В соответствии с требованиями корректные данные были привязаны по году, отображая массив объектов данных о шести лауреатах Нобелевской премии 1901 года.

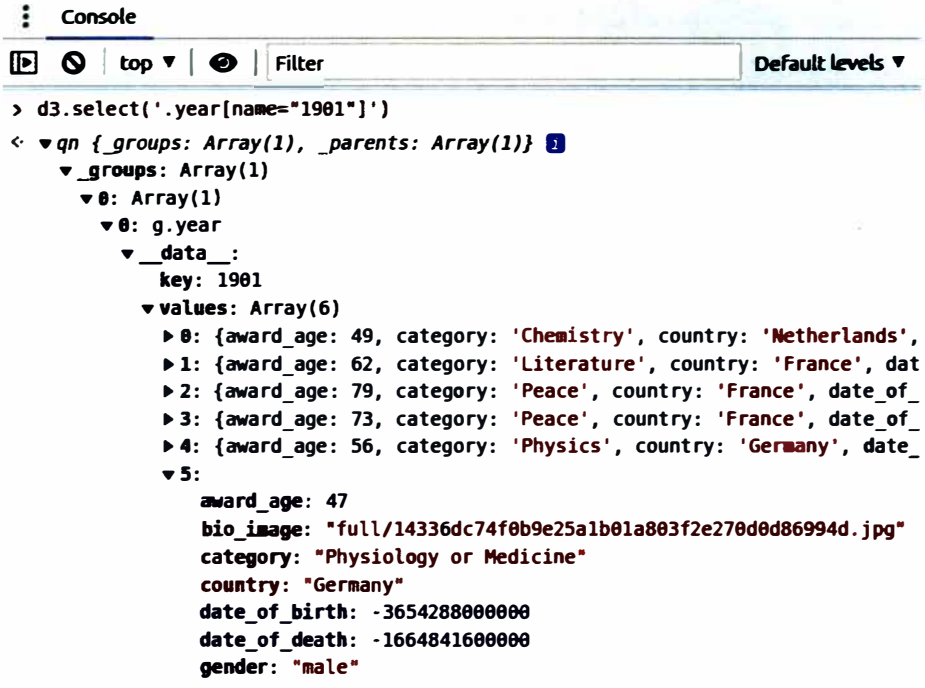


Рис. 18.4. *Проверка результатов первого объединения данных в консоли Chrome*

Привязав данные групп годов к соответствующим им группами, привяжем эти значения к группам круглых маркеров, представляющим каждый год. Первое, что нужно сделать — выбрать все группы годов с только что добавленными данными ключ-значение и привязать массив значений (количество вручений премии за этот год) к плейсхолдерам *winner*, которые вскоре будут привязаны к круглым маркерам. Это выполняет следующий код:

```
let winners = svg
      .selectAll('.year')
      .selectAll('circle') ❶
      .data(
         d => d.values, ❷
         d => d.name ❸
      );
```

❶ Для каждого вручения премии создаем круглый маркер в массиве values.
❷ Используем массив values, чтобы создать круги с помощью привязки данных D3.
❸ Используем опциональный ключ для отслеживания кругов (лауреатов) по имени, как будет показано далее, это пригодится для создания переходных эффектов.

Теперь, когда мы создали наши плейсхолдеры *кругов* с входными данными лауреатов, осталось только объединить их, используя метод D3 `join`. Это позволит отслеживать изменения данных и синхронизировать создание/удаление кругов. Оставшаяся часть на удивление лаконичного кода приведена в примере 18-4.

Пример 18-4. *Второе объединение данных для создания круглых индикаторов премии*

```
winners
   .join((enter) => {
      return enter.append('circle') ❶
            .attr('cy', height)
   })
   .attr('fill', function (d) {
      return nbviz.categoryFill(d.category) ❷
   })
   .attr('cx', xScale.bandwidth() / 2) ❸
   .attr('r', xScale.bandwidth() / 2)
   .attr("cy", (d, i) => yScale(i));
```

❶ Используем пользовательский метод `enter` для добавления новых кругов с позиционированием их по умолчанию внизу диаграммы (в SVG ось Y направлена вниз).
❷ Вспомогательный метод возвращает цвет в зависимости от номинации премии.
❸ Группа года уже правильно позиционирована по оси X, поэтому используем bandwidth столбиков для задания радиуса и центрирования кругов. Для установки высоты столбика по его индексу `i` в массиве лауреатов используем yScale.

Код из примера 18-4 строит временную диаграмму, при необходимости создавая новые круги индикаторов и позиционируя их согласно индексам массива (см. рисунок 18.5).

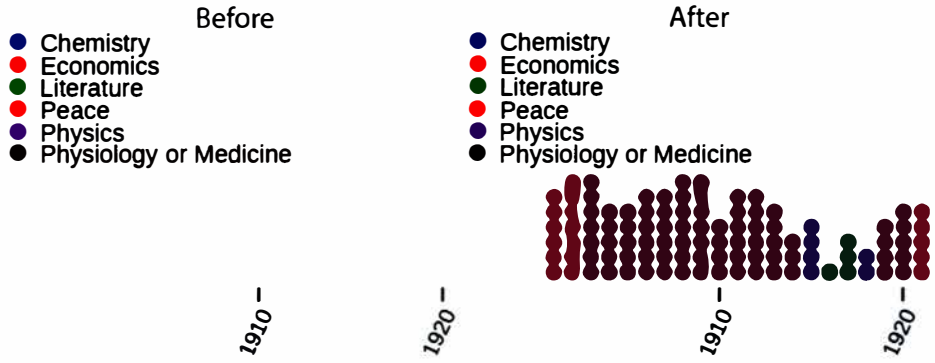


Рис. 18.5. *Успешный результат второго объединения данных*

Хотя мы получили полностью работоспособную временную диаграмму, реагирующую на изменения данных, переходы выглядят резкими[1]. Продемонстрируем мощь D3: добавив две строки кода, получим визуально эффектный переход.

Добавим немного блеска!

Когда пользователь выбирает новый набор данных[2], паттерн обновления, приведенный в примере 18-4, мгновенно задает позиции соответствующих кругов. Давайте сделаем анимацию для плавного перемещения в течение двух секунд.

Фильтрация может оставлять часть существующих индикаторов (например, при выборе из всех номинаций только Chemistry), добавлять новые (например, при смене номинации с Physics на Chemistry) или комбинировать оба варианта. В крайнем случае после фильтрации не останется ничего, например при выборе женщин-лауреатов Нобелевской премии по экономике. Необходимо определить, как должны себя вести существующие индикаторы и как сделать анимацию для новых.

Рисунок 18.6 показывает, что должно произойти при фильтрации нобелевских премий по физике. При выборе пользователем номинации Physics индикаторы других номинаций удаляются методами `exit` и `remove`. Тем временем существующие индикаторы Physics на протяжении двух секунд переходят из текущего положения в конечное, заданное индексом их массива.

---

[1] В подразделе «Переходы» мы выяснили, что визуальные переходы между наборами данных повышают информативность и создают комфортное ощущение непрерывности.

[2] Например, отфильтровав призы по номинациям, можно отобразить только лауреатов по физике.

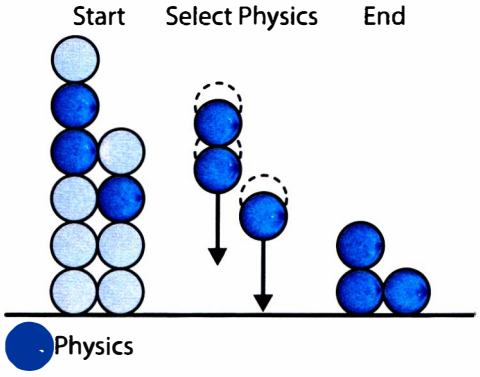


Рис. 18.6. *Переход при выборе подмножества существующих данных*

Удивительно, но оба эффекта — плавное перемещение существующих столбиков на новую позицию и рост новых столбиков снизу — достигаются добавлением всего двух строк кода. Это убедительное доказательство эффективности и зрелости концепции объединения данных в D3.

Для реализации переходов добавляем вызовы методов `transition` и `duration` перед установкой позиции Y круглого маркера (атрибут *cy*). В результате круг будет плавно перемещаться в течение двух секунд (2000 мс):

```
winners
   .join((enter) => {
      return enter.append('circle').attr('cy', height) ❶
   })
   .attr('fill', function (d) {
      return nbviz.categoryFill(d.category)
   })
   .attr('cx', xScale.bandwidth() / 2)
   .attr('r', xScale.bandwidth() / 2)
   .transition() ❷
   .duration(2000)
   .attr("cy", (d, i) => yScale(i));
```

❶ Каждый новый круг стартует от основания диаграммы.

❷ Все круги в течение 2000 мс плавно перемещаются по вертикали в свою позицию.

D3 позволяет с легкостью добавлять интересные визуальные эффекты к переходам данных — еще одно доказательство прочного теоретического фундамента

SKLADCHIK.ORG

библиотеки. Теперь временная диаграмма полностью готова и плавно реагирует на изменения данных.

Обновление столбчатой диаграммы

При импорте модуля временной диаграммы его коллбэк-функция добавляется в массив коллбэков, расположенный в основном модуле. При обновлении данных эта коллбэк-функция обновляет временную диаграмму в соответствии с новыми данными по странам, сгруппированными по году:

```
nbviz.callbacks.push(() => { ❶
   let data = nbviz.nestDataByYear(nbviz.countryDim\
               .top(Infinity))
   updateTimeChart(data)
})
```

❶ При обновлении данных эта анонимная функция вызывается из основного модуля.

Резюме

Начав знакомство с паттерном обновления со столбчатой диаграммы из главы 17, в этой главе мы его расширили, рассмотрев, как использовать второе объединение данных для вложенных данных при создании другого вида диаграмм. Хочу еще раз подчеркнуть, что способность создавать новые визуализации является большим достоинством D3: без привязки к конкретной функциональности традиционной библиотеки диаграмм вы можете добиваться уникальных преобразований своих данных. Как мы увидели на примере столбчатой диаграммы, традиционные динамические диаграммы строить легко, но D3 способна на большее.

Мы также увидели, что надежный паттерн обновления позволяет легко и просто оживить визуализации с помощью привлекательных преобразований.

В следующей главе мы создадим еще один компонент с использованием эффектной топографической библиотеки, предоставленной D3.

Картографирование с помощью D3

Создание и настройка визуализации карт — одна из сильных сторон D3. D3 предоставляет несколько библиотек, позволяющих использовать все виды проекций: от классической проекции Меркатора и ортографической проекции до менее известных, таких как равнопромежуточная коническая проекция. Похоже, что для основных разработчиков D3 Майка Бостока и Джейсона Дэвиса картографирование стало страстью, их внимание к деталям поражает. Если у вас есть проблема с картами, то D3, скорее всего, выполнит всю тяжелую работу по их созданию[1]. В этой главе мы ознакомимся с основными концепциями картографирования D3 на примере карты из визуализации данных о лауреатах Нобелевской премии (Nobel-viz) (рисунок 19.1).

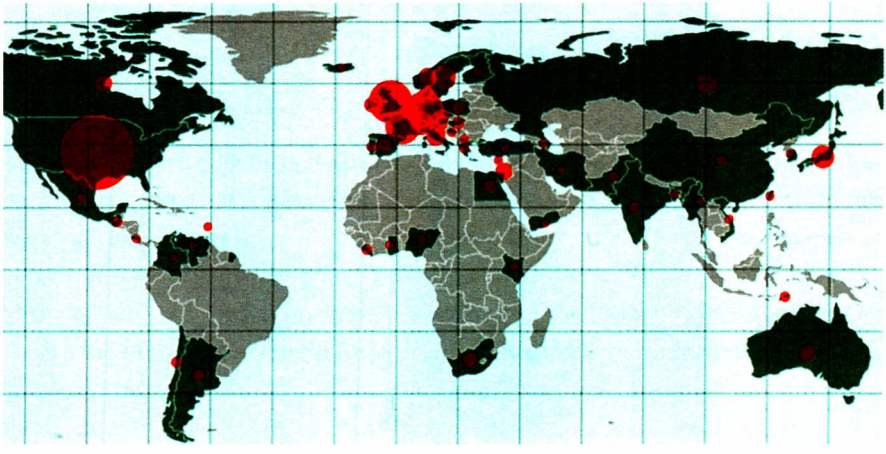


Рис. 19.1. *Целевой элемент этой главы*

Доступные карты

Самый популярный из старых форматов карт — шейп-файл (shapefile), разработанный для приложений геоинформационной системы (ГИС). Существует

[1] Например, математические расчеты геопроекций могут быстро стать сложными.

множество бесплатных и проприетарных компьютерных программ[1], позволяющих создавать и обрабатывать шейп-файлы.

К сожалению, шейп-файлы разрабатывались не для интернета. Для него больше подходят карты в формате JSON и требуются небольшие, эффективные представления для ограничения полосы пропускания и задержки.

Однако есть много удобных способов преобразования шейп-файлов в нужный нам формат TopoJSON[2], так что можно сначала создавать шейп-файлы, а затем преобразовывать их в подходящий веб-формат. Обычно для веб-визуализаций сначала ищут карты в формате TopoJSON или GeoJSON, затем в более обширном пуле шейп-файлов, и в крайнем случае создают собственную карту с помощью редактора шейп-файлов или эквивалентного ему редактора. Как правило удается найти готовое решение. Если нужна карта мира или континента (например, популярная в США проекция Альберса), обычно можно найти ряд решений с различной степенью точности.

Нам потребуется карта мира, где, как минимум, будут показаны все 58 стран, граждане которых становились лауреатами Нобелевской премии, с очертаниями и названиями практически каждой из них. К счастью, D3 предоставляет несколько карт мира, с разрешением сетки 50m и с меньшим разрешением 110m, что удовлетворяет нашим довольно простым требованиям[3].

Форматы данных для картографирования в D3

D3 использует два формата для представления географических данных, основанных на JSON: GeoJSON и его расширение, разработанное Майком Бостоком, — TopoJSON, которое более эффективно кодирует топологию. GeoJSON более читабельный, но в большинстве случаев эффективнее применять TopoJSON. Обычно для веб-доставки, где размер имеет значение, карты преобразуются в формат TopoJSON. Затем TopoJSON с помощью D3 преобразуется в браузере в GeoJSON для упрощения создания SVG-путей (path), оптимизации функций и др.

---

[1] Я пользуюсь опенсорсным десктопным приложением QGIS и настоятельно его рекомендую.

[2] Python-библиотека topojson.py и командная строка TopoJSON.

[3] Как мы увидим далее, на нашей карте отсутствует пара стран, где есть лауреаты Нобелевской премии. Эти страны слишком маленькие, чтобы на них можно было кликнуть, но у нас есть координаты их географических центров, что позволяет разместить на них визуальную подсказку.



Хороший обзор различий между TopoJSON и GeoJSON см. на Stack Overflow.

Рассмотрим эти два формата. Важно понимать их базовую структуру, а небольшие усилия при использовании форматов быстро окупятся, особенно если ваши картографические проекты станут более сложными.

GeoJSON

В файлах GeoJSON значение свойства `type` объекта может быть одним из следующих: Point, MultiPoint, LineString, MultiLineString, Polygon, MultiPolygon, GeometryCollection, Feature или FeatureCollection. Наименование типа объекта записывается в стиле CamelCase, как и показано выше. Объект GeoJSON может иметь необязательное свойство `crs`, определяющее конкретную систему координат CRS (Coordinate Reference System).

FeatureCollection — это крупнейший контейнер GeoJSON, который обычно применяется для карты с несколькими регионами. FeatureCollection содержат массив `features`, каждый элемент которого представляет собой объект GeoJSON одного из типов, перечисленных в предыдущем абзаце.

В примере 19-1 показан обычный FeatureCollection, содержащий массив карт стран, границы которых описывают полигоны (Polygon).

Пример 19-1. *GeoJSON — формат для представления географических данных*

```
{
  "type": "FeatureCollection", ❶
  "features": [ ❷
    {
      "type": "Feature",
      "id": "AFG",
      "properties": {
      "name": "Afghanistan"
    },
    "geometry": { ❸
      "type": "Polygon",
      "coordinates": [
        [
```

```
                [
                    61.210817, ❹
                    35.650072
                ],
                [
                    62.230651,
                    35.270664
                ],
                ...
            ]
        ]
    }
},
...
{
    "type": "Feature",
    "id": "ZWE",
    "properties": {
        "name": "Zimbabwe"
    },
    "geometry": {
        "type": "Polygon",
        "coordinates": [
            [
                [...] ] ]
        }
    }
]
}
```

❶ В каждом файле содержится единственный объект GeoJSON определенного типа, содержащий…

❷ …массив объектов (features), в данном случае объектов стран…

❸ …с основанными на координатах полигонами.

❹ Обратите внимание на порядок координат здесь [долгота, широта], тогда как традиционно в географических координатах сначала указывается широта, а потом долгота. Это связано с тем, что в GeoJSON используется система координат [X, Y].

GeoJSON более лаконичен, чем шейп-файлы, и основан на предпочтительном формате JSON, но все же в кодировке карт присутствует избыточность. Например, общие границы указываются дважды, а формат координат с плавающей точкой довольно негибкий и, во многих случаях, излишне точный. Для решения этих проблем и создания более эффективной доставки карт в браузер и был разработан формат TopoJSON.

TopoJSON

Разработанный Майком Бостоком TopoJSON — это расширение GeoJSON, которое кодирует топологию, объединяя геометрии из общего пула линейных сегментов, называемых дугами. Поскольку дуги используются повторно, объем файлов TopoJSON обычно на 80 % меньше эквивалентных файлов GeoJSON. Кроме того, применение топологического подхода к представлению карты позволяет реализовать ряд методов, которые используют топологию. Например, упрощение формы с сохранением топологии[1], которое позволяет исключить 95 % точек на карте, сохраняя при этом достаточную детализацию. Также этот подход упрощает автоматическую раскраску карт и создание картограмм. В примере 19-2 показана структура файла TopoJSON.

Пример 19-2. *Структура карты мира в формате TopoJSON*

```
{
   "type": "Topology",    ❶
   "objects":{            ❷
   "countries":{
      "type": "GeometryCollection",
      "geometries": [{
      "_id":24, "arcs":[[6,7,8],[10,11,12]], ... ❸
      ...}]},
      "land":{…},
   },
   "arcs":[[[67002,72360],[284,-219],[209..]], /*<- arc*/ номер 0 ❹
          [[70827,73379],[50,-165]], ...   /*<- arc номер 1*/
      ]
   "transform":{ ❺
      "scale":[
         0.003600036...,
```

[1] См. очень классный пример сайте Майка Бостока https://bost.ocks.org/mike/simplify/.

```
            0.001736468...,
        ],
        "translate":[
            -180,
            -90
        ]
    }
}
```

❶ Объект TopoJSON с типом `Topology` должен содержать объект `objects` (именованные объекты геометрии) и массив дуг (`arcs`).

❷ В этом случае эти объекты — `countries` и `land`, оба представляют собой коллекции `GeometryCollection`, определяемые дугами.

❸ Каждая геометрия (в данном случае форма страны) определяется рядом дуговых путей, состоящих из непрерывных дуг, на которые ссылаются их индексы в массиве дуг (`arcs`).

❹ Компоненты массива arcs используются для построения объектов. Дуги выбираются по индексам.

❺ Для квантования координат требуются целочисленные значения, а не числа с плавающей точкой.

При загрузке в браузер меньший размер формата TopoJSON оказывается большим преимуществом. Часто можно найти карты только в формате GeoJSON, поэтому полезно иметь возможность преобразования в TopoJSON. D3 предлагает для такого преобразования утилиту под названием `geo2topo`. Она входит в пакет TopoJSON и устанавливается через `node`.

Преобразование карт в формат TopoJSON

Вы можете установить TopoJSON через репозитории `node` (см. главу 1). Флаг `-g` делает установку глобальной[1]:

```
$ npm install -g topojson
```

После установки `topojson` преобразование GeoJSON в TopoJSON выполняется легче легкого. Вызываем `geo2topo` из командной строки для GeoJSON-файла *geo_input.json*, указывая выходной файл *topo_output.json*:

[1] При глобальной установке вы можете использовать команду `geo2topo` в любом каталоге.

```
$ geo2topo -o topo_output.json geo_input.json
```

Также можно перенаправить результат в файл:

```
$ geo2topo geo_input.json > topo_output.json
```

У `geo2topo` имеется ряд весьма полезных опций, например квантование (квантизация), которое позволяет указать точность карты. Поэкспериментировав с этой опцией, можно получить файл гораздо меньшего размера, незначительно потеряв в качестве. Полную спецификацию см. в справочнике по командной строке geo2topo. Есть удобная Python-библиотека *topojson.py*, которая позволяет преобразовывать файлы карт программно. См. библиотеку на GitHub.

Мы получили картографические данные в легковесном, эффективном, оптимизированном для интернета формате. Теперь рассмотрим, как с помощью JavaScript превратить их в интерактивные веб-карты.

Библиотека D3-geo, проекции и пути

В D3 есть клиентская библиотека *topojson*, предназначенная для данных в формате TopoJSON. Она преобразует основанный на дугах формат TopoJSON в основанный на координатах GeoJSON, готовый к обработке с помощью объектов библиотеки *d3.geo* — `projection` и `paths`.

В примере 19-3 показан процесс извлечения необходимых для нашей карты GeoJSON объектов (features) из TopoJSON-карты *world-100m.json*. Мы получаем полигоны на основе координат, представляющие страны и их границы.

Чтобы извлечь необходимые нам объекты GeoJSON из только что доставленного в браузер объекта TopoJSON `world`, используем методы библиотеки *topojson* — `feature` и `mesh`. Метод `feature` возвращает GeoJSON Feature или FeatureCollection для указанного объекта и объект геометрии GeoJSON MutliLineString, представляющий собой полигональную сетку — «меш» (`mesh`).

Методы `feature` и `mesh` принимают объект TopoJSON как первый аргумент и ссылку на объект, который мы хотим извлечь, как второй (`land` и `countries` в примере 19-3). На нашей карте мира `countries` — это FeatureCollection с массивом объектов (`features`) стран (пример 19-3 ❷).

У метода `mesh` есть третий аргумент, указывающий функцию фильтрации, которая принимает в качестве аргументов два геометрических объекта (`a` и `b`) с общей дугой меша. Если дуга не является общей, значит `a` и `b` совпадают, что позволяет нам отфильтровать внешние границы на карте мира (пример 19-3 ❸).

***Пример 19-3.** Извлечение TopoJSON features*

```
// nbviz_main.mjs
import {initMap} from './nbviz_map.mjs'

Promise.all([
   d3.json('static/data/world-110m.json'), ❶
   d3.csv('static/data/world-country-names-nobel.csv'),
   // ...
]).then(ready)

function ready([worldMap, countryNames, countryData, winnersData]) {
   // ...
   nbviz.initMap(worldMap, countryNames)
}
// nbviz_map.mjs
export let initMap = function(world, names) {
   // ИЗВЛЕЧЕНИЕ НУЖНЫХ ОБЪЕКТОВ ИЗ TOPOJSON
   let land = topojson.feature(world, world.objects.land),
      countries = topojson.feature(world, world.objects.countries)
                  .features, ❷
      borders = topojson.mesh(world, world.objects.countries,
                  function(a, b) {return a !== b;}); ❸
   // ...
}
```

❶ Загружаем данные карты с помощью вспомогательных функций D3 и передаем их в функцию `ready` для инициализации карты.

❷ Используем `topojson` для извлечения нужных объектов из данных TopoJSON и доставляем их в формат GeoJSON.

❸ Отфильтровываем только внутренние, общие для стран, границы. Если дуга используется только одной геометрией (в данном случае, страной), то a и b идентичны.

Представление карты в D3 обычно следует стандартному шаблону. Сначала создаем проекцию D3, используя один из многочисленных методов библиотеки. Затем с помощью этой проекции создадим путь (`path`). И, наконец, используем этот путь для преобразования features и meshes, извлеченных из объекта TopoJSON, в пути SVG, которые отобразятся в окне браузера. Перейдем к изучению обширных возможностей проекций D3.

Проекции

С тех пор, как стало известно, что Земля имеет форму шара, главная трудность картографирования заключается в том, как отобразить трехмерный шар или значительную его часть на двумерной плоскости. В 1569 году фламандский картограф Герард Меркатор решил эту проблему: продлил линии из центра Земли до ее поверхности, затем спроецировал эти точки на цилиндр, оборачивающий земной шар по экватору. Важная для мореплавания особенностью проекции Меркатора — локсодромия/линия румба (*rhumb lines*) на ней изображается отрезками прямой линии. К сожалению, в проекции искажаются расстояния и размеры, чем ближе к полюсам, тем сильнее. В результате огромная Африка кажется ненамного больше Гренландии, хотя на самом деле она примерно в 14 раз больше.

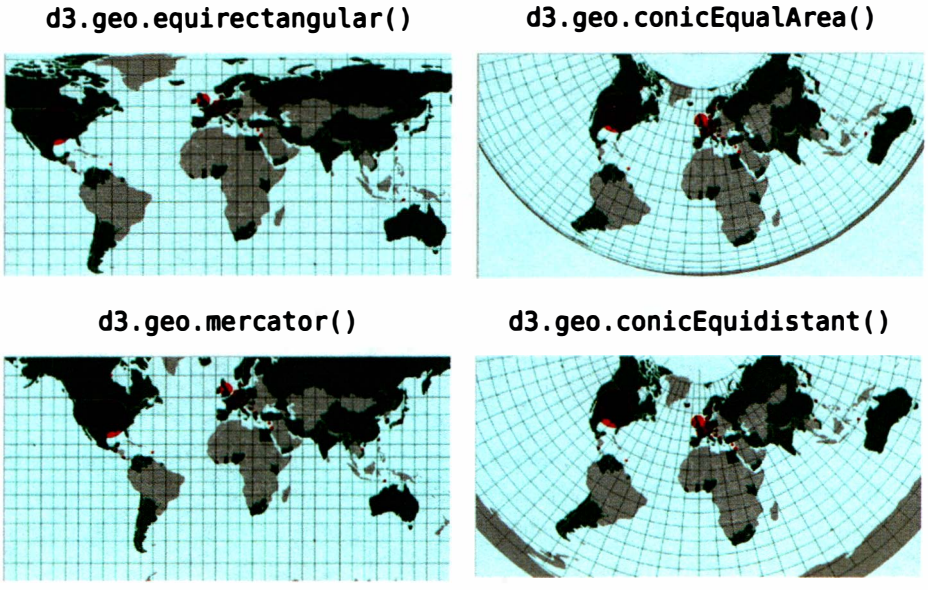


Рис. 19.2. *Несколько проекций карты Нобелевской премии*

Не только проекция Меркатора, но и все остальные являются компромиссным решением. Однако благодаря богатому выбору вариантов D3 всегда можно найти подходящую для конкретной работы проекцию[1]. На рисунке 19.2 показаны некоторые варианты проекции нашей карты, в их числе равнопромежуточная (equirectangular), которую мы выбрали для визуализации. Обязательным условием было показать все страны лауреатов Нобелевской премии в прямоугольном

[1] Расширенный набор D3-проекций — это часть расширения D3, а не основной библиотеки.

SKLADCHIK.ORG

окне и попытаться максимально использовать доступное пространство. Особенно это касается Европы, где много небольших по площади стран с относительно большим числом лауреатов.

Для создания проекции D3 используем один из методов *d3.geo*:

```
let projection = d3.geoEquirectangular()
// ...
```

У D3-проекций есть немало полезных методов. Метод `translate` обычно используется для смещения карты на половину ширины и высоты контейнера, переопределяя значение по умолчанию [480, 250]. Также можно задать точность (precision), которая влияет на степень адаптивной передискретизации (*adaptive resampling*), используемой в проекциях. Адаптивная передискретизация — это умная техника повышения точности проецируемых линий при сохранении эффективности вычислений[1]. Масштаб карты, а также долготу и широту ее центра можно задать с помощью методов `scale` и `center`.

Объединив методы проекций, получаем следующий код, который используется для равнопромежуточной карты Nobel-viz. Примечание: карта была доработана вручную, чтобы увеличить пространство для стран, имеющих нобелевских лауреатов. Были отрезаны оба полюса, поскольку ни в Арктике, ни в Антарктиде нет никаких лауреатов. Обратите внимание, что равнопромежуточные карты предполагают соотношение ширины и высоты, равное 2:

```
let projection = d3.geoEquirectangular()
   .scale(193 * (height/480)) ❶
   .center([15,15]) ❷
   .translate([width / 2, height / 2])
   .precision(.1);
```

❶ Небольшое увеличение: высота по умолчанию 480, масштаб 153.

❷ Координаты центра — 15 градусов восточной долготы, 15 градусов северной широты.

Рассмотрим, как использовать равнопромежуточную проекцию для создания `path`, который, в свою очередь, мы используем для создания карт SVG.

[1] Отличную демонстрацию см. на *https://oreil.ly/oAppn*.

Пути

После выбора подходящей проекции для карты используем ее для создания генератора географического пути `path` в D3 — специализированного варианта генератора путей SVG (`d3.svg.path`). Этот `path` принимает любой объект GeoJSON или геометрический объект, такой как FeatureCollection, Polygon или Point, и возвращает строку данных пути SVG для элемента `d`. Например, с помощью объекта `borders` нашей карты географические координаты границы, которые описывает `MultiLineString`, преобразуются в координаты пути для SVG.

Обычно мы создаем `path` и задаем его `projection` одним махом:

```
var projection = d3.geoEquirectangular()
// …

var path = d3.geoPath()
            .projection(projection);
```

Как правило, `path` используется как генератор атрибута `d` для SVG-пути. Данные GeoJSON привязываются с помощью метода `datum` (эта сокращенная форма записи `data([object])`, служит для привязки единичного объекта, а не массива). Итак, код ниже использует данные о границах, которые мы только что извлекли с помощью `topojson.mesh`, чтобы нарисовать границы страны:

```
// ОБОЗНАЧЕНИЕ ГРАНИЦЫ
svg.insert("path", ".graticule") ❶
   .datum(borders)
   .attr("class", "boundary")
   .attr("d", path);
```

❶ Вставляем границы SVG до наложения картографический сетки (`graticule`) на карту. В результате границы окажутся слоем ниже сетки.

На рисунке 19.3 показана консоль Chrome с выходными данными объекта TopoJSON `borders`, извлеченными из данных нашей карты мира, а также результирующий путь, сгенерированный *d3.geo* `path` с использованием равнопромежуточной проекции.

Генератор географического пути d3.geoPath — главная опора представлений карт D3. Я рекомендую поэкспериментировать с различными проекциями простых геометрий, чтобы прочувствовать, как они работают. Изучите

многочисленные примеры на сайте *bl.ocks.org* и документацию на странице D3 GitHub, а также ознакомьтесь с этой замечательной демонстрацией.

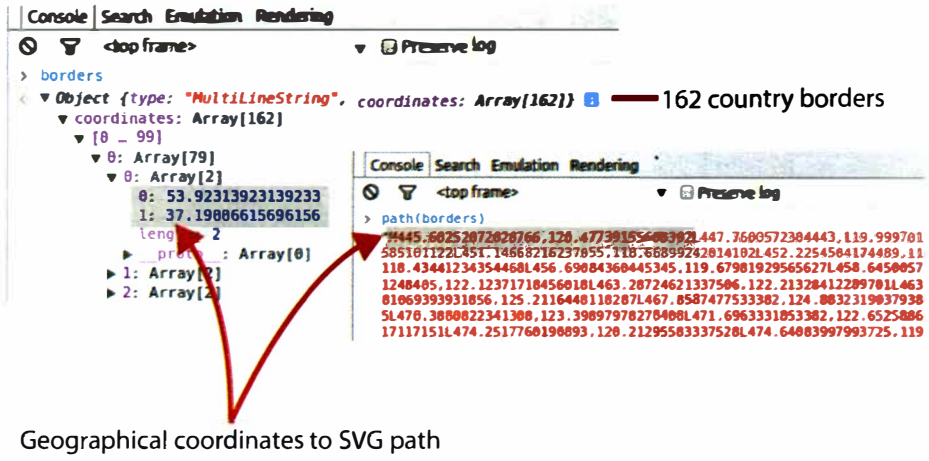


***Рис. 19.3.** Генератор путей, от геометрии до SVG-пути*

Теперь перейдем к одному из компонентов *d3.geo* — `graticule` (картографической/координатной сетке).

Координатные сетки

Координатная сетка `graticule`, которую мы используем на карте, — очень полезный компонент *d3.geo*. Она является одним из генераторов форм географических областей[1], который создает глобальную сетку меридианов и параллелей, расположенных по умолчанию с промежутками в 10 градусов. Если `path` применить к `graticule`, сгенерируется соответствующим образом спроектированная сетка, как показано на рисунке 19.1.

В примере 19-4 показано, как добавить `graticule` к карте. Обратите внимание: если вы хотите, чтобы сетка накладывалась поверх путей карты, то ее путь SVG должен располагаться в дереве DOM после путей карты. Чтобы обеспечить этот порядок, можно также использовать метод D3 `insert`.

***Пример 19-4.** Создание координатной сетки*

```
var graticule = d3.geo.graticule()
                  .step([20, 20]); ❶
```

[1] Полный список см. на D3 GitHub.

```
svg.append("path")
   .datum(graticule) ❷
   .attr("class", "graticule")
   .attr("d", path); ❹
```

❶ Создаем `graticule`, задаем шаг сетки 20 градусов.
❷ Обратите внимание: `datum` — краткая форма записи data([graticule]).
❹ Используем генератор `path` для получения данных `graticule` и возврата пути сетки.

Мы получили сетку, наложенную сверху, и возможность превратить файл карты в SVG-пути с нужной проекцией. Пора собрать все элементы вместе.

Соединение элементов воедино

Создадим карту-основу из компонентов `projection`, `path` и `graticule`. Эта карта будет реагировать на действия пользователя: при выборе лауреатов выделять их страны и отражать число лауреатов с помощью красных кругов в центре стран. Разберемся с этими интерактивными элементами по отдельности.

Код из примера 19-5 построит первичную карту мира. Код следует, надеюсь, уже знакомому вам шаблону: получение `mapContainer` из контейнера `div` (с идентификатором `nobel-map`), добавление к нему тега `<svg>`, затем добавляются SVG-элементы, в данном случае — пути карты, сгенерированные D3.

Фиксированные компоненты карты (например, выбранная проекция и путь) не зависят от изменения данных и определены вне инициализирующего метода `nbviz.initMap`. Этот метод вызывается, когда визуализация инициализируется данными со стороны сервера. Метод принимает TopoJSON-объект `world` и использует его для создания карты-основы с помощью объекта `path`. Результат показан на рисунке 19.4

Пример 19-5. *Построение основы карты*

```
// РАЗМЕРЫ И SVG
let mapContainer = d3.select('#nobel-map');
let boundingRect = mapContainer.node().getBoundingClientRect();
let width = boundingRect.width
   height = boundingRect.height;
let svg = mapContainer.append('svg');
```

```
// ВЫБРАННАЯ НАМИ ПРОЕКЦИЯ
let projection = d3.geo.equirectangular()
   .scale(193 * (height/480))
   .center([15,15])
   .translate([width / 2, height / 2])
   .precision(.1);
// СОЗДАНИЕ ПУТИ С ПОМОЩЬЮ ПРОЕКЦИИ
let path = d3.geoPath().projection(projection);
// ДОБАВЛЕНИЕ КООРДИНАТНОЙ СЕТКИ
var graticule = d3.geoGraticule().step([20, 20]);
svg.append("path").datum(graticule)
   .attr("class", "graticule")
   .attr("d", path);
// МАСШТАБ РАДИУСОВ ДЛЯ ИНДИКАТОРОВ ЦЕНТРОИДОВ
var radiusScale = d3.scaleSqrt()
   .range([nbviz.MIN_CENTROID_RADIUS, nbviz.MAX_CENTROID_RADIUS]);
// ОБЪЕКТ ДЛЯ СОПОСТАВЛЕНИЯ НАЗВАНИЯ СТРАНЫ С ОБЪЕКТОМ GEOJSON
var cnameToCountry = {};
// СОЗДАНИЕ ПЕРВОНАЧАЛЬНОЙ КАРТЫ С ИСПОЛЬЗОВАНИЕМ ЗАГРУЖЕННЫХ
// ДАННЫХ
export let initMap = function(world, names) { ❶
   // ИЗВЛЕЧЕНИЕ НУЖНЫХ ОБЪЕКТОВ ИЗ TOPOJSON
   var land = topojson.feature(world, world.objects.land),
      countries = topojson.feature(world, world.objects.countries)
                  .features,
      borders = topojson.mesh(world, world.objects.countries,
                  function(a, b) {return a!== b;});
      // СОЗДАНИЕ ОБЪЕКТА ДЛЯ СОПОСТАВЛЕНИЯ НАЗВАНИЙ СТРАН
      // С ФОРМАМИ GEOJSON
   var idToCountry = {};
   countries.forEach(function(c) {
      idToCountry[c.id] = c;
   });

   names.forEach(function(n) {
      cnameToCountry[n.name] = idToCountry[n.id]; ❷
   });
   // ОСНОВНАЯ КАРТА МИРА
   svg.insert("path", ".graticule") ❸
```

```
        .datum(land) ❹
        .attr("class", "land")
        .attr("d", path)
    ;
    // ПУТИ СТРАН
    svg.insert("g", ".graticule")
        .attr("class", 'countries');
    // ИНДИКАТОРЫ ЗНАЧЕНИЙ СТРАН
    svg.insert("g")
        .attr("class", "centroids");
    // ЛИНИИ ГРАНИЦ
    svg.insert("path", ".graticule")
        .datum(borders)
        .attr("class", "boundary")
        .attr("d", path);
};
```

❶ TopoJSON-объект `world` с помощью свойства объектов стран «features» с массивом `names` связывает названия стран с идентификаторами свойства объектов стран (например, `{id:36, name: 'Australia'}`).

❷ Если задан ключ названия страны, этот объект возвращает соответствующую ключу геометрию GeoJSON.

❸ Обратите внимание, что мы вставляем этот `path` перед координатной сеткой `graticule`, чтобы сетка накладывалась сверху.

❹ Используем `datum`, чтобы присвоить нашему `path` весь объект `land`.

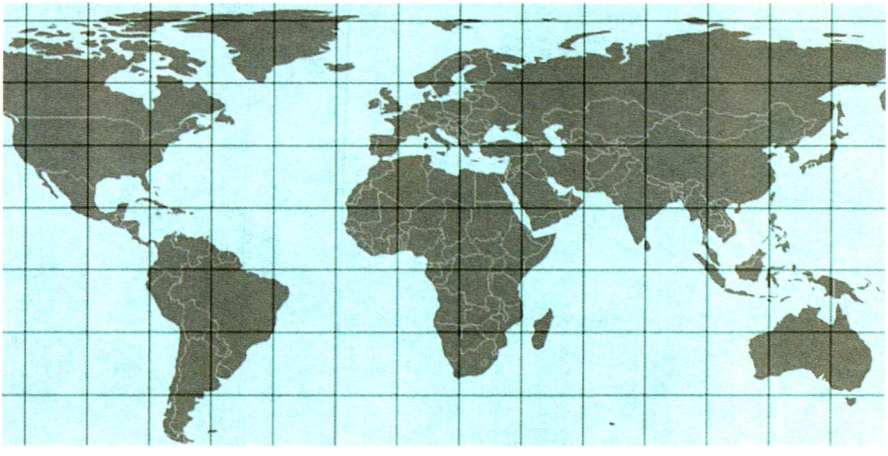


Рис. 19.4. *Карта-основа*

С помощью CSS оформим карту, как на рисунке 19.4 — окрасим океаны светлой лазурью, а сушу светло-серым. Координатная сетка будет темно-серой и полупрозрачной, а границы стран — белыми:

```
/* СТИЛИ NOBEL-MAP */
#nobel-map {
   background: azure;
}

.graticule {
   fill: none;
   stroke: #777;
   stroke-width: .5px;
   stroke-opacity: .5;
}

.land {
   fill: #ddd;
}

.boundary {
   fill: none;
   stroke: #fff;
   stroke-width: .5px;
}
```

Теперь рассмотрим, как на собранной SVG-карте с помощью набора данных о лауреатах Нобелевской премии нарисовать страны, в которых они жили, и как отобразить красный индикатор числа лауреатов для каждой из них.

Обновление карты

Первый раз наша карта обновляется при инициализации визуализации. На этом этапе используется полный набор данных о нобелевских лауреатах. Впоследствии фильтры, которые применит пользователь (например: все лауреаты по химии или лауреаты из Франции), изменят набор данных и карта отобразит это изменение.

Таким образом, передача карте заданного пользователем набора данных о странах с нобелевскими лауреатами, влечет за собой ее обновление. Чтобы сделать это, применим метод `updateMap`:

```
let updateMap = function(countryData) {//...
                }
```

Массив `countryData` выглядит следующим образом:

```
[
   {
      code: "USA",
      key: "United States",
      population: 319259000,
      value: 336 ❶
   },
   // … и еще 56 стран
]
```

❶ Число лауреатов из США в текущем наборе данных.

Перед передачей этого массива в карту D3 преобразуем его с помощью следующего кода, который вернет массив объектов стран со свойствами `geo` (GeoJSON-геометрия для страны), `name` (название страны) и `number` (число лауреатов):

```
let mapData = countryData
   .filter(d => d.value > 0) ❶
   .map(function(d) {
      return {
         geo: cnameToCountry[d.key], ❷
         name: d.key,
         number: d.value
      }
   });
```

❶ Отфильтруем страны без лауреатов — на карте отобразятся только страны с лауреатами.

❷ Используем ключ страны (в данном случае — название), чтобы обновить ее элементарный GeoJSON-объект.

Мы хотим отобразить красный круглый индикатор в центре каждой из стран, где есть лауреаты, указывающий их количество. Площадь кругов должна быть пропорциональна числу лауреатов (абсолютному или на душу населения). Она высчитывается по формуле площадь круга = π × радиус в квадрате, где радиус — это корень квадратный из числа лауреатов. Для этой цели D3 предоставляет удобную функцию `d3.scaleSqrt()` и позволяет задать для нее домен (в данном случае минимальное и максимальное число лауреатов) и диапазон (минимальный и максимальный радиус индикатора).

Рассмотрим пример применения `scaleSqrt`. В следующем коде мы задаем домен от 0 до 100 и диапазон от нуля до 5 (максимальная площадь 5 × 5 = 25). Вызвав функцию со значением 50 (середина домена), мы получим квадратный корень из половины максимальной площади (12.5):

```
var sc = d3.scaleSqrt().domain([0, 100]).range([0, 5]);
sc(50) // возвращает 3.5353…, квадратный корень из 12.5.
```

Чтобы создать шкалу радиусов индикаторов, задаем для функции `scaleSqrt` диапазон, используя максимальные и минимальные радиусы, указанные в файле *nbviz_core.js*:

```
var radiusScale = d3.scaleSqrt()
   .range([nbviz.MIN_CENTROID_RADIUS,
   nbviz.MAX_CENTROID_RADIUS]);
```

Максимальное значение домена получаем с помощью `mapData`, оно будет равно максимальному числу лауреатов на страну, а минимальное значение 0:

```
var maxWinners = d3.max(mapData.map(d => d.number))
// ДОМЕН ШКАЛЫ ЗНАЧЕНИЯ ИНДИКАТОРОВ
radiusScale.domain([0, maxWinners]);
```

Чтобы добавить контуры стран к существующей карте, мы сначала привязываем `mapData` к выборке из группы `countries` класса `country` и применяем паттерн обновления (см. «Обновление DOM при изменении данных» на стр. 516), добавляя очертания любых стран, которые требуются для `mapData`. Вместо удаления непривязанных контуров используем CSS-свойство `opacity`, чтобы привязанные страны стали видимыми, а остальные — невидимыми. Чтобы исчезновение и появление стран было плавным, применяем двухсекундный переход. В примере 19-6 показан паттерн обновления.

Пример 19-6. *Обновление очертаний стран*

```
let countries = svg
   .select('.countries').selectAll('.country')
   .data(mapData, d => d.name)
// Используем объединение данных, чтобы сделать выбранные страны
// видимыми и плавно проявлять их в течение TRANS_DURATION
// миллисекунд
countries
   .join(
      (enter) => {
         return enter
            .append('path') ❶
               .attr('d', function (d) {
                  return path(d.geo)
            })
            .attr('class', 'country')
            .attr('name', d => d.name)
            .on('mouseenter', function (event, d) { ❷
               d3.select(this).classed('active', true)
            })
            .on('mouseout', function (d) {
               d3.select(this).classed('active', false)
            })
   },
   (update) => update,
   (exit) => { ❸
      return exit
         .classed('visible', false)
         .transition()
         .duration(nbviz.TRANS_DURATION)
         .style('opacity', 0)
   }
)
.classed('visible', true)
.transition() ❹
.duration(nbviz.TRANS_DURATION)
.style('opacity', 1)
```

SKLADCHIK.ORG

❶ Используем данные GeoJSON, чтобы создать очертания страны на карте с помощью объекта `path`.

❷ Плейсхолдеры UI, которые задают SVG-путям класс *active* при событии mouseover. Обратите внимание, что здесь мы используем ключевое слово `function` вместо обычной стрелочной нотации (⇒). Это связано с тем, что мы используем D3 для доступа к DOM-элементу (области карты) через ключевое слово `this`, которое недоступно в стрелочных функциях.

❸ Настраиваемая функция `exit`, которая делает очертания страны совершенно прозрачными (opacity 0) в течение 2000 миллисекунд (`TRANS_DURATION`).

❹ Любые новые страны становятся видимыми (opacity 1) в течение 2000 миллисекунд (`TRANS_DURATION`).

Обратите внимание, что только что добавленным странам присваивается CSS-класс `country`, устанавливающий светло-зеленую заливку. При наведении на страну курсора мыши используется класс `country.active` и страна окрашивается в темно-зеленый цвет. Классы CSS:

```
.country{
   fill: rgb(175, 195, 186); /* светло-зеленый */
}

.country.active{
   fill: rgb(155, 175, 166); /* темно-зеленый */
}
```

Показанный в примере 19-6 паттерн обновления позволяет плавно переходить от старых наборов данных к новым, созданным в ответ на фильтры, которые применил пользователь, и переданным в `updateMap`. Нам осталось только добавить респонсивные круглые индикаторы, помещенные в центрах активных стран и отражающие текущее значение: либо абсолютный, либо относительный (на душу населения) показатель нобелевских лауреатов.

Добавление индикаторов показателей

Чтобы добавить круглые индикаторы показателей, потребуется паттерн обновления, зеркалирующий тот, который мы использовали при создании SVG-путей для страны. Нам нужно сделать привязку к набору данных `mapData` и добавлять, обновлять или удалять круги-индикаторы. Как и в случае с очертаниями

стран, мы будем менять прозрачность индикаторов, чтобы они появлялись и исчезали на карте.

Индикаторы необходимо разместить в центрах соответствующих стран. D3-генератор путей `path` предоставляет ряд ценных вспомогательных методов для работы с геометриями GeoJSON. Один из этих методов, `centroid`, вычисляет проекцию центроида для указанного элементарного объекта:

```
// Для GeoJSON страны (country.geo)
// вычисляем координаты (x, y) его центра
var center = path.centroid(country.geo);
// center = [x, y]
```

Метод `path.centroid` иногда возвращает странные результаты для сильно вогнутых геометрий, хотя, как правило, работает довольно хорошо и очень полезен для маркировки очертаний, границ и др. К счастью, данные о странах мира, которые мы сохранили (см. «Получение данных о странах для визуализации нобелевских лауреатов» из главы 5), содержат координаты центров всех стран, где есть нобелевские лауреаты.

Сначала напишем функцию для извлечения данных из объекта `mapData`:

```
var getCentroid = function(d) {
   var latlng = nbviz.data.countryData[d.name].latlng; ❶
   return projection([latlng[1], latlng[0]]); ❷
};
```

❶ Получаем широту и долготу центра страны по ее наименованию, используя сохраненные данные о странах мира.

❷ Преобразуем полученную широту и долготу в координаты SVG, используя равнопромежуточную проекцию.

В примере 19-7 мы привязываем `mapData` к выборке всех элементов с классом `centroid` из группы `centroids`, добавленной в коде из примера 19-5. Эти данные привязаны через ключ `name`.

Пример 19-7. *Добавление индикаторов в центроидах стран, где есть нобелевские лауреаты*

```
let updateMap = function(countryData) {
//...
```

```
    // ПРИВЯЗКА ДАННЫХ КАРТЫ ПО КЛЮЧУ NAME
    let centroids = svg
       .select('.centroids').selectAll('.centroid')
       .data(mapData, d => d.name) ❶
    // ПРИВЯЗКА ДАННЫХ К КРУГАМ-ИНДИКАТОРАМ
    centroids
       .join(
          (enter) => {
             return enter
                .append("circle")
                .attr("class", "centroid")
                .attr("name", (d) => d.name)
                .attr("cx", (d) => getCentroid(d)[0]) ❷
                .attr("cy", (d) => getCentroid(d)[1])
          },
          (update) => update,
          (exit) => exit.style("opacity", 0)
       )
       .classed("active",
          (d) => d.name === nbviz.activeCountry)
       .transition()
       .duration(nbviz.TRANS_DURATION) ❸
       .style("opacity", 1)
       .attr("r", (d) => radiusScale(+d.number))
};
```

❶ Привязываем данные карты к центроидам элементов по ключу `name`.

❷ Используем функцию `getCentroid`, которая возвращает позицию в пикселях для географического центра стран.

❸ За 2000 мс круглый маркер плавно наращивает яркость и параллельно переходит к новому радиусу.

С помощью CSS мы сделаем индикаторы красными и слегка прозрачными, чтобы можно было видеть сквозь них детали карты и другие индикаторы, на которые они могут накладываться (особенно это касается Европы). Если пользователь выбирает страну с помощью фильтра стран на панели UI, она классифицируется как `active`, и ее индикатор меняет цвет на золотистый. Код CSS, который делает это:

```
.centroid{
   fill: red;
   fill-opacity: 0.3;
   pointer-events: none; ❶
}

   .centroid.active {fill: goldenrod;
   fill-opacity: 0.6;
}
```

❶ Это свойство дает возможность событиям мыши распространяться на очертания стран, перекрытых кругами-индикаторами, что позволяет пользователю кликать по этим странам.

Итак, мы добавили последний элемент на нашу карту Нобелевских премий — индикаторы активных центроидов стран. Давайте окинем взглядом полную картину.

Готовая карта

В зависимости от фильтров, которые применяет пользователь, карта плавно переходит в нужное состояние благодаря паттернам обновления и индикаторам стран. На рисунке 19.5 показан результат отбора лауреатов Нобелевской премии в области экономики. Зеленым выделены только страны с лауреатами в области экономики. Размеры индикаторов обновлены, отражая доминирование США в этой номинации.

Карта сейчас не интерактивна, но реагирует на наведение курсора мыши на определенную страну, вызывая коллбэк-функции `mouseenter` и `mouseout` и соответственно добавляя или удаляя класс `active`. Эти коллбэки можно использовать для добавления карте дополнительной функциональности, например всплывающих подсказок или использования стран в качестве кликабельных фильтров данных. Давайте создадим простую всплывающую подсказку, которая при наведении курсора мыши покажет название страны и краткую информацию о премии.

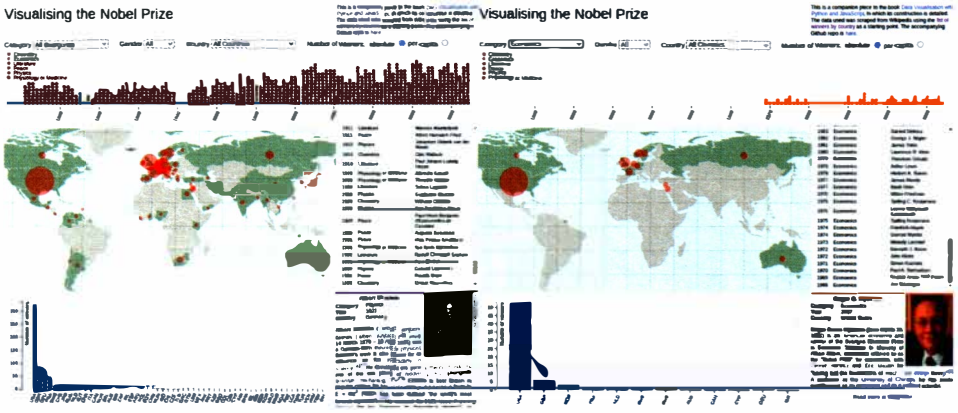


Рис. 19.5. *Слева показан полный набор данных, справа — результат фильтрации по номинации «Экономика», показывающий страны с лауреатами (а также доминирование экономистов из США)*

Создание простой всплывающей подсказки

Всплывающие подсказки и другие интерактивные виджеты, как правило, востребованы при веб-визуализации данных. Иногда они бывают довольно сложными, особенно если сами по себе интерактивны (например, меню, появляющиеся при наведении мыши). В то же время существует несколько простых рецептов, которые полезно знать. В этом подразделе мы рассмотрим, как создать простую, но эффективную всплывающую подсказку. На рисунке 19.6 показано, что мы хотим сделать.



Рис. 19.6. *Простая всплывающая подсказка для карты Нобелевской премии*

Припомним обновление текущих `countries`, где обработчики событий `mouseenter` и `mouseout` добавляются во время объединения данных:

```
    // ВВОД И ДОБАВЛЕНИЕ НОВЫХ СТРАН
    countries.join((enter) => {
       return enter.append('path')
    // …
       .on('mouseenter', function(d) {
          d3.select(this).classed('active', true);
       })
       .on('mouseout', function(d) {
          d3.select(this).classed('active', false);
       })
    })
;
```

Для добавления всплывающей подсказки к карте нужно сделать три вещи:

1. Создать HTML-блок подсказки с плейсхолдерами для информации, которую мы хотим отразить, в данном случае — это название страны и число лауреатов в выбранной номинации.
2. Отобразить блок HTML, когда пользователь наведет курсор мыши на страну, и спрятать, когда курсор окажется за ее пределами.
3. Обновить блок, отобразив в нем данные, привязанные к стране, на которую наведен курсор мыши.

Создаем HTML для всплывающей подсказки: добавим блок контента для участка карты Nobel-viz с идентификатором `map-tooltip`, заголовком `<h2>` для названия страны и тегом `<p>` для текста подсказки:

```
<!– index.html –>
   <!– ... –>
      <div id="nobel-map">
         <div id="map-tooltip">
            <h2></h2>
            <p></p>
         </div>
   <!– ... –>
```

Нам также понадобится CSS для оформления внешнего вида и поведения подсказки, добавленный в наш файл *style.css:*

```
/* css/style.css */
/* ПОДСКАЗКА ДЛЯ КАРТЫ */
#map-tooltip {
   position: absolute;
   pointer-events: none; ❶
   color: #eee;
   font-size: 12px;
   opacity: 0.7; /* слегка прозрачная */
   background: #222;
   border: 2px solid #555;
   border-color: goldenrod;
   padding: 10px;
   left: -999px; ❷
}

#map-tooltip h2 {
   text-align: center;
   padding: 0px;
   margin: 0px;
}
```

❶ Установим для `pointer-events` значение `none`, чтобы взаимодействовать с элементами, которые перекрывает подсказка.

❷ Изначально подсказка скрыта далеко за левой границей окна браузера с помощью отрицательной координаты по оси X.

Имея HTML-код для подсказки и спрятанный слева от окна браузера элемент (`left` –999 пикселей), нам просто нужно расширить коллбэк-функции `mousein` и `mouseout`, чтобы отображать или скрывать подсказку. Основную часть работы делает функция `mousein`, которая вызывается, когда пользователь перемещает курсор мыши внутрь страны:

```
// …
countries.join(
   (enter) => {
   .append('path')
   .attr('class', 'country')
   .on('mouseenter', function(event) {
```

```
            var country = d3.select(this);
            // если страна невидима, ничего не происходит
            if(!country.classed('visible')){return;}

            // получаем объект данных страны
            var cData = country.datum();
            // если единственный лауреат, используется 'prize' в ед. ч.
            var prize_string = (cData.number === 1)?
               ' prize in ': ' prizes in ';
            // задаем заголовок и текст подсказки
            tooltip.select('h2').text(cData.name);
            tooltip.select('p').text(cData.number
               + prize_string + nbviz.activeCategory);
            // задаем цвет рамки подсказки в соответствии с выбранной
            // номинацией премии
            var borderColor =
               (nbviz.activeCategory === nbviz.ALL_CATS)?
               'goldenrod':
               nbviz.categoryFill(nbviz.activeCategory);
            tooltip.style('border-color', borderColor);

            var mouseCoords = d3.pointer(event); ❶
            var w = parseInt(tooltip.style('width')), ❷
                h = parseInt(tooltip.style('height'));
            tooltip.style('top', (mouseCoords[1] - h) + 'px'); ❸
            tooltip.style('left', (mouseCoords[0] - w/2) + 'px');

            d3.select(this).classed('active', true);
         })
         .on('mouseout', function (d) {
            tooltip.style('left', '-9999px') ❹
            d3.select(this).classed('active', false)
         })
      }, // ...
)
```

❶ Метод `pointer` из D3 возвращает координаты курсора мыши в пикселях из объекта `event` (здесь относительно родительской группы карты), которые используются при позиционировании всплывающей подсказки.

❷ Получаем подсчитанную ширину и высоту области подсказки, настроенные с учетом названия страны и строки информации о премии.

❸ Используем координаты мыши, а также ширину и высоту области подсказки, чтобы выровнять текст по центру горизонтали и поместить его примерно над курсором мыши (в ширину и высоту не входят 10 пикселей внутреннего отступа вокруг блока подсказки).

❹ Когда курсор мыши выходит за пределы страны, скрываем подсказку, помещая ее далеко слева от карты.

После того, как написана коллбэк-функция `mouseenter`, нам требуется только `mouseout`, чтобы прятать подсказку далеко слева от области видимости:

```
countries.join(
      (enter) => {
      .append('path')
      .attr('class', 'country')
      .on('mouseenter', function(event) {
      // ...
      })
      .on('mouseout', function (d) {
         tooltip.style('left', '-9999px') ❶
         d3.select(this).classed('active', false)
      })
   }, // ...
)
```

❶ Как только курсор мыши покидает страну, прячем подсказку далеко слева, за пределами видимой пользователю веб-страницы, и удаляем для страны класс `'active'`, возвращая ее цвет по умолчанию.

При совместном использовании функций `mouseenter` и `mouseout` всплывающая подсказка появляется и исчезает там, где необходимо, как показано на рисунке 19.6.

Обновление карты

При импорте модуля карты коллбэк-функция добавляется в конец массива коллбэков в основном модуле. При обновлении данных в ответ на действия

пользователя эта коллбэк-функция вызывается, и столбчатая диаграмма обновляется в соответствии с новыми данными по странам:

```
nbviz.callbacks.push(() => { ❶
   let data = nbviz.getCountryData()
   updateMap(data)
})
```

❶ При обновлении данных эта анонимная функция вызывается из основного модуля.

Мы создали карту — компонент для визуализации данных о нобелевских лауреатах и теперь подведем итоги изученного. Затем перейдем к демонстрации влияния на визуализацию действий пользователя.

Резюме

Картографирование с помощью D3 — обширная область. D3 предоставляет множество различных проекций и вспомогательных методов, облегчающих манипулирование геометриями. Но построение карты следует довольно стандартной схеме, о чем говорилось в этой главе. Сначала вам нужно выбрать проекцию, скажем, Меркатора или коническую проекцию Альберса, которую предпочитают использовать в США. Затем, используя эту проекцию, вы создаете генератор D3-путей, который преобразует объекты GeoJSON в пути SVG и построит карту, которую вы видите. GeoJSON обычно получают из более эффективных данных TopoJSON.

В этой главе также было показано, насколько просто с помощью D3 интерактивно подсвечивать карту и реагировать на перемещение курсора. Используя совокупность полученных навыков, вы уже можете приступать к созданию картографических визуализаций.

Итак, мы создали все графические элементы на основе SVG. Теперь давайте посмотрим, насколько хорошо D3 работает с обычными элементами HTML, создав список лауреатов и область для биографии отдельного лауреата.

SKLADCHIK.ORG

Визуализация данных отдельных лауреатов

Мы хотим, чтобы визуализация данных о лауреатах Нобелевской премии (Nobel-viz) включала список выбранных в данный момент лауреатов и биографический блок, в которой будут отображаться сведения об отдельном лауреате (см. рисунок 20.1). Кликнув по фамилии лауреата в списке, пользователь увидит информацию о нем в биографическом блоке. Из этой главы вы узнаете, как создать список лауреатов и биографический блок, как заполнить список новыми данными, которые выбрал пользователь с помощью фильтров в строке меню, и как сделать список кликабельным.

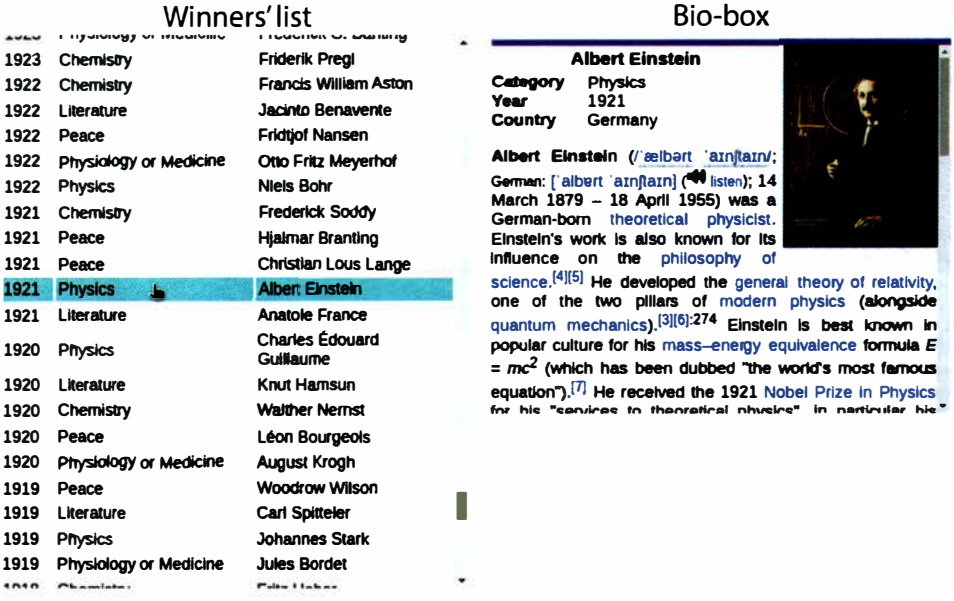


Рис. 20.1. *Элементы, описанные в главе*

Как вы узнаете из этой главы, D3 справляется не только с созданием SVG-визуализаций. Вы можете привязать данные к любому элементу DOM и изменять с их помощью его атрибуты, свойства и коллбэк-функции обработки событий.

Привязка данных и обработка событий в D3 (с помощью метода `on`) хорошо работают со стандартными пользовательскими интерфейсами, такими как интерактивный список, описанный в этой главе, и выпадающий список[1].

Сначала займемся списком лауреатов и выясним, как он формируется в зависимости от выбранного набора данных.

Создание списка лауреатов

Мы строим список лауреатов (см. рисунок 20.1) на основе HTML-таблицы со столбцами Year, Category и Name. Базовый каркас списка находится в файле *index.html* нашего проекта Nobel-viz:

```
<! DOCTYPE html>
<meta charset="utf-8">
<body>
...
   <div id="nobel-list">
      <h2>Selected winners</h2>
      <table>
         <thead>
            <tr>
               <th id='year'>Year</th>
               <th id='category'>Category</th>
               <th id='name'>Name</th>
            </tr>
         </thead>
         <tbody>
         </tbody>
      </table>
   </div>
...
</body>
```

Чтобы настроить ширину столбцов и размер шрифта таблицы, мы берем стили CSS из файла *style.css*:

[1] Выпадающими списками мы займемся в главе 21.

```
/* СПИСОК ЛАУРЕАТОВ */
#nobel-list {overflow: scroll; overflow-x: hidden;} ❶

#nobel-list table{font-size: 10px;}
#nobel-list table th#year {width: 30px}
#nobel-list table th#category {width: 120px}
#nobel-list table th#name {width: 120px}
#nobel-list h2 {font-size: 14px; margin: 4px;
text-align: center}
```

❶ Свойство `overflow: scroll` ограничивает снизу содержимое списка (в рамках контейнера `nobel-list`) и добавляет вертикальную полосу прокрутки, предоставляя доступ ко всем лауреатам. Свойство `overflow-x: hidden` отключает горизонтальную прокрутку.

Чтобы создать список, внутрь элемента таблицы `<tbody>` вставим элементы строк `<tr>` (в них содержатся ячейки данных `<td>`, по одной на столбец) для каждого лауреата текущего набора данных:

```
...
<tbody>
   <tr>
      <td>2014</td>
      <td>Chemistry</td>
      <td>Eric Betzig</td>
   </tr>
   ...
</tbody>
...
```

При инициализации приложения и после каждого изменения фильтров (см. «Основной поток данных» из главы 16) центральный `onData Change` вызывает метод `updateList`. Данные, полученные `updateList` имеют следующую структуру:

```
// data =
[{
   name:"C\u00e9sar Milstein",
   category:"Physiology or Medicine",
```

```
    gender:"male",
    country:"Argentina",
    year: 1984
    _id: "5693be6c26a7113f2cc0b3f4"
  },
  ...
]
```

Метод `updateList` показан в примере 20-1. Сначала полученные данные сортируются по годам, а затем — после удаления существующих строк, — используются для построения строк таблицы.

Пример 20-1. *Создание списка выбранных лауреатов*

```
let updateList = function (data) {
   let tableBody, rows, cells
   // Сортировка данных по годам
   data = data.sort(function (a, b) {
      return +b.year - +a.year
   })
   // выбираем table-body из index.html
   tableBody = d3.select('#nobel-list tbody')
   // создаем плейсхолдер для строк, привязанных к данным лауреатов
   rows = tableBody.selectAll('tr').data(data)

   rows.join( ❶
      (enter) => {
         // создаем нужные строки
         return enter.append('tr').on('click', function (event, d) {❷
            console.log('You clicked a row ' + JSON.stringify(d))
            displayWinner(d)
         })
      },
      (update) => update,
      (exit) => {
         return exit ❸
            .transition()
            .duration(nbviz.TRANS_DURATION)
            .style('opacity', 0)
            .remove()
```

```
        }
    )

    cells = tableBody ❹
        .selectAll('tr')
        .selectAll('td')
        .data(function (d) {
            return [d.year, d.category, d.name]
        })
    // Добавляем ячейки данных и вставляем в них текст
    cells.join('td').text(d => d) ❺
    // Если данные доступны, на экран выводится случайно выбранный
    // лауреат
    if (data.length) { ❻
        displayWinner(data[Math.floor(Math.random() * data.length)])
    }
}
```

❶ Уже знакомый нам паттерн объединения использует привязанные данные лауреатов для создания и обновления элементов списка.

❷ Когда пользователь кликает по строке, эта функция-обработчик передает данные лауреата в метод `displayWinner`, который обновляет блок с биографией.

❸ Кастомная функция `exit` плавно скрывает лишние строки в течение двух секунд, делая их все более прозрачными (сводит opacity к 0), а затем удаляет.

❹ Сначала на основе данных лауреата создаем массив с годом, номинацией и именем для создания в строке ячеек `<td>`,

❺ …а затем привязываем массив к ячейкам (`td`) строки и вставляем в них текст.

❻ При каждом изменении данных мы случайным образом выбираем лауреата из нового набора данных и отображаем в блоке с биографией.

При наведении курсора на строку таблицы лауреатов должна подсвечиваться строка и меняться стиль курсора, указывая на кликабельность элемента. Такое поведение обеспечивает следующий код CSS, который добавляется из файла *style.css*:

```
#nobel-list tr: hover{
    cursor: pointer;
    background: lightblue;
}
```

Метод `updateList` вызывает метод `displayWinner`, возвращающий область биографии лауреата по клику на строке или при изменении данных случайным образом. Теперь рассмотрим, как создается биографический блок.

Создание биографического блока

Биографический блок заполняется данными из объекта лауреата (nobel-winner), содержащего краткую информацию. HTML-структура для биографического блока, содержащаяся в файле *index.html* состоит из блоков контента для элементов биографии и футера `readmore` со ссылкой на статью о лауреате в Википедии.

```
<! DOCTYPE html>
<meta charset="utf-8">
<body>
...
      <div id="nobel-winner">
         <div id="picbox"></div>
         <div id='winner-title'></div>
         <div id='infobox'>
            <div class='property'>
               <div class='label'>Category</div>
               <span name='category'></span>
            </div>
            <div class='property'>
               <div class='label'>Year</div>
               <span name='year'></span>
            </div>
            <div class='property'>
               <div class='label'>Country</div>
               <span name='country'></span>
            </div>
         </div>
         <div id='biobox'></div>
         <div id='readmore'>
            <a href='#'>Read more at Wikipedia</a></div>
      </div>
...
</body>
```

CSS-код из файла *style.css* задает позиционирование списка и элементов биографического блока, размер блоков контента, а также стиль рамок и шрифтов:

```
/* БЛОК С ИНФОРМАЦИЕЙ О ЛАУРЕАТЕ */

#nobel-winner {
   font-size: 11px;
   overflow: auto;
   overflow-x: hidden;
   border-top: 4px solid;
}

#nobel-winner #winner-title {
   font-size: 12px;
   text-align: center;
   padding: 2px;
   font-weight: bold;
}

#nobel-winner #infobox .label {
   display: inline-block;
   width: 60px;
   font-weight: bold;
}

#nobel-winner #biobox {font-size: 11px;}
#nobel-winner #biobox p  {text-align: justify;}

#nobel-winner #picbox {
   float: right;
   margin-left: 5px;
}
#nobel-winner #picbox img {width:100px;}

#nobel-winner #readmore {
   font-weight: bold;
   text-align: center;
}
```

Итак, блоки контента готовы, теперь нам нужен коллбэк к API, чтобы получить данные для заполнения блоков. В примере 20-2 показан метод `displayWinner` для создания биографического блока (bio-box).

***Пример 20-2.** Обновление биографического блока данными о выбранном лауреате*

```
let displayWinner = function (wData) {
   // сохраняем элемент биографического блока лауреата
   let nw = d3.select('#nobel-winner')

   nw.select('#winner-title').text(wData.name)
   nw.style('border-color', nbviz.categoryFill(wData.category)) ❶

   nw.selectAll('.property span').text(function (d) { ❷
      var property = d3.select(this).attr('name')
      return wData[property]
   })

   nw.select('#biobox').html(wData.mini_bio)
   // Добавляем новое изображение, если оно есть, иначе удаляем
   // старое
   if (wData.bio_image) { ❸
      nw.select('#picbox img')
         .attr('src', 'static/images/winners/' + wData.bio_image)
         .style('display', 'inline')
   } else {
      nw.select('#picbox img').style('display', 'none')
   }

   nw.select('#readmore a').attr( ❹
      'href',
      'http://en.wikipedia.org/wiki/' + wData.name
   )
}
```

❶ Верхнюю границу ((CSS: `border-top: 4px solid`)) элемента `nobel-winner` окрашиваем в цвет номинации премии с помощью метода `categoryFill`, определенного в файле *nbviz_core.js*.

❷ Выбираем теги всех блоков div с классом property. Они имеют форму . Используем атрибут name, чтобы извлечь нужное свойство из данных выбранного нобелевского лауреата и использовать его для размещения текста тега span.

❸ Задаем атрибут src — источник фотографии лауреата, если она есть. Мы используем атрибут display тега изображения, чтобы скрыть его (установив none), если изображение недоступно, или показать, если доступно (значение по умолчанию inline.

❹ Имя лауреата мы получили из Википедии и можем использовать для перехода на его страницу.

Обновление списка лауреатов

При импорте модуля со списком лауреатов и областью биографии его коллбэк-функция добавляется в конец массива коллбэков в основном модуле. Когда при взаимодействии с пользователем обновляются данные, эта коллбэк-функция вызывается и обновляет список новыми данными о стране, используя измерение стран из библиотеки Crossfilter:

```
nbviz.callbacks.push(() => { ❶
   let data = nbviz.countryDim.top(Infinity)
   updateList(data)
})
```

❶ При обновлении данных эта анонимная функция вызывается из основного модуля.

Итак, мы узнали, как добавить немного индивидуальности нашему Nobel-viz, предоставив пользователю возможность отобразить биографию лауреата. Теперь подведем итоги этой главы, прежде чем перейти к знакомству со строкой меню.

Резюме

В этой главе мы рассмотрели, как использовать D3 для создания традиционных HTML-конструкций. Библиотека D3 так же хороша для построения списков, таблиц и др., как и для отображения кругов или вращения линии. Как правило, с помощью D3 можно эффективно и изящно решать любые задачи по отображению изменения данных с помощью элементов веб-страницы.

Получив список лауреатов и область биографии, мы завершили знакомство с визуальными элементами Nobel-viz. Остается только выяснить, как построена строка меню, и как визуальные элементы отражают изменения, которые она позволяет применить к набору данных и к показателю числа лауреатов.

SKLADCHIK.ORG

Строка меню

В предыдущих главах рассказывалось, как создать визуальные компоненты интерактивной визуализации Нобелевской премии: временную диаграмму для отображения нобелевских лауреатов по годам, карту, показывающую их распределение по странам, список выбранных на данный момент лауреатов и столбчатую диаграмму для сравнения абсолютного числа лауреатов и их количества на душу населения по стране. Эта глава посвящена взаимодействию пользователя и визуализации данных. Пользователь с помощью выпадающих списков и группы радиокнопок (см. рисунок 21.1) может отфильтровать интересующий его набор данных, который отразят визуальные компоненты. Например, если выбрать номинацию Physics в списке Category, то элементы Nobel-viz отобразят только лауреатов Нобелевской премии по физике. Фильтры в строке меню являются накопительными, то есть можно применить сразу несколько фильтров, допустим, выбрать только женщин из Франции, которые получили Нобелевскую премию по физике[1].



Рис. 21.1. *Описанная в этой главе строка меню*

В следующей секции я покажу, как применять D3 для создания строки меню и как использовать коллбэк-функции JavaScript в ответ на изменения, инициированные пользователем.

[1] Примечательно, что такую премию получили Мария Кюри и ее дочь Ирен Жолио-Кюри (Irène Joliot-Curie).

Создание HTML-элементов с помощью D3

Многие считают, что D3 — инструмент, который специализируется только на создании SVG-визуализаций, состоящих из графических примитивов, таких как линии и круги. D3 действительно прекрасно справляется с такими задачами, но не менее уверенно она создает традиционные HTML-элементы, например: таблицы и выпадающие списки. При создании сложных структур HTML, управляемых данными, таких как иерархические меню, вложенные объединения данных D3 идеально справляются с созданием элементов DOM и коллбэк-функций для обработки выборок пользователя.

В главе 20 мы увидели, насколько легко создать строки таблицы для отфильтрованного набора данных или заполнить область биографии данными лауреата. В этой главе мы будем заполнять выпадающие списки опциями, основанными на изменяющихся наборах данных, и прикреплять коллбэк-функции к элементам пользовательского интерфейса: выпадающим спискам и радиокнопкам.



Стабильные элементы HTML (например, выпадающий список, который не зависит от изменения данных) лучше всего написать на HTML, а затем использовать D3 для присоединения коллбэк-функций, необходимых для обработки пользовательского ввода. Как и в случае со стилями CSS, следует как можно больше использовать чистый HTML. Это сохраняет чистоту кодовой базы и облегчает ее понимание как разработчиками, так и не разработчиками. В этой главе я слегка нарушаю это правило, чтобы продемонстрировать создание элементов HTML с помощью D3, но все же лучше его соблюдать.

Создание строки меню

Как уже говорилось в подразделе «HTML-каркас», наш проект Nobel-viz построен на основе HTML-блоков `<div>` с плейсхолдерами, которые «облекаются плотью» с помощью JavaScript и D3. Как видно из примера 21-1, строка меню построена в контейнере `<div>` с идентификатором nobel-menu, который размещен выше контейнера с ID chart-holder. Строка меню состоит из трех выпадающих списков для выбора по номинации, полу и стране, а также двух радиокнопок для выбора показателя числа лауреатов в странах (абсолютного или на душу населения).

Пример 21-1. *HTML-каркас для строки меню*

```
<!-- ... -->
<body>
<!-- ... -->
   <!-- ПЛЕЙСХОЛДЕРЫ ДЛЯ ВИЗУАЛЬНЫХ КОМПОНЕНТОВ -->
      <div id="nbviz">
         <!-- НАЧАЛО СТРОКИ МЕНЮ -->
         <div id="nobel-menu">
            <div id="cat-select">
               Category
               <select></select>
            </div>
            <div id="gender-select">
               Gender
               <select>
                  <option value="All">All</option>
                  <option value="female">Female</option>
                  <option value="male">Male</option>
               </select>
            </div>
            <div id="country-select">
               Country
               <select></select>
            </div>
            <div id='metric-radio'>
               Number of Winners: 
               <form>
                  <label>absolute
                     <input type="radio" name="mode"
                           value="0" checked>
                     </label>
                  <label>per-capita
                     <input type="radio" name="mode" value="1">
                  </label>
               </form>
            </div>
         </div>
         <!-- КОНЕЦ СТРОКИ МЕНЮ -->
```

```
    <div id='chart-holder'>
  <!- ... ->
  </body>
```

Теперь будем по очереди добавлять элементы UI, начиная с выпадающих списков.

Создание выпадающего списка номинаций

Чтобы создать выпадающий список номинаций, нам понадобится список из строк опций. Создадим этот список с помощью списка `CATEGORIES`, определенного в файле *nbviz_core.js*:

```
import nbviz from './nbviz_core.mjs'

let catList = ['All Categories'].concat(nbviz.CATEGORIES) ❶
```

❶ Создаем выпадающий список номинаций `['All Categories', 'Chemistry', 'Econo mics', ...]`, объединяя списки `['All Categories']` и `['Chemistry', 'Economics', ...]`.

Теперь используем этот список номинаций для создания тегов опций. Сначала применим D3, чтобы захватить тег select с идентификатором `#cat-select`.

```
//...
let catSelect = d3.select('#cat-select select');
```

Получив переменную `catSelect`, используем стандартное объединение данных D3, чтобы преобразовать список номинаций `catList` в HTML-теги `option`:

```
catSelect.selectAll('option')
    .data(catList)
    .join('option') ❶
    .attr('value', d => d) ❷
    .html(d => d);
```

❶ После привязки данных добавляем тег `option` к каждому элементу списка `catList`.

❷ Указываем атрибут `value` и текст опции для номинации (например, `<option value="Peace">Peace</option>`).

В результате предыдущих операций `append` получается следующий DOM-элемент `cat-select`:

```
<div id="cat-select">
   "Category "
   <select>
      <option value="All Categories">All Categories</option>
      <option value="Chemistry">Chemistry</option>
      <option value="Economics">Economics</option>
      <option value="Literature">Literature</option>
      <option value="Peace">Peace</option>
      <option value="Physics">Physics</option>
      <option value="Physiology or Medicine">
      Physiology or Medicine</option>
   </select>
</div>
```

Создав выпадающий список, используем D3-метод `on`, чтобы присоединить коллбэк-функции обработчика событий, срабатывающего при изменении значения списка:

```
catSelect.on('change', function(d) {
        let category = d3.select(this).property('value'); ❶
        nbviz.filterByCategory(category); ❷
        nbviz.onDataChange(); ❸
});
```

❶ Ключевое слово `this` вместе со свойством `value` позволяет обратиться к выбранной опции номинации.

❷ Вызываем из *nbviz_core.js* метод `filterByCategory`, чтобы отфильтровать набор данных по выбранной номинации премии.

❸ `onDataChange` запускает методы обновления визуальных компонентов, которые изменятся в соответствии с новым отфильтрованным набором данных.

Рисунок 21.2 схематично изображает коллбэк выбора. При выборе номинации Physics вызывается анонимная коллбэк-функция, закрепленная за событием

изменения значения выпадающего списка. Эта функция инициирует обновление визуальных элементов Nobel-viz.

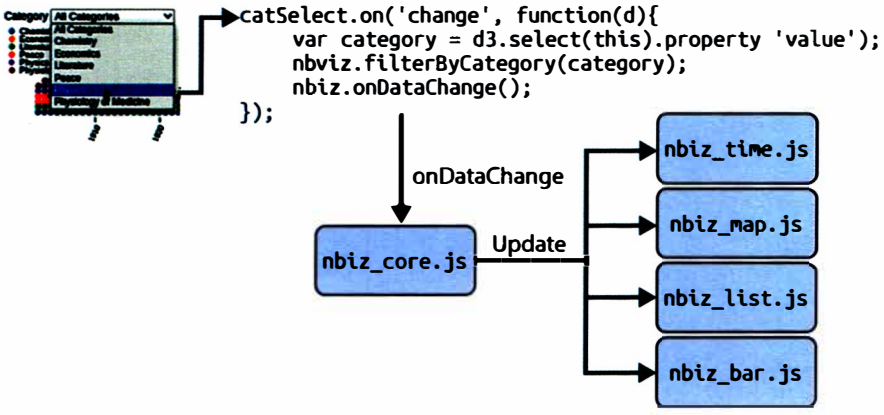


***Рис. 21.2.** Коллбэк-функция для списка номинаций*

С помощью коллбэк-функции выпадающего списка номинаций сначала вызываем метод `filterByCategory`[1], чтобы выбрать только лауреатов премии по физике, и метод `onDataChange` для запуска обновления всех визуальных компонентов. Изменение данных отразится там, где это требуется. Например, на карте круглые индикаторы изменят размер, а в странах, где нет лауреатов по физике, исчезнут.

Добавление выпадающего списка полов

Мы уже добавили HTML-код для выпадающего списка полов и его опций в описание строки меню в *index.html*:

```
<!- ... ->
      <div id="gender-select">
         Gender
         <select>
            <option value="All">All</option>
            <option value="female">Female</option>
            <option value="male">Male</option>
         </select>
      </div>
<!- ... ->
```

[1] Определен в файле *nbviz_core.js*.

Теперь осталось выбрать тег `select` для пола и добавить коллбэк-функцию для обработки действий пользователя. Применим для этого D3-метод `on`:

```
d3.select('#gender-select select')
   .on('change', function(d) {
      let gender = d3.select(this).property('value');
      if(gender === 'All'){
         nbviz.genderDim.filter(); ❶
      }
      else{
         nbviz.genderDim.filter(gender);
      }
      nbviz.onDataChange();
   });
```

❶ Вызов фильтра genderDim без аргумента сбрасывает его значение в All.

Сначала мы выбираем значение опции выпадающего списка. Затем используем это значение для фильтрации текущего набора данных. В завершение вызываем `onDataChange`, чтобы инициировать все изменения визуальных компонентов Nobel-viz, вызванные новым набором данных.

С помощью CSS задаем для выпадающего списка Gender левый внешний отступ в 20 пикселей, чтобы отодвинуть его от выпадающего списка Category:

```
#gender-select{margin-left:20px;}
```

Добавление выпадающего списка стран

Добавить выпадающий список для выбора стран немного сложнее, чем для выбора номинаций и пола. Список стран, где есть нобелевские лауреаты, довольно длинный (см. рисунок 17.1), при этом во многих странах всего один-два лауреата. Конечно, можно добавить их все, но выпадающий список станет длинным и неудобным. Лучше создадим группы для стран с одним и двумя лауреатами, уменьшив число опций ради легкости управления и добавив контекст к диаграмме — распределение стран с одним или двумя лауреатами на временной диаграмме не по одиночке, а группой может продемонстрировать изменяющуюся динамику распределения Нобелевской премии[1].

[1] Из диаграммы видно, что среди стран с одним лауреатом преобладают Нобелевские премии мира, за которыми следуют премии по литературе.

Чтобы добавить группы стран с одним и двумя лауреатами, нам понадобится измерение стран с с помощью кросс-фильтра для получения размеров групп стран. То есть выпадающий список стран будет создан после загрузки набора данных о нобелевских лауреатах. Для этого мы передаем его в метод `nbviz.initUI`, который вызывается в основном скрипте *nbviz_main.js* после создания измерений кросс-фильтра.

Код ниже создает выпадающий список. Страны, где есть не менее трех лауреатов, получают собственное место в выпадающем списке, расположенное ниже All Winners. Страны с одним или двумя лауреатами добавляются в соответствующие группы выпадающего списка: Single Winning Countries и Double Winning Countries. По этим опциям их может выбрать пользователь.

```
export let initMenu = function() {
   let ALL_WINNERS = 'All Winners';
   let SINGLE_WINNERS = 'Single Winning Countries';
   let DOUBLE_WINNERS = 'Double Winning Countries';

   let nats = nbviz.countrySelectGroups = nbviz.countryDim
      .group().all() ❶
      .sort(function(a, b) {
         return b.value - a.value; // по убыванию
      });

   let fewWinners = {1:[], 2:[]}; ❷
   let selectData = [ALL_WINNERS];

   nats.forEach(function(o) {
      if(o.value > 2){ ❸
         selectData.push(o.key);
      }
      else{
         fewWinners[o.value].push(o.key); ❹
      }
   });

   selectData.push(
      DOUBLE_WINNERS,
      SINGLE_WINNERS
   );
```

```
    //...
})
```

❶ Сортируем массив групп вида (`{key:"United States", value:336}, ...`), где `value` — число лауреатов в стране.

❷ Объект со списками для хранения стран с одним и двумя лауреатами.

❸ Страна, где не менее трех лауреатов, получает отдельный слот в списке `selectData`.

❹ Страны с одним и двумя лауреатами добавляются в списки, соответствующие их числу (value): 1 или 2.

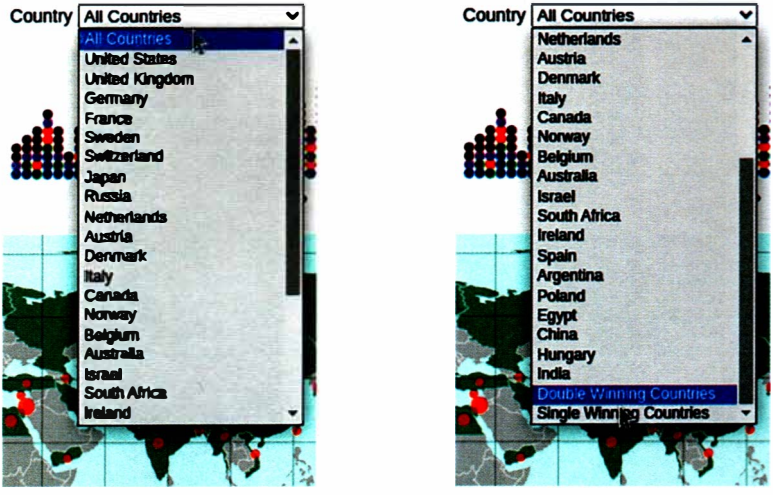


***Рис. 21.3.** Выпадающий список стран*

Получив список `selectData` с соответствующими массивами `fewWinners`, мы можем использовать его для создания опций выпадающего списка стран. С помощью D3 получаем тег `select` выпадающего списка стран, затем добавляем опции через стандартную привязку данных:

```
let countrySelect = d3.select('#country-select select');
countrySelect
   .selectAll("option")
   .data(selectData)
   .join("option")
   .attr("value", (d) => d)
   .html((d) => d);
```

После добавления опций `selectData` выпадающий список выглядит, как на рисунке 21.3.

Теперь нам нужна только коллбэк-функция, срабатывающая при выборе опции, которая отфильтрует основной набор данных по выбранной стране. В следующем коде вы увидите, как это сделать. Сначала получим свойство `value` (1) — отдельную страну или одно из значений: `ALL_WINNERS`, `DOUBLE_WINNERS` и `SINGLE_WINNERS`. Затем мы создадим список стран, чтобы передать его методу `nbviz.filterByCountries` в файле *nbviz_core.js*.

```
countrySelect.on('change', function(d) {

    let countries;
    let country = d3.select(this).property('value');

    if(country === ALL_WINNERS){ ❶
      countries = [];
    }
    else if(country === DOUBLE_WINNERS){
      countries = fewWinners[2];
    }
    else if(country === SINGLE_WINNERS){
      countries = fewWinners[1];
    }
    else{
      countries = [country];
    }

    nbviz.filterByCountries(countries); ❷
    nbviz.onDataChange(); ❸
});
```

❶ С помощью этих условных конструкций создаем массив `countries`, в зависимости от значения строки `country`. Мы получим пустой массив, массив с одним значением или один из массивов `fewWinners`.

❷ Вызываем `filterByCountries` для фильтрации основного набора данных нобелевских лауреатов с помощью массива стран. Запускаем обновление всех элементов Nobel-viz.

❸ Функция `filterByCountries` показана в примере 21-2. Если аргумент `countryNames` пустой, то фильтр сбрасывается. В противном случае мы фильтруем измерение страны `countryDim` для всех стран в `countryNames`.

SKLADCHIK.ORG

Пример 21-2. Функция для фильтрации по странам

```
nbviz.filterByCountries = function(countryNames) {

   if(!countryNames.length){ ❶
      nbviz.countryDim.filter();
   }
   else{
      nbviz.countryDim.filter(function(name) { ❷
         return countryNames.indexOf(name) > -1;
      });
   }

   if(countryNames.length === 1){ ❸
      nbviz.activeCountry = countryNames[0];
   }
   else{
      nbviz.activeCountry = null;
   }
};
```

❶ Сбрасывает фильтр, если массив `countryNames` пустой (пользователь выбрал All Countries).

❷ Создаем функцию фильтрации по измерению кросс-фильтра (`crossfilter`) для стран, которая возвращает `true`, если страна есть в списке `countryNames` (содержащем либо одну страну, либо группу стран с одним или с двумя лауреатами).

❸ Ведет учет любой единственной выбранной страны в определенном порядке, например, для ее подсвечивания на карте и столбчатой диаграмме.

Выпадающие списки для фильтрации данных по номинации, полу и странам созданы, осталось только добавить коллбэк-функцию для обработки изменений радиокнопок, переключающих с показателя абсолютного числа лауреатов на пересчет на душу населения.

Подключение радиокнопок

Радиокнопки мы создали ранее с помощью HTML-кода, включающего форму с тегом `<input>` с типом `radio`:

```
<div   id='metric-radio'>
   Number of Winners:  ❶
   <form>
      <label>absolute
         <input type="radio" name="mode" value="0" checked> ❷
      </label>
      <label>per-capita
         <input type="radio" name="mode" value="1">
      </label>
   </form>
</div>
```

❶ ` ` создает неразрывный пробел между формой и ее подписью.

❷ Оба элемента input с типом `radio`, имеющие одно и то же имя `mode`, группируются. При активации одной радиокнопки другая деактивируется. Они различаются по значению (value), в данном случае: `0` и `1`. Мы используем атрибут `checked` для изначальной установки радиокнопки в активное состояние `0`.

Поскольку форма с радиокнопками уже есть, нам остается только выбрать все элементы input и добавить коллбэк-функцию для обработки вызывающих изменения нажатий на кнопки:

```
d3.selectAll('#metric-radio input').on('change', function() {
        var val = d3.select(this).property('value');
        nbviz.valuePerCapita = parseInt(val); ❶
        nbviz.onDataChange();
    });
```

❶ Обновляем значение `valuePerCapita` перед вызовом `onDataChange` и запуском обновления визуальных элементов.

Сохраняем текущее состояние радиокнопок с помощью целочисленного значения `valuePerCapita`. Когда пользователь выбирает одну из радиокнопок, это значение меняется и с помощью `onDataChange` запускается перерисовка визуальных элементов с новым показателем.

Итак, готовы все элементы строки меню Nobel-viz, позволяющие пользователю уточнять критерии отображения набора данных и детализировать интересующие подмножества.

Резюме

В этой главе мы рассмотрели добавление выпадающих списков и блока радиокнопок к визуализации данных о лауреатах Нобелевской премии. Существует ряд других HTML-элементов пользовательского интерфейса, таких как группы флажков (чекбоксов), элементы выбора времени, простые кнопки и др[1]. Но их реализация также следует шаблонам, описанным в этой главе. Список данных используется для добавления и вставки элементов DOM, настройки свойств там, где это необходимо, а коллбэк-функции привязываются к событиям change. Это очень мощный способ работы с такими идиомами D3 (и JS), как цепочка методов и анонимные функции. Он быстро становится привычной частью вашего рабочего процесса с использованием D3.

[1] В HTML5 также есть нативные слайдеры, ранее для их создания приходилось использовать плагины jQuery.

Заключение

В основу этой книги положено последовательное изложение действий по преобразованию базовых HTML-страниц Википедии в современную интерактивную веб-визуализацию на JavaScript. По мере необходимости книгу также можно использовать как справочник. Каждый раздел является самостоятельным, что позволяет рассматривать различные стадии обработки набора данных независимо друг от друга. Давайте вкратце повторим рассмотренный материал, а затем поговорим о некоторых идеях, которые могут быть полезны для дальнейшей работы с визуализациями.

Подведение итогов

Книга содержит пять разделов. Первый раздел — введение в базовый набор инструментов Python и JavaScript для визуализации. В остальных четырех показано, как собирать и очищать сырые данные, анализировать их и превращать в современную веб-визуализацию. Этот процесс очистки и преобразования был направлен на решение задач визуализации данных: взять довольно примитивный список нобелевских лауреатов из Википедии и преобразовать содержащийся в нем набор данных во что-то более интересное и информативное. Давайте кратко сформулируем, какие основные уроки можно извлечь из каждого раздела.

Раздел I. Базовый пакет инструментов

Наш базовый набор инструментов составляют:

— Обучающий мостик между Python и JavaScript, который призван сгладить переход между языками, подчеркнуть их сходные элементы и подготовить окружение для создания современной визуализации с помощью обоих языков. В текущей реализации у Python и JavaScript много общего, поэтому переключаться с одного на другой стало проще.

— Одна из сильных сторон Python — чтение/запись основных форматов (например, JSON и CSV), а также поддержка баз данных (как SQL, так и NoSQL). Мы увидели, как легко Python передает данные, преобразуя их из одного формата в другой и меняя базы данных по мере необходимости. Такая гибкость в управлении данными — ключевой элемент, обеспечивающий плавную работу любого тулчейна визуализации.
— Мы охватили базовые навыки веб-разработки, необходимые для создания современной интерактивной визуализации в браузере. Чтобы минимизировать рутинное веб-программирование и сосредоточиться на разработке ваших визуальных проектов, мы создали одностраничное веб-приложение (Single-Page Aapplication, SPA) на JavaScript. Введение в SVG — основной строительный блок D3-визуализаций — подготовил нас к созданию нашей визуализации данных в разделе IV.

Раздел II. Подготовка данных

В этой части книги мы рассмотрели, как самому получить данные из интернета с помощью Python, если вам не предоставили готовый файл с чистыми данными:

— Если вам повезло, и в открытом доступе есть такой файл в подходящем формате, например JSON или CSV, то достаточно отправить простой HTTP-запрос. Кроме того, для вашего набора данных может отыскаться специальный web API, хорошо, если это будет RESTful API. В качестве примера мы рассмотрели применение Python-библиотеки Tweepy для доступа к Twitter API. Мы также увидели, как использовать Google Таблицы, популярный инструмент для обмена данными в визуализации.
— Задача усложняется, если данные представлены в интернете в формате, ориентированном на людей, например: в виде HTML-таблицы, списков или структурированного контента. Тогда для извлечения сырого HTML-контента придется использовать *скрейпинг*, а затем с помощью парсера получить доступ к внедренным в него данным. Мы использовали для скрейпинга легковесную библиотеку Beautiful Soup и куда более многофункциональную и тяжеловесную Scrapy, самую крупную звезду веб-скрейпинга на небосклоне Python.

Раздел III. Очистка и исследование данных с помощью pandas

В этом разделе для очистки и исследования наборов данных мы задействовали «тяжелую артиллерию» — Python-библиотеку pandas, мощный программируемый

аналог электронных таблиц. Сначала мы рассмотрели pandas как часть экосистемы NumPy, предоставляющей доступ к мощным низкоуровневым библиотекам для быстрой обработки массивов данных. Особое внимание мы уделили очистке и анализу набора данных о нобелевских лауреатах с помощью pandas:

— Даже те данные, которые получены через официальные web API, в основном грязные. Чтобы очистить их и подготовить для визуализации, потребуется гораздо больше времени, чем вы, вероятно, ожидаете. Прежде, чем приступить к исследованию данных и их последующей визуализации, мы постепенно очистили набор данных о лауреатах. Нашли и удалили неточные даты, аномальные типы данных, пропущенные поля и прочую «грязь».
— Имея очищенный (насколько это было возможно) набор данных о Нобелевской премии, мы увидели, насколько просто с помощью pandas и Matplotlib интерактивно исследовать данные, легко создавая встроенные диаграммы, выполняя срезы данных различными способами и получая общее представление о них, параллельно выискивая те интересные идеи, которые стоит донести с помощью визуализации.

Раздел IV. Доставка данных

Из этой части мы узнали, как с помощью Flask создать минимальный data API, чтобы доставлять в браузер данные как в статическом, так и в динамическом режиме.

Сначала мы рассмотрели, как использовать Flask для работы со статическими файлами, а затем — как запустить собственный базовый data API для передачи данных из локальной БД. Минимализм Flask позволяет создать очень тонкий сервисный слой между результатами обработки данных с помощью Python и их конечной визуализацией в браузере. Прелесть открытого ПО в том, что всегда можно найти надежную и простую в использовании библиотеку, которая решит вашу задачу лучше, чем если бы вы делали все вручную. Во второй главе раздела мы узнали, насколько просто использовать лучшие в своем классе Python-библиотеки (на примере Flask) для создания надежного и гибкого RESTful API, готового к передаче данных онлайн. Также мы рассмотрели простое развертывание сервера данных на облачной платформе Heroku, популярной среди питонистов.

Раздел V. Визуализация данных с помощью D3 и Plotly

В первой главе раздела мы рассмотрели, как из данных, отобранных после анализа с помощью pandas, создать диаграммы и карты и разместить их в сети.

Matplotlib позволяет создавать статические диаграммы полиграфического качества, а Plotly — добавлять элементы управления и динамические диаграммы. Также мы узнали, как перенести диаграмму Plotly напрямую из Jupyter Notebook на веб-страницу.

Надо отметить, что использовать D3 для этой книги было очень смелой идеей, но я был полон решимости продемонстрировать построение многоэлементной визуализации, какие могут вам потребоваться в профессиональной деятельности. Один из плюсов библиотеки D3 — множество примеров ее применения, доступных в сети, но большинство из них демонстрирует отдельные техники, а примеры координации нескольких визуальных элементов встречаются редко. В главах о D3 мы увидели, как синхронизировать обновление временной диаграммы (отображающей все Нобелевские премии), карты, столбчатой диаграммы и списка лауреатов при фильтрации набора данных Нобелевских премий или изменении показателя присуждения премии (абсолютного или на душу населения).

Освоение тем, которые рассмотрены в этих главах, позволит вам дать волю воображению и учиться на практике. Я бы порекомендовал выбрать интересные для вас данные и на их основе разработать что-нибудь с помощью D3.

Дальнейшее развитие

Как я уже говорил, Python и JavaScript для обработки и визуализации данных сейчас невероятно активны и строятся на очень прочной основе.

Хотя процессы получения и очистки данных, рассмотренные в разделе II и главе 9, постепенно совершенствуются и упрощаются по мере роста вашего мастерства (например, владения pandas), Python продолжает активно предлагать новые мощные инструменты обработки данных. Достаточно полный их список доступен на вики-странице Python. Предлагаю вашему вниманию несколько идей для создания ваших будущих визуализаций.

Визуализация данных из социальных медиа

С появлением социальных медиа возник огромный пласт интересных данных, которые зачастую легко скрейпить или получать через web API. Также существуют специально подобранные наборы данных социальных медиа, например Stanford Large Network Dataset Collection или UCIrvine collection (коллекции Калифорнийского университета в Ирвайне). На этих наборах данных можно потренироваться в области визуализации сетей, которая становится все популярнее.

Две самые популярные Python-библиотеки для сетевого анализа — это graph-tool и Network X. Хотя graph-tool более оптимизирована, NetworkX считается более удобной для пользователя. Обе библиотеки работают с графами распространенных форматов GraphML и GML. Библиотека D3 не может напрямую читать GML-файлы, но их достаточно легко преобразовать в понятный для нее формат JSON. Обратите внимание, что в D3 версии 4 изменился forceSimulation API. Краткое введение в новый API, который использует объект `forceSimulation` для отслеживания узлов см. на Pluralsight.

Визуализации машинного обучения

Машинное обучение сейчас на пике популярности, и Python предлагает фантастический набор инструментов для анализа данных с использованием множества алгоритмов от базовой регрессии до передовых ансамблевых методов вроде случайного леса (Random Forest, RF). См. замечательный обзор различных разновидностей алгоритмов.

Лидер среди Python-инструментов машинного обучения — scikit-learn, часть экосистемы NumPy, куда также входят SciPy и Matplotlib. Библиотека scikit-learn предоставляет поразительные возможности для эффективного получения и анализа данных. Алгоритмы, на создание которых ранее уходили недели, теперь доступны через импорт, хорошо спроектированы и дают полезные результаты в несколько строк кода.

С помощью таких инструментов, как scikit-learn, можно обнаруживать глубоко спрятанные закономерности в данных (если таковые существуют). Чтобы познакомиться с некоторыми методами машинного обучения и узнать, как D3 используется для визуализации процесса и результатов, см. отличную демонстрацию на R2D3. Эта демонстрация — яркий пример свободы творчества, которая достигается мастерским владением D3. Становится ясно, что хорошая веб-визуализация данных расширяет границы возможностей, создавая новые виды визуальных элементов, которые вовлекают так, как никогда раньше, и доступ к которым может получить любой пользователь.

В репозитории на GitHub собрана грандиозная коллекция IPython/Jupyter блокнотов для статистики, машинного обучения и Data Science. Из них можно почерпнуть приемы визуализации, которые после адаптации и расширения подойдут для ваших собственных работ.

Заключительные замечания

Я упомянул лишь малую часть тех областей, где вы можете применить новые навыки визуализации данных с помощью Python и JavaScript. Надеюсь, эта книга заложит прочную основу для развития вашего мастерства веб-визуализации данных, которое пригодится для создания собственных проектов или для поиска работы. Эти навыки сейчас становятся очень востребованными. Рассмотренные нами Python-библиотеки с их огромными возможностями обработки данных, а также решения широкого спектра задач, и JavaScript-библиотеки (самая выдающаяся среди которых D3) с их мощными и зрелыми средствами веб-визуализации представляют самый богатый стек для визуализации данных изо всех мне известных. Навыки в этой области уже сейчас приносят ощутимую прибыль, но темп изменений и масштаб доходности продолжают быстро расти. Надеюсь, эта интересная и перспективная область принесет вам такое же удовлетворение, как и мне.

Паттерн enter/exit библиотеки D3

Теперь в библиотеке D3 есть более удобный метод `join`, заменяющий старую реализацию привязки/объединения данных с использованием паттерна на основе методов `enter`, `exit` и `remove`. Метод `join` — ценное дополнение к D3, но в сети можно встретить тысячи примеров использования старого паттерна привязки данных. Чтобы использовать или преобразовать эти примеры, полезно узнать, что происходит под капотом, когда D3 объединяет данные.

Давайте рассмотрим, как D3 это делает. Начнем с диаграммы без столбиков, с холстом SVG и группой диаграммы:

```
...
      <div id="nobel-bar">
         <svg width="600" height="400">
            <g class="chart" transform="translate(40, 20)"></g>
         </svg>
      </div>
...
```

Чтобы объединить данные с помощью D3, нужно получить их в подходящей форме. Обычно это массив объектов, наподобие `nobelData` нашей столбчатой диаграммы:

```
var nobelData = [
   {key:'United States', value:336},
   {key:'United Kingdom', value:98},
   {key:'Germany', value:79},
   ...
]
```

Объединение данных с помощью D3 выполняется в два этапа: сначала добавим данные, которые хотим привязать, с помощью метода `data`, затем выполним их привязку с помощью метода `join`.

SKLADCHIK.ORG

Чтобы добавить данные о Нобелевской премии к группе столбиков, сначала выбираем контейнер для столбиков, в данном случае — группу элементов SVG с классом `chart`. Затем определяем контейнер, в данном случае — выбираем все элементы с классом `bar`:

```
var svg = d3.select('#nobel-bar.chart');

var bars = svg.selectAll('.bar')
            .data(nobelData);
```

Теперь мы подошли к несколько парадоксальному аспекту метода `data` библиотеки D3. Первый `select` вернул группу `chart` на нашем холсте SVG `nobel-bar`, но второй `selectAll` вернул все элементы с классом `bar`, которых нет на диаграмме. Если столбиков нет, то к чему же мы привязываем данные? Дело в том, что D3 ведет свою внутреннюю бухгалтерию, поэтому объект `bars`, возвращенный методом `data`, знает, какие элементы DOM были привязаны к `nobelData`, а какие — нет. Теперь рассмотрим, как использовать эту особенность, применяя фундаментальный метод `enter`.

Метод enter

Метод `enter` (и родственный ему `exit`) одновременно является как базисом мощи и выразительности D3, так и причиной многих недоразумений. Хотя новый метод `join` упрощает ситуацию, `enter` стоит освоить, чтобы действительно улучшить свои навыки работы с D3. Познакомимся с ним поближе на примере очень простой и неторопливой демонстрации.

Начнем с канонически простой демонстрации — добавим прямоугольник-столбик для каждого элемента данных о Нобелевской премии. В качестве привязанных данных используем первые шесть стран с нобелевскими лауреатами:

```
var nobelData = [
   {key:'United States', value:200},
   {key:'United Kingdom', value:80},
   {key:'France', value:47},
   {key:'Switzerland', value:23},
   {key:'Japan', value:21},
   {key:'Austria', value:12}
];
```

Теперь в помощью D3 получим группу диаграммы и присвоим ее переменной `svg`. Используем ее, чтобы сделать выборку всех элементов с классом `bar` (которых пока что нет):

```
var svg = d3.select('#nobel-bar.chart');

var bars = svg.selectAll('.bar')
            .data(nobelData);
```

Теперь, хотя выборка `bars` пустая, D3 сохранила запись данных, которые мы только что к ней привязали. Мы можем этим воспользоваться и применить метод `enter` для создания нескольких столбиков с помощью данных. Вызов `enter` для выборки `bars` возвращает подвыборку всех данных (в данном случае `nobelData`), которые не были привязаны к столбику. Поскольку в исходной выборке столбиков не было (диаграмма пустая) и все данные не привязаны, то `enter` возвращает выборку входных значений (прейсхолдеры узлов для всех непривязанных данных) размером шесть:

```
bars = bars.enter(); # возвращает шесть плейсхолдеров узлов
```

Используем эти прейсхолдеры в `bars`, чтобы создать элементы DOM — в данном случае несколько столбиков. Мы не будем переворачивать столбики (в SVG ось Y по соглашению направлена вниз), просто зададим их позиционирование и высоту с помощью значений данных и индексов:

```
bars.append('rect')
    .classed('bar', true)
    .attr('width', 10)
    .attr('height', function(d){return d.value;}) ❶
    .attr('x', function(d, i) {return i * 12;});
```

❶ Если вы предоставляете коллбэк-функцию сеттерам D3 (`attr`, `style` и др.), то первый и второй аргументы — это значение отдельного объекта данных (например, `d == {key: 'United States', value: 200}`) и его индекс (`i`).

С помощью коллбэк-функций задаем высоту столбиков и их позиции по оси X (с отступом между столбиками по 2 пикселя) и вызываем `append` для наших шести узлов. Результат показан на рисунке A.1.



Рис. А.1. *Создание столбиков с помощью метода D3 enter*

Для изучения HTML-кода, который вы создаете с помощью D3, я рекомендую использовать вкладку Elements в браузере Chrome (или эквивалентную). Исследование нашей столбчатой мини-диаграммы с помощью Elements показано на рисунке А.2.

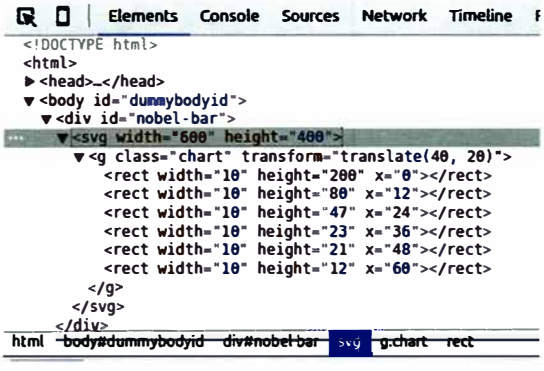


Рис. А.2. *Просмотр HTML-кода, сгенерированного enter и append, во вкладке Elements*

Итак, мы увидели, что происходит, если вызвать `enter` для пустой выборки. Но что случится, если у нас уже есть несколько столбиков, как было бы в интерактивной диаграмме с набором данных, который может изменять пользователь?

Добавим пару прямоугольников с классом `bar` в исходный HTML-код:

```
<div id="nobel-bar">
   <svg width="600" height="400">
      <g class="chart" transform="translate(40, 20)">
```

```
        <rect class='bar'></rect>
        <rect class='bar'></rect>
      </g>
    </svg>
  </div>
```

Если повторить ту же привязку и ввод данных при вызове `data` для нашей выборки, то два плейсхолдера прямоугольников будут привязаны к первым двум элементам массива `nobelData` (то есть `[{key: 'United States', value: 200}, {key: 'United Kingdom', value:80}]`). То есть `enter`, который возвращает плейсхолдеры непривязанных данных, теперь вернул только четыре плейсхолдера последних четырех элементов массива `nobelData`:

```
var svg = d3.select('#nobel-bar.chart');

var bars = svg.selectAll('.bar')
            .data(nobelData);

bars = bars.enter(); # возвращает четыре плейсхолдера узлов
```

Если, как в предыдущий раз, мы вызовем `append` для введенного `bars`, то получим результат, показанный на рисунке А.3 — отобразятся четыре столбика. Обратите внимание, что они сохраняют свой индекс `i`, по которому определяется позиция на оси X.



Рис. А.3. *Результат вызова enter и append в случае существующих столбиков*

На рисунке А.4 показан HTML-код, сгенерированный для последних четырех столбиков. Мы видим, что данные первых двух элементов теперь привязаны к двум фиктивным узлам, которые мы добавили в исходную группу столбиков. Мы еще не использовали их для настройки атрибутов этих прямоугольников. Обновление старых столбиков новыми данными — один из ключевых элементов паттерна обновления, который мы вскоре рассмотрим.

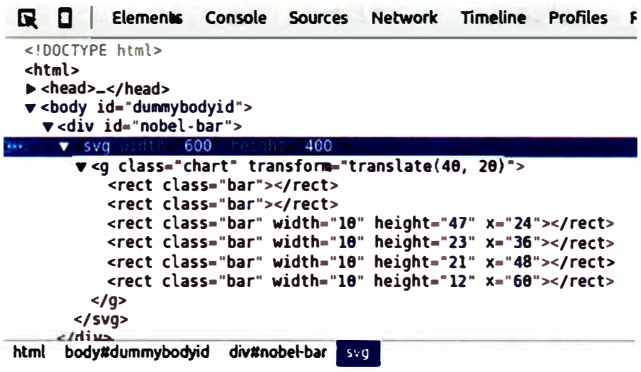


Рис. А.4. *Просмотр HTML-кода, сгенерированного для частичной выборки методами enter и append, во вкладке Elements*

Подчеркну, что понимание особенностей `enter` и `exit` (а также `remove`) имеет решающее значение для глубокого освоения D3. Поэкспериментируйте, проверьте HTML-код, который вы создаете, введите немного данных и поманипулируйте ими всячески, изучая все аспекты. Прежде чем перейти к паттерну обновления D3, рассмотрим доступ к привязанным данным.

Доступ к привязанным данным

Хороший способ увидеть, что происходит с DOM, — использовать HTML-инспектор и консоль браузера для отслеживания изменений, сделанных D3. На рисунке А.1 изображена консоль Chrome, на которой мы видим элемент `rect`, представляющий первый столбик с рисунка А.3 до привязки данных и после привязки `nobelData` к столбикам с помощью метода `data`. Как вы можете видеть, библиотека D3 добавила к элементу `rect` объект `__data__`, с помощью которого она будет хранить привязанные данные — первый элемент списка `nobelData`.

Объект `__data__` используется внутренней бухгалтерией D3, данные в нем становятся доступными для функций, которые передаются в методы обновления, такие как `attr`.

Рассмотрим, как использовать данные из объекта `__data__` элемента, чтобы задать его атрибут `name`. Атрибут `name` удобно использовать при выполнении конкретных выборок D3. Например, если пользователь выбирает конкретную страну, то мы можем использовать D3, чтобы получить все именованные компоненты и при необходимости настроить их стиль. Используем столбик

с привязанными данными (рисунок А.5) и зададим имя, используя свойство `key` привязанных к нему данных:

```
let bar = d3.select('#nobel-bar.bar');

bar.attr('name', function(d, i){ ❶

   let sane_key = d.key.replace(/ /g, '_'); ❷

   console.log('__data__ is: ' + JSON.stringify(d)
   + ', index is ' + i)

   return 'bar__' + sane_key; ❸
   });
// вывод в консоль:
//__data__  is: {"key":"United States","value":336}, index is 0
```

❶ Все сеттер-методы D3 в качестве второго аргумента могут принимать функцию. Эта функция получает данные (`d`), привязанные к выбранному элементу, и его позицию в массиве данных (`i`).

❷ Заменим все пробелы в ключе на подчеркивания (например, United States → United_States) с помощью регулярного выражения (regex).

❸ Значение атрибута `name` будет установлено в `'bar__United_States'`.

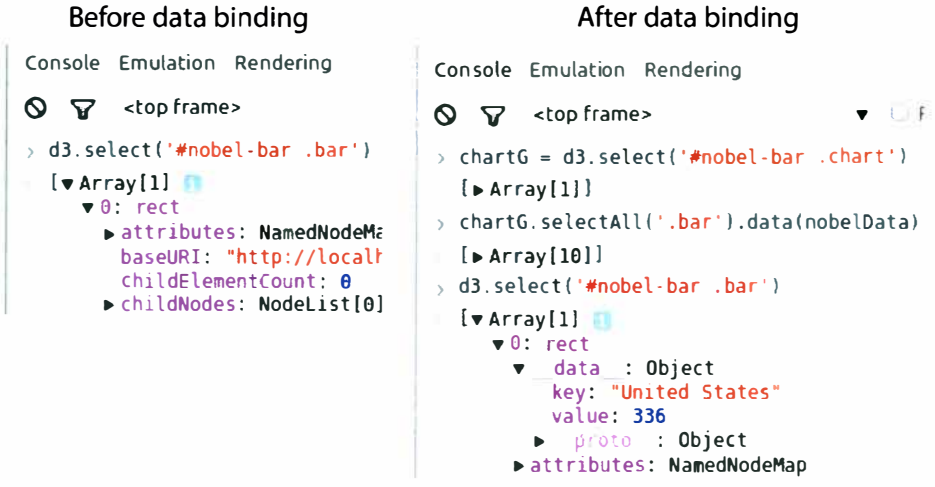


Рис. А.5. *В консоли Chrome показано добавление объекта __data__ после привязки данных с использованием метода D3 data*

Все сеттер-методы, указанные на рисунке 17.3 (`attr`, `style`, `text` и др.), могут принимать в качестве второго аргумента функцию, которая будет получать данные, привязанные к элементу, и индекс элемента в массиве. Результат, который возвращает функция, используется для установки значения свойства. Как мы увидим, изменения в наборе данных интерактивных визуализаций отобразятся, когда после привязки новых данных мы используем такие функциональные сеттеры для адаптации атрибутов, стилей и свойств.

Об авторе

Киран Дейл начал свою карьеру как ученый-исследователь, но со временем стал программистом-фрилансером, хакером-энтузиастом, независимым исследователем и иногда предпринимателем. В числе его увлечений бег по пересеченной местности и джазовые импровизации на фортепиано. За 15 с лишним лет исследовательской работы он немало экспериментировал с кодом, изучил множество библиотек и выбрал для себя самые любимые инструменты. По его мнению, сегодня с помощью Python, JavaScript и небольшого количества C++ можно решить большинство задач. Киран Дейл специализируется на быстром прототипировании и анализе осуществимости гипотез с уклоном в алгоритмы, но главное, ему просто нравится создавать крутые вещи.

Послесловие

Пчелы на обложке книги «*Визуализация данных с помощью Python и JavaScript*» — это синеполосая пчела (*Amegilla cingulata*), орхидная пчела (триба *Euglossini*) и голубая пчела-плотник (*Xylocopa caerulea*). Пчелы чрезвычайно важны для сельского хозяйства: они опыляют сельскохозяйственные культуры и другие цветковые растения, а также собирают пыльцу и нектар.

Синеполосая пчела родом из Австралии, водится в лесах, на пустошах и даже в городских районах. В соответствии с названием, этих пчел отличают переливающиеся синие полосы на брюшке: у самцов их пять, а у самок — четыре. Эти пчелы добывают пыльцу с помощью "buzz pollination" («вибрационного опыления») — высвобождают пыльцу с помощью вибрации при жужжании. Этот вид может вибрировать на цветке с поразительной частотой: 350 колебаний в секунду. Многие растения, включая томаты, эффективно опыляются именно таким способом.

Яркоокрашенная орхидейная пчела встречается в тропических лесах Центральной и Южной Америки. Их окраска с металлически отливом может быть ярко-зеленой, синей, золотистой, красной и фиолетовой. Хоботки этих пчел почти в два раза длиннее, чем тело. У мужских особей орхидейных пчел задние ноги видоизмененные, с небольшими мешочками, где собираются и хранятся ароматные соединения, которые затем выделяются (возможно, во время брачного ритуала). Пыльца некоторых видов орхидей скрыта в определенном месте, выделяющем аромат, который привлекает трутней орхидейной пчелы. Таким образом, орхидеи полагаются исключительно на этот вид опыления.

Голубая пчела-плотник — крупное, покрытое голубыми волосками насекомое (длина тела в среднем составляет 2,3 см). Широко распространена в Юго-Восточной Азии, Индии и Китае. Плотником эту пчелу назвали потому, что почти все ее виды гнездятся в засохших деревьях, бамбуке или древесине искусственных сооружений. Они проделывают отверстия, вибрируя своим телом и царапая древесину челюстями (однако древесину выбрасывают, так как питаются нектаром). Пчелы-плотники не образуют колоний, ведут одиночный образ жизни. Однако несколько пчел могут построить гнезда на одной и той же территории.

Иллюстрацию для обложке создала Карен Монтгомери (Karen Montgomery) на основе старинной гравюры из книги *Insects Abroad*. Шрифты обложки: Gilroy Semibold и Guardian Sans. Текст набран шрифтом Adobe Minion Pro, текст заголовков — Adobe Myriad Condensed, программный код набран шрифтом Ubuntu Mono, разработанным Dalton Maag Ltd.

Алфавитный указатель

A
API 414
append 201, 434, 442, 458, 503, 506, 507, 508, 509, 520, 521, 522, 526, 528, 536, 538, 539, 546, 548, 562, 563, 568, 571, 574, 575, 577, 582, 593, 610, 611, 612, 613
Array 542
axis 526

B
Beautiful Soup 194, 202
Beautiful Soup и lxml 190
binning 371

C
Chrome DevTools 130, 140
CORS 420
CSS 137
CSS — каскадность 137
CSS-селекторы 190
CSV 264
CubicInOut 533

D
d3 511
D3 502, 519, 523, 533, 540, 569, 590, 592
D3 group 542
DataFrame 256, 260, 269, 279, 296
DELETE 178
displot 373
DOM 131
drop 296
dtype 246

E
Excel 266

F
FacetGrid 339
filterByCountries 598
flag 163
Flask 388, 418
Flask RESTful для работы с данными 402
flex 427

G
GeoJSON 552
Geopy 436
GET 177, 178
GraphQL 178
graticule 561, 564
gspread 183

H
heatmap 371, 380
Heroku 418, 422
HTML 132, 426, 456, 574
HTML-каркас 130, 132, 143, 449, 451, 471, 473, 477, 518, 590, 591
HTTP 125, 178
HTTP-метод PATCH 410

I
ipynb 126
iterrows 303

J
JavaScript 542
Jinja2 389
JSON 262, 393

K
KDE 377

M
Matplotlib 315, 322, 327, 430
MethodView 411
MongoDB 119, 269

SKLADCHIK.ORG

mouseenter 568, 572, 573, 574, 575, 577
mousein 575
mouseout 575

N
NaT 304
Nominatim 436
NumPy 243, 250

O
ObjectId 119
onData Change 581
onDataChange 488, 543
opacity 567
openpyx 265
option 592
Organisation for Economic Cooperation and Development, OECD 179

P
PairGrid 343
pandas 257, 291
pandas Series 272
pandas предоставляет специальный метод isnul 290
pandas тип данных object 350
parse_cols 267
parseInt 504
path 570
Plotly 430, 431, 438, 454, 456, 459
Plotly.js 449
polyfit 335
POST 178, 409
pyplot 316
pyplot и объектно-ориентированная библиотека Matplotlib 314
Python 3 365

R
random 297
remove 519
RESTful 417
RESTful Data с помощью Flask 400

S
scaleSqrt 567
Scrapy 225, 229, 233, 238
seaborn 336
select 499
selectAll 499
Series 353
SOAP 177
Sources 142
SPA 126
spread 453
SQL 267
SQLAlchemy 267, 268, 400, 403, 404, 406, 407, 411, 414
SVG 156, 162, 165

T
TopoJSON 554
Tota 443

U
unstack 356
updateList 582
updateMap 566

V
Visual Studio Code (VSCode) 128

W
WebGL 154
Wikidata 221, 222

X
xlrd 265
XML 177
XML-RPC 177
XPath 207, 214
xpath-селекторы 212

Y
ython 122

А
Адаптация к размерам экрана 429
Адаптивная передискретизация 559
Анализ данных с помощью pandas 347
атрибут d 161

Б
Базовая страница с плейсхолдерами 142
Безье 163
библиотека-обертка 183

Библиотека D3-geo, проекции и пути 556
Библиотека pandas 288
библиотека Requests 174
Блок мини-биографии с фотографией 467
Блочная диаграмма 374
Браузер с инструментами разработчика 130

В

Введение в D3 на примере столбчатой диаграммы 498
Введение в NumPy 243
Визуализации машинного обучения 606
визуализации OpenGL 136
Визуализация данных из социальных медиа 605
Визуализация данных отдельных лауреатов 579
Визуализация данных с помощью 423
Визуализация данных с помощью D3 и Plotly 604
Визуализация данных с помощью Matplotlib 314
Визуализация отдельных премий 535
Вкладка Elements 140
Вложенные данные 540
Возраст и ожидаемая продолжительность жизни лауреатов 373
Возраст на момент получения премии 373
Выбор визуальных элементов 461
Выбор групп 259

Г

Гендерные диспропорции 352
Генерация запросов 225
горизонтальными столбцами 330
Готовая карта 572
Готовность к работе 488
Графические объекты Plotly 434
группы countries 567

Д

Данные 139
Движок JavaScript 481
Диаграмма рассеивания 332
Динамическое обновление данных с помощью Flask API 396
Добавление выпадающего списка полов 594
Добавление выпадающего списка стран 595
Добавление индикаторов показателей 569
Добавление лауреатов с помощью вложенных объединений данных 543
Добавление линии регрессии 335
Добавление маршрутов RESTful API 405
Добавление пользовательских элементов управления с помощью Plotly 441
Добавление столбца born_in 307
Добавление элементов DOM 503
Документы Excel 265
Доставка данных 604
Доставка файлов данных 391
Доступ к привязанным данным 613
Доступ к web API с помощью библиотек 183
Доступные карты 550
Другие инструменты Chrome DevTools 142
другие количественные шкалы 513

З

Заголовки и метки осей 320
Заключение 602
Заключительные замечания 607
Замена строк 287
Заполнение плейсхолдеров контентом 153
запрос AJAX 393
Запуск интерактивной сессии 315
Запуск приложения для визуализации данных о нобелевских лауреатах 496
Зачем нужен скрейпинг 189
Знакомство с библиотекой pandas 254

И

Изменить pandas DataFrame 283
Из Notebook в веб-формат с помощью Plotly 444
Импорт модулей JS 483
Импорт скриптов 482
Индексация и срезы массива 248
Индексы 257
Индексы и отбор данных с помощью pandas 282
Инициализация визуализации Нобелевской премии 487
Инструменты для работы с RESTful 400
Интерактивная визуализация Plotly с помощью JavaScript и HTML 454

Использование динамической или статической доставки 398
Использование изображений и активов удаленно 430
Использование API с помощью JavaScript 420
Использование D3 510
Использование Google Таблиц 183
Использование Python для получения данных через web API 177
Использование RESTful web API с помощью Requests 178
Использование Twitter API с помощью Tweepy 186
Исторические тренды 356
Исторические тренды в распределении премии 365

К

Карта, показывающая выборку стран нобелевских лауреатов 464
Картографирование с помощью D3 550
Кеширование веб-страниц 198
Кеширование страниц 224
Классификация данных и измерения 255
код состояни 175
Количественные шкалы 511
Количество лауреатов на душу населения 361
Конвейеры Scrapy 229
Конфигурирование Matplotlib 318
Координатные сетки 561

Л

Линии, прямоугольники и многоугольники 158
логическая маска 285, 286

М

Массив NumPy 244
Масштабирование и вращение 164
Масштабируемая векторная графика 155
международный стандарт ISO 8601 122
Метки и легенды 319
Метки номинаций 538
методы проекций 559
Методы head 280
метод d3.rollup 452
Метод enter 609
метод groupby 292
метод Panel to_excel 267
Метод path.centroid 570
метод rangeRoundBands 536
метод reindex 297
метод render_template 389
метод scaleBand 514
метод sort 293
меш 556
минимум для стартового шаблона 132
Мифы об IDE, фреймворках и инструментах 128
Модуль Express библиотеки Plotly 432

Н

Надежный текстовый редактор 129
Настройка инструментов 126
Настройка конвейеров для каждого из пауков 238
настройка отступов 129
Настройка размера фигуры 318
Начало исследования 348
некоторые методы массивов 250
Нобелевская диаспора 380

О

Обнаружение смешанных типов данных 287
Обновление карты 565
Обновление DOM при изменении данных 516
объединение данных 543
объекте margin 505
Одностраничные приложения 126
однострочный сервер 393
Организация файлов 471
Оси 537
Оси и метки 523, 538
Оси и подграфики 323
оси X 514
Основной код 485
Основной поток данных 484
Основные диаграммы 431
Основы веб-разработки 125
Отправка данных через API 409
Очистка данных 286
Очистка данных с помощью pandas 276
Очистка и исследование данных с помощью pandas 603

П

Пагинация возвращаемых данных 414
паттерн Plotly 441
Первый паук Scrapy 214
Передача данных 472
Передача данных с помощью Flask 388
Передача данных через HTTP 131
Перенос диаграмм в интернет с помощью 425
Переходы 529
Повышение продолжительности жизни с течением времени 379
Подготовка данных 603
Подключение радиокнопок 599
Позиционирование и изменение размера контейнеров с помощью Flex 146
Поиск дубликатов 290
Поиск при помощи относительного Xpath 212
поле первичного ключа 401
Поле favorited 256
Полная функция для очистки данных 306
Получение данных из интернета с помощью библиотеки Requests 173
Получение данных о странах для визуализации нобелевских лауреатов 181
Получение объекта BeautifulSoup 191
Получение размеров элемента 504
Получение файлов данных с помощью Requests 174
Порядковые шкалы 514
Постановка целей 205
Построение графиков с помощью pandas 350
Построение диаграмм с помощью Plotly 430
Представления и копии 295
Премии по номинациям 363
Преобразование карт в формат TopoJSON 555
Привязка данных к элементам DOM: главное преимущество D3 516
Применение стилей CSS 157
Проверка качества данных 278
Проекции 558
Пункты 319
Пути 161, 560

Р

Работа с выборкой 499
Работа с группами 165
Работа с датами, временем и сложными типами данных 122
Работа с датами и временем 302
Работа с регулярными выражениями 218
Работа с XPath в Scrapy 207
Разметка контента 134
Распределение премии по годам 463
Расчет скользящей средней 252
Расширение API с помощью MethodView 411
Решение проблемы недостающих полей 299

С

Сборка столбчатой диаграммы 520
свойства opacity 158
свойство opacity 166
Сериализация с помощью marshmallow 404
Скрейпинг 189
Скрейпинг\
    первая попытка 190
Скрейпинг биографических страниц лауреатов 221
Скрейпинг данных 189
Скрейпинг национальности лауреатов 199
Скрейпинг текста и изображений с помощью конвейера 231
Слияние нескольких DataFrame 310
сложенные столбики 331
Соединение элементов воедино 562
Создание базы данных 401
Создание биографического блока 584
Создание веб-страницы 130
Создание визуализации 470
Создание выпадающего списка номинаций 592
Создание интерактивных графиков с помощью глобального состояния pyplot 316
Создание и сохранение структур DataFrame 261
Создание карт с помощью Plotly 435
Создание массивов 246
Создание нативных JavaScript-диаграмм с помощью Plotly 448
Создание слоев и прозрачность 166

Создание списка лауреатов 580
Создание статических диаграмм с помощью Matplotlib 425
Создание строки меню 590
Создание фильтра 492
Создание функций для работы с массивами 251
Создание шаблонов для отбора 194
Создание DataFrame из Series 272
Создание HTML-элементов с помощью D3 590
Сортировка данных 293
Сохранение очищенных наборов данных 312
Статические страницы 392
Стили CSS 477
Столбчатая диаграмма 465, 526, 329, 327
Строка меню 462
Строки и столбцы 258
строчные элементы 135

Т

Текст 159
Терминал или командная строка 130
Тестирование XPath в Scrapy Shell 208
Типы графиков 327

У

Удаление дубликатов 295
Удаление строк 289
Удаленное развертывание API на Heroku 418
упорядоченные списки 206
Установка Scrapy 204

Ф

Файлы Excel 265
Фигуры и объектно-ориентированная Matplotlib 322
Фильтрация данных с помощью Crossfilter 489, 492, 596
Форматы данных для картографирования в D3 551
Формулирование задачи 499
Функция ready 488

Ц

Цветовые карты в Matplotlib 334
Цепочка запросов и извлечение данных 224

Ш

шейп-файл 550
Шкалы 536
Шкалы в D3: от данных к их визуальному представлению 510

Э

Элемент <g> 155
Эффективный скрейпинг с помощью Scrapy 203

Барлық құқықтар қорғалған. Бұл кітапты басып шығарушының рұқсатынсыз онлайн немесе кез келген басқа жолмен сканерлеу, жүктеп алу немесе зарсыз тарату заң бойынша жазаланады / Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме с помощью каких-либо электронных или механических средств, включая изготовление фотокопий, аудиозапись, репродукцию или любой иной способ, или систем поиска и хранения информации без письменного разрешения издателя.

12+

Ағылшын тілінен аударма / Перевод с английского
«Астана иностранная пресса»

O'REILLY. КНИГИ ПО ПРОГРАММИРОВАНИЮ

Киран Дейл

ВИЗУАЛИЗАЦИЯ ДАННЫХ С ПОМОЩЬЮ PYTHON И JAVASCRIPT.
АНАЛИЗ И ПРЕОБРАЗОВАНИЕ ДАННЫХ

Бас редактор / Главный редактор
Ысқақбай Рабиға

Басуға 14.11.2025 қол қойылды / Подписано в печать 14.11.2025.
Пішіні 70×100 $^{1}/_{16}$. Қағазы офсеттік. Офсеттік басылыс /
Формат 70×100 $^{1}/_{16}$. Бумага офсетная. Печать офсетная.
Шартты баспа табағы 50,56. Таралымы 1000 дана. Тапсырыс № М-2653.
Усл. печ. л. 50,56. Тираж 1000 экз. Заказ № М-2653.

Тауар белгісі / Товарный знак: «Astana International Publishing»
Сапасы жөнінде мына мекемеге хабарласыңыз / С претензиями по качеству обращаться:
«Астана иностранная пресса» ЖШС
ҚР, Алматы қ., Аль-Фараби даңғ., 19/1, «Нұрлы-Тау» БО, 3Б блогы, 12 офис.
ТОО «Астана иностранная пресса»
РК, г. Алматы, пр. Аль-Фараби, 19/1, БЦ, «Нурлы-Тау», блок 3Б, помещение 12.

Телефон: +7 (771) 276-41-09 E-mail: info@astanapublishing.kz
Көтерме сауда бөлімі / Отдел оптовых продаж
Алматы қ., Аль-Фараби даңғ., 19/1, «Нұрлы-Тау» БО, 3Б блогы, 12 офис.
г. Алматы, пр. Аль-Фараби, 19/1, БЦ «Нурлы-Тау», блок 3Б, помещение 12.
E-mail: info@astanapublishing.kz

Тауар барлық сапа және қауіпсіздік стандарттарына сай /
Товар соответствует всем стандартам качества и безопасности
Сертификация қарастырылмаған / Сертификация не предусмотрена
Сақтау мерзімі шектелмеген / Срок годности не ограничен
Өндіруші ел: Қазақстан / Страна-изготовитель: Казахстан
Қазақстан Республикасындағы дистрибьютор: «РДЦ-Алматы»ЖШС /
Дистрибьютор в Республике Казахстан: ТОО «РДЦ-Алматы»
Алматы қ., Домбровский көшесі, 3«а», литер Б, офис 1. Тел.: 8 (727) 251-59-90/91/92
г. Алматы, ул. Домбровского, 3«а», литер Б, офис 1. Тел.: 8 (727) 251-59-90/91/92

Қазастан Республикасындағы Дистрибьютор интернет-дүкені /
Интернет-магазин дистрибьютора в РК: www.book24.kz

«Татмедиа «ПИК» Идел-Пресс « АҚ филиалы баспасында басылды
420066, Татарстан Респ, Қазан қ, Декабристов к-сі, № 2 үй.
Отпечатано в типографии Филиал АО «ТАТМЕДИА» ПИК «Идел-Пресс»
420066, Татарстан Респ, Казань г, Декабристов ул, дом № 2.

ISBN 978-601-12-4680-4



astana international publishing

O'REILLY®

Визуализация данных с помощью Python и JavaScript

Полный цикл: от сырых данных — к интерактивному представлению в браузере.

Эта книга — практическое руководство по созданию цепочки инструментов для веб-визуализации данных. Используя Python и JavaScript, вы научитесь собирать, очищать, анализировать и превращать данные в динамичные визуализации, доступные в браузере, — локально или через интернет.

Вы узнаете, как:

- Получить необходимые данные из открытых источников с помощью веб-API: Requests, Scrapy, Beautiful Soup
- Очистить и обработать данные с использованием мощных библиотек Python в экосистеме NumPy
- Доставить информацию в браузер, задействуя Flask и RESTful API
- Приобрести стойкие навыки веб-разработки (HTML, CSS, JS), чтобы разместить диаграммы и графики в интернете
- Уверенно работать в веб-среде, представляя информацию просто и наглядно

Если вы только начинаете знакомство с визуализацией данных, эта книга станет вашим надежным наставником. Она предлагает пошаговое практическое руководство по базовым инструментам Python и JavaScript, которое поможет уже за несколько недель освоить основные библиотеки Matplotlib, Seaborn и Chart.js и создавать красивые графики и анимированные диаграммы. Если нужно быстро и эффективно войти в мир визуализации данных — это лучший выбор.

Давид Салтхуцишвили,
директор IT-продуктов
студии визуализации данных
AkademiaDev

Киран Дейл — специалист по визуализации данных, разработчик и энтузиаст в области машинного обучения. Работает над проектами, объединяющими современные технологии и обработку данных, сочетая инженерный подход с визуальной выразительностью.

ISBN 978-601-12-4680-4



astana
international
publishing