Михаил Шардин

Excel Python API



Автоматизация данных и управление офисом, домом, финансами...

Таблицы, которые обновляются сами: погода, курсы валют, биржевые акции и котировки, автозаполнение данных и скрытые API

ChatGPT как помощник: генерация скриптов с нуля, поиск ошибок, автоответы и автонапоминания в почте и мессенджерах

Умный дом только под вашим контролем: распознавание лиц, управление техникой и освещением, свой голосовой ассистент и Home Assistant



+ виртуальный диск с готовыми скриптами и примерами

НиТ
издательство

Михаил Шардин

Excel, Python и API

Автоматизация данных и управление офисом, домом, финансами…

+ виртуальный диск с готовыми скриптами и примерами

"Издательство Наука и Техника"

Санкт-Петербург

УДК 004.42
ББК 32.973

Шардин М. В.

Excel, Python и API: автоматизация данных и управление офисом, домом, финансами... + *виртуальный диск с готовыми скриптами и примерами* — СПб.: Издательство Наука и Техника, 2026 г. — 432 с., ил.

ISBN 978-5-6049658-5-6

Эта книга не просто абстрактная документация по изучению Excel, Python или API, и программирование в ней выступает не как профессия, а как бытовой навык, такой как умение пользоваться таблицами или Интернетом.

Книга объясняет, как совмещать вместе код, таблицы и нейросети так, чтобы они усиливали работу друг друга: генерировали сотни документов за пару минут, создавали автоматические отчёты, делали Вам умный дом без облака, проводили анализ личных расходов, мониторили цены на инвестиции и многое другое.

На конкретных примерах будет показано, какие именно повседневные задачи (например, создание однотипных отчётов, ручной поиск данных, бесконечные напоминания и т.п.) имеет смысл автоматизировать и как это делать с помощью уже знакомых инструментов: Excel'я или Google Таблиц, при этом дополняя их простыми скриптами на Python и небольшой работой с API.

Главной особенностью книги является её прикладной характер, поэтому все примеры, которые приведены в книге, взяты из реальной жизни – это офисная работа, домашние задачи, сложные семейные процессы, а также хобби и финансы. В книге нет обобщённых задач – они все конкретные, их можно взять из книги и адаптировать для себя.

Книга не требует специальных навыков, предназначена для широкой аудитории и будет полезна как офисным сотрудникам, бухгалтерам, программистам (начинающим и не только), так и всем тем, кто хочет заставить компьютер работать на себя.

Для удобства читателей, к книге прилагается виртуальный диск с готовыми скриптами и примерами из книги, это позволит оперативно использовать всю техническую часть книги (код, скрипты, таблицы и т.д.) для своих целей и задач.



Контактные телефоны издательства:
(812) 412 70 26

Официальный сайт: www.nit.com.ru

© Шардин М. В.
© Издательство Наука и Техника

Содержание

Введение.....15

В.1. Проблема цифровой неэффективности.....17

В.2. Что такое "бытовая автоматизация"?.....18

В.3. Три кита автоматизации.....19

Excel/Google Таблицы — ваш знакомый фундамент.....20

Python — дружелюбный швейцарский нож.....20

API — универсальный пульт от всего в Интернете.....20

Синергия: как работает связка троицы.....21

В.4. Для кого эта книга.....21

Если вы бухгалтер и устали от ВПР (функция Excel для поиска и подтягивания данных из другой таблицы), ручного свода и вечных правок.....21

Если вы офисный работник с набором повторяющихся задач.....22

Если вы энтузиаст и хотите, чтобы кофеварка включалась, когда вы снимаете телефон с зарядки.....22

Если вы начинающий разработчик и ищете идеи для пет-проектов.22

Кому эта книга будет малоинтересна.....22

В.5. Дорожная карта: что ждет вас впереди.....23

В.5.1. Сначала мы наведем порядок на работе и вернем вам десятки часов рабочего времени.....23

В.5.2. Затем перенесем эти навыки домой, сделав быт комфортнее, а хобби — интереснее.....23

В.5.3. Когда вы освоитесь, мы поднимем планку и научимся применять автоматизацию для управления личными финансами и тестирования идей.....24

В.5.4. Бонус. А в конце мы заглянем в будущее и покажем, как использовать искусственный интеллект в качестве вашего личного ассистента-программиста.....24

В.5.5. Как читать книгу, чтобы получить максимум.....24

В.6. Быстрый старт: ваша первая автоматизация за 30 минут (без программирования).....24

В.6.1. Способ 1: магия формул (для тех, кто любит копировать и вставлять) 25
В.6.2. Способ 2: визуальный конструктор (для тех, кто доверяет кнопкам) 28

ГЛАВА 1. **Excel и Google Таблицы на стероидах: заставляем данные работать на нас**....37

Анатомия "живой" таблицы 38

1.1. Магия формул. Быстрые данные без кода 39

1.1.1. Google Таблицы: погода в реальном времени с IMPORTDATA . 39

Почему выбран именно ***api.open-meteo.com***?....40
Формула и ее подробный разбор 40
Почему пример в Google Таблицах, а не в Excel? 41
Как адаптировать под ваш город?....41

1.1.2. Excel: котировки акций с WEBSERVICE + FILTER.XML 41

1.2. Визуальный конструктор. Собираем данные с Power Query в Excel....44

Практика по шагам 44
1.2.1. Подключение: создаем три запроса к трем CSV 44
1.2.2. Трансформация: приводим столбцы к единому виду....45
1.2.3. Объединение: соединяем три очищенных запроса....47
1.2.4. Загрузка и обновление....48

1.3. Пишем скрипты. Когда формул и конструкторов недостаточно 49

1.3.1. Скрапинг в Excel с VBA: парсим Википедию 50
1.3.2. Облачная автоматизация в Google Таблицах с Apps Script: создаем ассистента по поиску цен 58

Код и его анатомия 60
Как это работает 62
Как заставить это работать у себя? 63
Важные нюансы и возможные улучшения....64

Заключение 65

ГЛАВА 2. **Магия одного клика: работа с документами и файлами**..67

2.1. Word — ручной станок для документов 68

2.1.1. Простой пример: генерация именных сертификатов для участников кулинарных курсов 69

Источник данных 69

Шаблон 70

2.1.2. Процесс: пошаговая инструкция 71

2.1.3. Практические хитрости 78

2.2. Облачная фабрика документов с Google Apps Script 79

2.2.1. Пример: автоматизация договоров дарения 80

Разбор решения 80

2.2.2. Apps Script — "швейцарский нож" для автоматизации в Google .. 89

2.3. Python — универсальный инструмент инженера 90

2.3.1. Простой пример: скрипт-сортировщик на Python 90

Где особенно полезен Python? 90

Проблема: цифровая свалка в папке "Загрузки" 90

Решение: "цифровой дворецкий" на Python 91

Код: 20 строк, которые сэкономят часы 91

Разбор полетов: как это работает? 93

Вывод: от простого скрипта к целой системе автоматизации .. 94

2.3.2. Сложный пример: сбор сводного отчета из сотен Excel-файлов 95

Идея: программный рейд по файлам 95

Почему именно VBA? 96

Разбор скрипта: как заставить Excel работать на себя 97

Результат: реестр за минуту 101

Вывод: универсальный паттерн пакетной обработки 101

Заключение 102

ГЛАВА 3. **Ваш личный секретарь: автоматизируем почту и мессенджеры** 103

От курьера к диспетчеру: автоматизация коммуникаций ... 104

Арсенал цифрового коммуникатора 105

3.1. Почтовый автомат без кода 106

Gmail: фильтры и шаблоны как первая линия обороны 106

3.2. Ваш личный почтовый робот на Google Apps Script 108

Шаг 1. Создаем скрипт 110

Шаг 2. Разбираем код 111

Шаг 3. Настраиваем и запускаем 111

Искусство автоответа 112

Подводные камни и зачем это вообще нужно 112

3.3. Диалог с миром (мессенджеры) 113

Отложенная отправка в мессенджерах: цифровой этикет и здравый смысл 113

Два пути автоматизации мессенджеров 114

3.3.1. Пример: оповещение автовладельцев через WhatsApp 115

Постановка задачи 116

Техническая реализация 117

3.3.2. Думаем как IT-специалист: безопасность и риски 121

Риск блокировки аккаунта 121

Работа с персональными данными 122

Техническая надежность 122

3.3.3. Пример 2: автоматизация личной переписки в Telegram 122

Постановка задачи 122

Техническая реализация 123

Полный код скрипта 124

Этика автоматизации 127

3.4. Управление репутацией (соцсети) 129

Поддержание социальной активности для "лентяев" в ВКонтакте 129

Думаем о последствиях: репутационные риски слепой автоматизации 134

Заключение: ваш цифровой секретарь готов к работе 136

ГЛАВА 4. **Умный дом, который работает на вас, а не на корпорации** 139

4.1. Бунт против облака: почему ваш умный дом должен принадлежать вам..141

4.1.1. Решение — локальный умный дом: почему это работает ...142

4.1.2. Знакомство с "мозгом" — Home Assistant как практический выбор....143

4.1.3. Практический чек-лист для перехода 144

4.2. Практикум: разворачиваем "мозг" за 30 минут....144

4.2.1. Железо для "мозга": выбираем платформу....144

4.2.2. Пошаговая инструкция: от коробки до рабочего сервера 145

Шаг 1. Что купить: минимальный набор 145

Шаг 2. Как установить Home Assistant OS: проще не бывает.. 145

Шаг 3. Первый запуск: магия начинается 146

4.2.3. Результат: ваш цифровой суверенитет обрел форму 148

4.3. Первые "победы": автоматизация, которую можно потрогать 148

4.3.1. Пример первый: два контура освещения в квартире без ремонта 148

Решение: одна умная лампа вместо новой проводки 149

Практическая настройка в Home Assistant 150

Результат: экономия времени, денег и нервов....151

4.3.2. Пример второй: автоматические рулонные шторы без проводки 220В 151

Решение: аккумуляторные приводы 152

Интеграция с Home Assistant....153

Честный разговор о недостатках....154

4.4. "Разблокируем" бытовую технику: возвращаем умные функции 155

4.4.1. История одной "мертвой" стиральной машины 155

4.4.2. Практикум: возвращаем Bosch к жизни 156

Шаг 1. Обходим географические ограничения....156

Шаг 2. Подключаем технику к Wi-Fi....157

Шаг 3. Регистрируемся в системе разработчиков 157

Шаг 4. Подключаем к Home Assistant....157

Честный разговор об ограничениях 158

4.5. Продвинутый уровень: ваш личный швейцар и охранник....159

4.5.1. Создаем пульт для всей семьи: настенная панель управления 160

Решение: "освобождаем" заводское устройство 160

4.6. Анализ лиц с домофонной камеры 162

4.6.1. Почему Python? 163

4.6.2. Практикум: от видеоархива к отчету 163

Подготовка окружения: обходим "подводные камни" 163

Анализ видео: разбираем скрипт ***face_report.py*** 165

Поиск лиц на кадре 165

Превращение лица в "цифровой отпечаток" 173

Сравнение и уникализация 173

Генерация HTML-отчета 173

Результат: кто ходит у вашей двери? 174

4.7. Финальный штрих: избавляемся от Яндекс-Алисы и создаем приватного ассистента 175

4.7.1. Почему пора расстаться с облачными ассистентами 175

4.7.2. Анатомия локального ассистента: как это работает изнутри ... 175

4.7.3. Практикум: два пути к голосовой независимости 177

Вариант 1. "Быстрый старт" — все на Raspberry Pi 177

Вариант 2. "Правильная архитектура" — выносим обработку на отдельный сервер 180

Интеграция с Home Assistant 182

Настройка "спутников" 183

Практические советы по настройке 184

Заключение 184

ГЛАВА 5. **Цифровые лайфхаки для семьи и хобби** 187

5.1. Простой старт: решаем бытовые мелочи за 5 минут 188

Шаг 1. Создаем основу в Google Таблицах 188

Шаг 2. Подключаем Google Apps Script 190

Шаг 3. Проверяем и используем 191

Шаг 4. Почему это важно 191

5.2. Скрипт для напоминания о днях рождения с подсчетом возраста 192

Решение: "умные" события в календаре с помощью Google Apps Script 192

5.3. Продвинутый уровень: автоматизация для сложных жизненных сценариев 205

5.4. Скрипт для координации жизни ребенка после развода 206

Решение: умная фильтрация событий с помощью Google Apps Script 207
Шаг 1. Создаем структуру календарей 208
Шаг 2. Создаем маркерное событие 208
Шаг 3. Пишем и настраиваем скрипт 209
Шаг 4. Настраиваем автоматический запуск 212
Почему это работает и зачем это нужно 212

5.5. Автоматизируем хобби и укрощаем цифровой быт 213

Решение: единая таблица всех соревнований с автоматическим обновлением 213
Шаг 1. Создаем структуру таблицы 214
Шаг 2. Подключаем библиотеку Cheerio 215
Шаг 3. Пишем функции парсинга 215
Шаг 4. Собираем все воедино 220
Шаг 5. Настраиваем автоматический запуск 220
Почему это важно 220

5.6. Когда менять шины? Python и API погоды для автовладельца 221

Шаг 1. Находим координаты своего города 222
Шаг 2. Пишем код 222
Шаг 3. Интерпретируем результат 224
Шаг 4. Расширяем идею — уведомления в Telegram 225

5.7. Сканируем в домашнем офисе: превращаем односторонний сканер в двусторонний 226

5.7.1. Часть 1: магия двустороннего сканирования 227

Шаг 1. Сканируем лицевые стороны 227
Шаг 2. Переворачиваем всю стопку целиком 227
Шаг 3. Сканируем оборотные стороны 227
Знакомимся с pdftk 228
Шаг 4. Склеиваем два файла в один правильный документ 228

5.7.2. Часть 2: автоматическая уборка пустых страниц 230

Проблема обнаружения пустоты 231
Пишем скрипт для очистки 231

Запускаем скрипт....233

Заключение....234

ГЛАВА 6. **Персональный аналитик: управляем личными данными** 237

6.1. Фундамент анализа: ваш личный дата-центр в Google Таблицах 239

Практикум: магия функции QUERY....242
Задача: собрать единую картину 242
Решение: объединяем листы 242
Разбираем формулу шаг за шагом 244
Следующий шаг: баланс всех карт 244

6.2. Практикум: анализируем свои траты с помощью ИИ....246

Идея: пусть ИИ анализирует, а мы принимаем решения 247
Шаг 1. Подготовка данных 247
Шаг 2. Общий финансовый чекап 247
Шаг 3. Расследование аномалий 249
Шаг 4. Глубокое погружение в категорию....250
Итого: ИИ как надстройка над таблицей....251
Схема: Google Таблица => ИИ-отчет => Решение....251

6.3. За пределами финансов: другие идеи для персональной аналитики 252

Личный музей: учитываем коллекции и увлечения 252
Аналитика тела: визуализируем спорт и привычки 253
Когда данные работают сами: отслеживаем цены с помощью Python 254

Заключение 255

ГЛАВА 7. **Автоматический мониторинг инвестиционного портфеля** 257

7.1. Создаем универсальную таблицу для учета всех активов 258

Шаг 1. Структура данных – наш скелет....259

Шаг 2. Формулы, которые считают за нас. 260
Промежуточный результат 262

7.2. Уровень 1: автоматический сбор котировок с помощью встроенных формул 262

GOOGLEFINANCE – самый простой способ для мировых рынков .263
IMPORTXML – решение для Московской биржи 264
Как менять URL для других типов активов с Московской биржи ...265
Парсинг сайтов экзотики – монеты, недвижимость и др. 266
Когда простых формул недостаточно – Google Apps Script 267

7.3. Уровень 2: локальный прокси-сервер: получаем данные, которых нет в открытом доступе 269

Давайте разберем, как это работает, шаг за шагом 270
Создаем сервер (на примере Тинькофф Инвестиций) 271
Тестирование: проверяем, что всё работает 275
Безопасность и эксплуатация 275

7.4. Уровень 3: автоматизируем поиск и анализ облигаций 276

Автоматический расчёт денежного потока 277
Python-скрипт собирает календарь за минуты 277
Автоматизация поиска выгодных облигаций. Задача: найти иголку в стоге сена 283
Шаг 1. Определяем критерии отбора. 284
Шаг 2. Скрипт собирает данные с биржи 286
Шаг 3. Применяем фильтры последовательно 287
Шаг 4. Сохраняем результат. 288

Заключение главы: от ручного учета к системе поддержки решений 289

ГЛАВА 8. **Взламываем повседневность: подключаемся к API любимых сервисов** 291

8.1. API как секретный вход в любой сервис 292

8.2. API в действии: автоматизируем расписание 295

Шаг 1. Расследование: ищем "скрытый" API. 295
Шаг 2. Разбор полётов: что значат эти поля. 296

Шаг 3. Пишем решение: Google Apps Script в роли API-клиента ...297
Шаг 4. Проверяем и настраиваем автоматический запуск.....300
Этика и границы допустимого 301

8.3. API для данных: создаем единый центр для личной аналитики 301

Информационная раздробленность 302
Парсинг HTML на примере сайта Хабр.....303
Работа со "скрытым" API.....304
Строим свою единую панель данных 305
Данные работают на нас.....311

8.4. API для контроля: наводим порядок в своих цифровых сообществах 311

Проблема: цифровой мусор и его последствия.....312
Безопасность прежде всего 313
Скрипт рабочего примера на Telethon 313
Разъяснения по коду и ограничения 326
Отсылка к главе 3 – репутационные риски.....326

Резюме главы: Вы – дирижер цифрового оркестра 327

ГЛАВА 9. **Первый шаг к алгоритмам: тестируем инвестиционную идею**.....329

9.1. Практикум №1: "Эскиз на салфетке". Тестируем идею за 15 минут в TradingView.....332

Реализация на Pine Script.....334
Анализ результатов 340
Результаты на дневном графике 340
Результаты на часовом графике 340

9.2. Практикум №2. Глубокий анализ в Python.....342

Зачем нам Python, если есть TradingView? 342
Подготовка к тесту: ищем, на чем торговать 343
Скрипт: находим самые ликвидные акции по кварталам за последние 20 лет.....345

9.3. Бэктестинг на Python 358

Почему стоит выбрать Backtrader? 358

Архитектура нашего "Народного шаблона" 359
Шаг 1. Учим Python читать наши данные. 360
Шаг 2. Фундамент для стратегий....361
Шаг 3. Реализация стратегии пересечения средних. 364
Шаг 4. Анализ и отчетность. 373
Шаг 5. Главный пульт управления....382
Запуск теста и анализ результатов 390
Анализ результатов 391

9.4. Заглядывая за горизонт – можно ли научить ИИ читать графики? 392

Революционная идея: от цифр к словам 393
Шаг 1. Данные....393
Шаг 2. Генерация признаков. 393
Шаг 3. Перевод на "человеческий" язык. 394
Шаг 4. Обучение модели. 394
Результаты: холодный душ или новая надежда?....395
Главный вывод раздела 397

Заключение: вы сделали первый шаг от инвестора к исследователю 398

ГЛАВА 10. **ChatGPT как ваш личный программист**....401

10.1. От "я пишу код" к "я управляю написанием кода" 403

Проблема чистого листа 403
Простой пример: "я управляю", а не "я вспоминаю синтаксис" 404

10.2. Искусство промпт-инжиниринга: как ставить ТЗ, чтобы не переделывать 406

Формула идеального промпта 406
Разбираем на примере 408
Итеративное улучшение: диалог, а не монолог 410

10.3. Вайбкодинг на практике: пишем скрипт с нуля за 10 минут....410

Практикум: создаём утилиту для чистки рабочего стола от скриншотов....410

10.4. Отладка и объяснение: если код не работает 412

10.5. Безопасность и галлюцинации: где ИИ врет, а где опасен 417

Галлюцинации: когда ИИ выдумывает то, чего не существует 418

Устаревшие данные: модели помнят прошлое, но не знают будущего 419

Мы всегда проверяем код на своей версии библиотек 419

Заключение 420

Список использованных источников информации 423

Введение

Каждое утро понедельника человек приходит в офис с легким комом в груди, но не от страха перед работой, а от повторения знакомой до боли процедуры. Его ритуал очень прост: он открывает 15 Excel-файлов, присланных коллегами. Из каждого файла вытаскивает по три строки — номер, дату и пару показателей, затем сводит все это в один файл. Проверяет форматирование, чтобы таблица не убежала, сохраняет результат в PDF. Отправляет этот отчет по трем адресам. В сумме полтора часа механического труда каждую неделю.

Работник знает, что делает это хорошо: аккуратно, без ошибок и быстрее большинства. Но это не приносит ни радости, ни профессиональной гордости. Лишь растущее чувство тихого, но постоянного недовольства: сожаление о потерянном времени, о неисполненных обещаниях "поработать над идеей, которая действительно может изменить отчетность". В голове мелькают мысли о бессмысленности усилий: зачем превращаться в личного секретаря своего компьютера, если компьютер можно научить работать самому?

Мы тратим часы на операции, которые встречаются с завидным постоянством, и при этом лелеем идею, что "когда-нибудь" займемся чем-то более важным. Между тем технологии уже умеют брать на себя рутину — остается научиться правильно просить их об этом.

Эта книга — инструкция по бунту против цифровой рутины. В следующих главах вы не найдете абстрактных рассуждений — только практические приемы: реальные скрипты, понятные алгоритмы и готовые рецепты, которые позволят сохранить время для семьи, хобби или действительно творческой работы.

В.1. Проблема цифровой неэффективности

Эта история — локальная трагедия одного рабочего утра, но проблема носит системный характер. Таких работников в офисах сотни и тысячи, а рутинных действий — десятки в неделю у каждого. Если сложить потери времени всех сотрудников компании, всех домохозяйств и всех людей в городе — получим не просто часы, а дни и недели, украденные у жизни и творчества. И это не гипотеза, а повседневная реальность — повторяющиеся однообразные операции съедают человеческий ресурс, который нельзя воспроизвести или купить обратно.

Какие же конкретно задачи чаще всего выступают в роли "пожирателей времени"? Вот краткий ориентировочный список — думаю, он знаком большинству читателей этой книги:

1. **Офис**:
   - ручное создание однотипных документов — договоров, счетов, актов, каждый раз с теми же полями и маленькими отличиями;
   - сбор данных с сайтов и порталов вручную (ввод в Excel из системы партнеров, свод отчетов от подрядчиков);
   - персонализированные рассылки и уведомления, подготовка сопроводительных писем;
   - заполнение CRM-системы: перенос контактов, обновление статусов сделок, дублирование информации между системами;
   - подготовка презентаций с типовой структурой под разных клиентов;
   - создание отчетов для руководства с одинаковой структурой каждую неделю/месяц.
2. **Дом**:
   - рутинные проверки: сравнение цен перед покупкой, мониторинг погоды для поездок, контроль счетов и подписок;
   - управление "умным" хозяйством через десяток несовместимых приложений;
   - планирование покупок: составление списков, поиск магазинов с нужными товарами;

- уплата коммунальных платежей и штрафов через разные сервисы;
- ведение семейного календаря событий и напоминаний.

3. **Хобби**:
   - мониторинг афиш и событий, сбор интервью и материалов для блогов;
   - мониторинг цен на интересующие товары (техника, билеты на концерты);
   - обновление профилей на сайтах знакомств, форумах, профессиональных сетях;
   - размещение контента в соцсетях — адаптация под форматы разных платформ.

> **Проблема не в количестве задач — проблема в том, что мы частично выполняем работу, для которой и были созданы машины.**

Мы делаем за них эти операции, вместо того чтобы научить их это делать. Самый дефицитный ресурс в XXI веке — время; тратить его на механическую рутину — все равно что отдавать деньги за то, что можно получить бесплатно. В этой книге вы научитесь отдавать рутину машинам и возвращать себе время — на идеи, на людей и на то, что делает жизнь значимой.

B.2. Что такое "бытовая автоматизация"?

Если коротко — **автоматизация бытовых и офисных задач** — это не магия и не привилегия программистов. Это набор простых приемов и инструментов, которые позволяют передать компьютеру повторяющуюся работу: сбор данных, подстановку значений в шаблоны, генерацию отчетов, отправку писем и управление устройствами. Главное — отнять у этой идеи мистику: автоматизация — это навык, которому можно научиться по шагам, а не дар, "встроенный" в мозг профессионального разработчика.

Подумайте об этом как о вождении автомобиля.

Вам не обязательно быть автомехаником, чтобы сесть за руль и доехать от **точки А** до **точки Б** с комфортом. Достаточно освоить правила, потренироваться и понять устройство машины на базовом уровне. Точно так же, освоив несколько простых приемов (формулы в таблицах, базовые скрипты, готовые API-запросы), вы сможете решать 80% рутинных задач, которые сегодня съедают ваше время.

Если вы уже умеете писать формулу `СУММ` в Excel, поздравляю: вы уже на полпути к созданию своего первого скрипта. Формула `=СУММ(A1:A10)` — это по сути мини-программа: вы даете машине описание, что нужно сложить, и получаете результат.

Скрипт на Python выполняет те же действия, только масштабируемо: берет данные из нескольких файлов, делает проверки, сохраняет и рассылает отчет — без вашего участия. Разница — в синтаксисе и инструментах, а не в принципе мышления.

Развенчаем еще один миф: "Мне нужно год учиться программированию". На практике нужен не университетский курс по алгоритмам, а набор конкретных инструментов для конкретных задач. Освойте шаблоны документов, научитесь парсить страницу или отправлять HTTP-запрос, поймите, как работают CSV и регулярные выражения — и вы сможете автоматизировать десятки повторяющихся процессов. Эта книга — не учебник по информатике, это сборник практических рецептов: короткие пошаговые решения, которые можно адаптировать и комбинировать.

И наконец, о современных способах. Появился термин "вайб-кодинг" (vibe coding), означающий подход, когда вы описываете задачу на обычном языке, а современные инструменты на базе ИИ (LLM) помогают генерировать код, шаблоны или даже целые прототипы. Вайб-кодинг отлично подходит для прототипирования и быстрого получения рабочего результата — особенно если вы не хотите вдаваться в детали. Однако важно помнить: ИИ-генерация — это помощник, а не волшебная замена знаниям, итоговый код нужно проверять и понимать.

B.3. Три кита автоматизации

Задача этого раздела — не напугать вас названиями и аббревиатурами, а показать, что все инструменты просты, понятны и лучше работают вместе.

Представьте три надежных помощника: у каждого своя специализация, но вместе они закрывают почти любые бытовые и офисные рутинные задачи. Ниже дано краткое, но емкое описание каждого "кита" и его конкретной пользы для вас.

Excel/Google Таблицы — ваш знакомый фундамент

Вы уже знакомы с этим инструментом: формулы, таблицы, фильтры — звучит привычно. Но у Excel есть "секретные кнопки", которые превращают его из простого калькулятора в центр сбора и обработки данных. Это формулы, сводные таблицы, условное форматирование, Power Query (запросы), макросы и шаблоны. С помощью них вы быстро приводите десятки файлов к единому формату, автоматически обновляете отчеты и делаете интерактивные дашборды, которые коллеги понимают с полуслова. Google Таблицы добавляют удобство совместной работы и простую интеграцию с облачными API — идеально для семейного бюджета или командного трекера задач. Excel — это удобный интерфейс и быстрый визуальный контроль того, что происходит с вашими данными.

Python — дружелюбный швейцарский нож

Если Excel — это мастерская столяра, то Python — универсальный электрический инструмент: быстро, гибко и для всего. Библиотеки вроде pandas превращают таблицы в объекты, которые можно фильтровать, объединять и анализировать за пару строк. requests умеет разговаривать с сайтами, openpyxl/pyxlsb — читать и писать Excel, а библиотеки для работы с почтой и мессенджерами — отправлять результаты подписчикам автоматически.

Python склеивает разрозненные процессы: берет данные из API, чистит их, помещает в шаблон документа и высылает итог по списку адресов. При этом можно писать скрипты, которые запускаются по расписанию, — и вы уже не контролер, а архитектор процесса.

API — универсальный пульт от всего в Интернете

API (Application Programming Interface, "программный интерфейс приложения") — это способ, с помощью которого один сервис или программа могут "разговаривать" с другими.

Проще говоря, это как кнопки на пульте: вы нажимаете — и телевизор понимает команду. Сайт или сервис предлагает программисту такой "пульт" для получения данных напрямую. Погода, курсы валют, выписки по картам, посты из соцсетей — все доступно через API, но только если у сервиса есть "служебный вход".

Обычный формат — JSON: вы отправляете запрос, получаете структурированные данные и дальше решаете, что с ними делать. Важные практические вещи: учимся работать с ключами доступа (tokens), уважать лимиты запросов и хранить секреты безопасно. Освоив базовый набор приемов, вы перестаете копировать страницы вручную — данные приходят в вашу систему автоматически.

Синергия: как работает связка троицы

Отдельно каждый инструмент полезен, но все вместе они — сила.

Пример: в понедельник утром Python забирает из API-банков выписки и данные от партнеров, чистит и складывает их в Google Таблицу, где настроен дашборд, и несколько сводных таблиц; затем тот же Python экспортирует PDF-отчет по шаблону Excel и отправляет его группе адресатов. Excel дает понятный интерфейс и отчетность, Python — логику и интеграции, API — источник живых данных.

B.4. Для кого эта книга

Если вы бухгалтер и устали от ВПР (функция Excel для поиска и подтягивания данных из другой таблицы), ручного свода и вечных правок

Вы научитесь автоматически собирать выписки и отчеты, сопоставлять транзакции с накладными, генерировать счета и акты по шаблонам и выгружать их в требуемом формате — все это без постоянного копипаста. Пара простых макросов + один Python-скрипт — и ручная работа сокращается в разы. Практические рецепты: валидация данных, защита шаблонов и безопасное хранение учетных данных.

Если вы офисный работник с набором повторяющихся задач

От переименования сотни файлов до персонализированных рассылок и сводов — здесь вы найдете готовые шаблоны и пошаговые инструкции: как превратить однотипные операции в задачу "нажать кнопку" (или запланировать по расписанию). Разберем интеграцию Excel <=> почта <=> мессенджер и покажем, как пользоваться Power Query и простыми API, не влезая в дебри.

Если вы энтузиаст и хотите, чтобы кофеварка включалась, когда вы снимаете телефон с зарядки

Для DIY-любителей — простые сценарии для умного дома: как заставить старые гаджеты "разговаривать" друг с другом через решения (IFTTT, MQTT, Home Assistant), как консолидировать управление в одном скрипте и избегать типичных ловушек безопасности. Практические проекты — от автовключения светильника до умной рассылки уведомлений.

Если вы начинающий разработчик и ищете идеи для пет-проектов

Книга даст набор реальных проектов, которые легко превратить в портфолио: от парсера цен до системы учета семейного бюджета с базовым бэктестингом инвестиционных идей. Каждая глава — это мини-проект с описанием требований, кода и критериями успеха, чтобы вы могли быстро собрать рабочую демоверсию и залить на GitHub.

Кому эта книга будет малоинтересна

Если вы — опытный backend-инженер, исследователь масштабируемых систем или DevOps-специалист, ищете архитектурные шаблоны для распределенных сервисов с миллионами пользователей — это не ваш профиль. Мы не пишем здесь кластерные решения, не строим высоконагруженные платформы и не лезем в глубины оптимизаций на уровне C/C++.

Также эта книга и не для тех, кто хочет "волшебную кнопку" без усилий: базовое понимание логики и готовность попробовать — обязательны.

В итоге: эта книга — про практику и результат. Если вы готовы потратить несколько часов на то, чтобы настроить систему один раз и получать выгоду долгие месяцы — вы попали по адресу. Если же ваша цель — изучать программирование на уровне "диплома" — вам лучше поискать учебник.

B.5. Дорожная карта: что ждет вас впереди

B.5.1. Сначала мы наведем порядок на работе и вернем вам десятки часов рабочего времени

Цель: убрать из вашей рабочей недели рутину, которая забирает энергию и время.

Эта часть — про быстрые выигрыши: готовые шаблоны, макросы и скрипты, которые можно быстро внедрить. Практикумы показывают, как автоматически собирать данные в таблицу, генерировать пачку договоров по шаблону, приводить в порядок сотни файлов и настраивать рассылки — все с минимальными навыками программирования. Вы выйдете с набором "ключей", которые позволят сократить рутинную работу в 5–10 раз.

B.5.2. Затем перенесем эти навыки домой, сделав быт комфортнее, а хобби — интереснее

Цель: использовать тот же набор инструментов для улучшения домашней жизни.

Мы покажем, как построить локальный умный дом на базе Home Assistant, как из старого смартфона сделать панель управления, как автоматизировать семейные сценарии (напоминания, учет расходов, управление техникой). Практикумы — от распознавания лиц соседей и курьеров с камеры домофона до создания скриптов для учета коллекций и управления расписанием семьи. Результат: меньше мелочей, которые отвлекают, и больше того, что действительно важно.

B.5.3. Когда вы освоитесь, мы поднимем планку и научимся применять автоматизацию для управления личными финансами и тестирования идей

Цель: перейти от удобства к пользе с точки зрения денег и риска.

Здесь вы найдете проекты по автономному сбору котировок, автоматическому мониторингу выгодных облигаций, и самое главное — простой вводный бэктестинг для проверки инвестиционных гипотез. Практикумы показывают, как подключаться к API сервисов, аккуратно хранить ключи и честно тестировать стратегии — чтобы не верить в чудеса, а понимать, где есть смысл и где — ловушка.

B.5.4. Бонус. А в конце мы заглянем в будущее и покажем, как использовать искусственный интеллект в качестве вашего личного ассистента-программиста

Цель: дать инструменты, которые ускоряют написание и проверку кода.

Вы научитесь формулировать задачи для ИИ, использовать ChatGPT для генерации прототипов, разбирать и улучшать сгенерированный код. Это не заменяет понимание, но сильно ускоряет путь от идеи до рабочего сценария — особенно при прототипировании и рутинной доработке.

B.5.5. Как читать книгу, чтобы получить максимум

Книга задумана как практическое руководство — читайте в том порядке, в котором вам удобнее: если нужен быстрый эффект в офисе — начните с первой части; если хочется "поумнеть" на дому — сразу вторая. Во всех главах есть Практикум с пошаговыми инструкциями и готовыми скриптами — копируйте, запускайте, адаптируйте.

B.6. Быстрый старт: ваша первая автоматизация за 30 минут (без программирования)

Теория — это хорошо, но ничто не убеждает лучше, чем результат, который можно потрогать руками.

Прежде чем мы погрузимся в мир Python-скриптов и API-ключей, давайте докажем главный тезис этой книги на практике: автоматизация доступна каждому, и начать можно прямо сейчас.

Вы могли ожидать, что первая задача будет про офисную рутину — таблицы сотрудников, учет отпусков или сверку накладных. Но начнем мы с курсов валют. Почему? Потому что это универсальная тема: курсы нужны бухгалтерам для пересчета операций, предпринимателям — для контрактов, а инвесторам — для оценки активов.

Создадим в Excel файл, который будет каждый день сам, без нашего участия, показывать актуальные курсы валют с официального сайта Центрального Банка РФ. Больше не нужно открывать браузер, искать сайт и копировать цифры. Вы открываете файл — и свежие данные уже там.

Мы сделаем это двумя способами. Выберите тот, что вам больше по душе.

В.6.1. Способ 1: магия формул (для тех, кто любит копировать и вставлять)

Этот метод идеален, если вы хотите быстро поделиться решением с коллегой — достаточно просто отправить ему пару формул в мессенджере. Он основан на двух встроенных функциях Excel: ВЕБСЛУЖБА (которая умеет "ходить" по ссылкам и забирать оттуда данные) и ФИЛЬТР.XML (которая помогает извлечь из полученного кода нужную информацию).

Звучит сложно? Вовсе нет. Вам нужно лишь скопировать и вставить готовые строки.

Шаг 1. Откройте новый лист Excel.

Шаг 2. Выберите три свободные ячейки одну под другой и вставьте в них следующие формулы:

В первую ячейку (для красивого заголовка):

```
="--- Официальные курсы ЦБ РФ на "&ТЕКСТ(СЕГОДНЯ();"дд.ММ.гггг")&": ---"
```

Во вторую ячейку (для получения курса юаня):

```
="Юань (CNY):
"&@ФИЛЬТР.XML(ВЕБСЛУЖБА("https://www.cbr.ru/scripts/XML_dynamic.asp?date_req1=
"&ТЕКСТ(СЕГОДНЯ()-1;"дд/ММ/гггг")&"&date_req2="&ТЕКСТ(СЕГОДНЯ()+1;"дд/
ММ/гггг")&"&VAL_NM_RQ=R01375");"//Value")
```

В третью ячейку (для получения курса доллара США):

```
="Доллар США (USD):
"&@ФИЛЬТР.XML(ВЕБСЛУЖБА("https://www.cbr.ru/scripts/XML_dynamic.asp?date_req1=
"&ТЕКСТ(СЕГОДНЯ()-1;"дд/ММ/гггг")&"&date_req2="&ТЕКСТ(СЕГОДНЯ()+1;"дд/
ММ/гггг")&"&VAL_NM_RQ=R01235");"//Value")
```

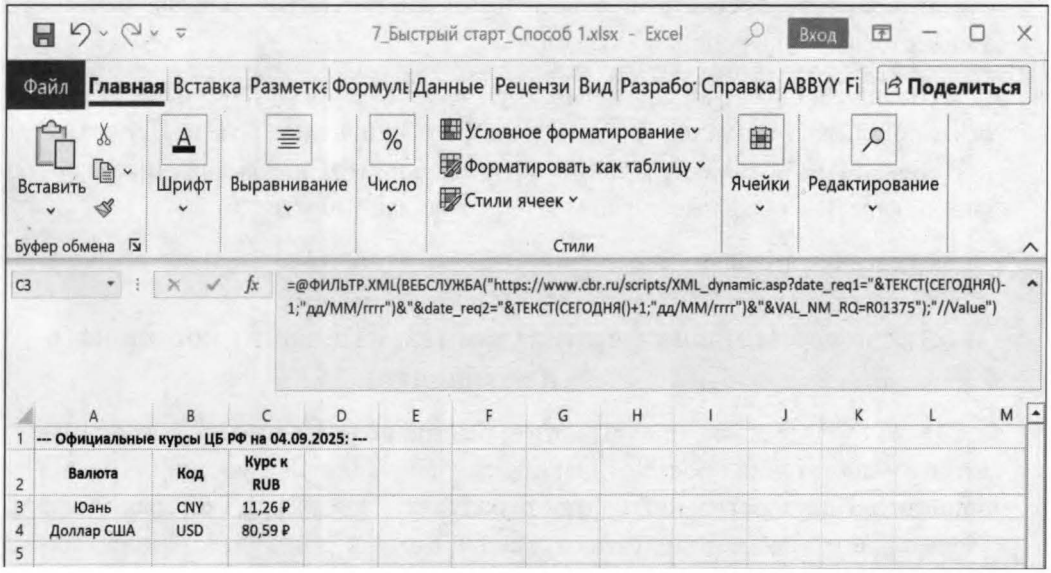


Рис. В.1. ***Подготовленный файл Excel с формулами получения курсов валют на сегодня***

Готово! Как только вы нажмете **Enter**, Excel сам сходит на сайт ЦБ, заберет данные и покажет их в ячейке (рис. В.1). Завтра вы откроете этот же файл, и дата в заголовке вместе с курсами обновится автоматически. Просто, не правда ли?

А хотите заглянуть "под капот" этой формулы и узнать, как она получает такой результат? Как будто маленький робот внутри Excel сам ходит в Интернет за курсом валют и вставляет их в таблицу.

Вот более понятная запись формулы в несколько строк (если вставить в таком виде в Excel — тоже будет работать):

```
="Юань (CNY): "&@ФИЛЬТР.XML(
  ВЕБСЛУЖБА(
    "https://www.cbr.ru/scripts/XML_dynamic.asp?date_req1="&
    ТЕКСТ(СЕГОДНЯ()-1;"дд/ММ/гггг")&
    "&date_req2="&
    ТЕКСТ(СЕГОДНЯ()+1;"дд/ММ/гггг")&
    "&VAL_NM_RQ=R01375"
  );
  "//Value"
)
```

Что делает эта формула:

1. Формирует ссылку на сайт Центробанка (ЦБ РФ).
   - » Адрес ведет на XML-сервис ЦБ, где можно получить динамику валютных курсов.
   - » В параметрах *date_req1* и *date_req2* подставляются даты — вчерашний день и завтрашний.
   - » Это нужно для того, чтобы точно попасть в "окно", где будет опубликован курс на сегодня.
   - » Код валюты R01375 — это юань (китайский CNY).

На выходе Excel получает ссылку вида:

```
https://www.cbr.ru/scripts/XML_dynamic.asp?date_req1=03/09/2025&
date_req2=05/09/2025&VAL_NM_RQ=R01375
```

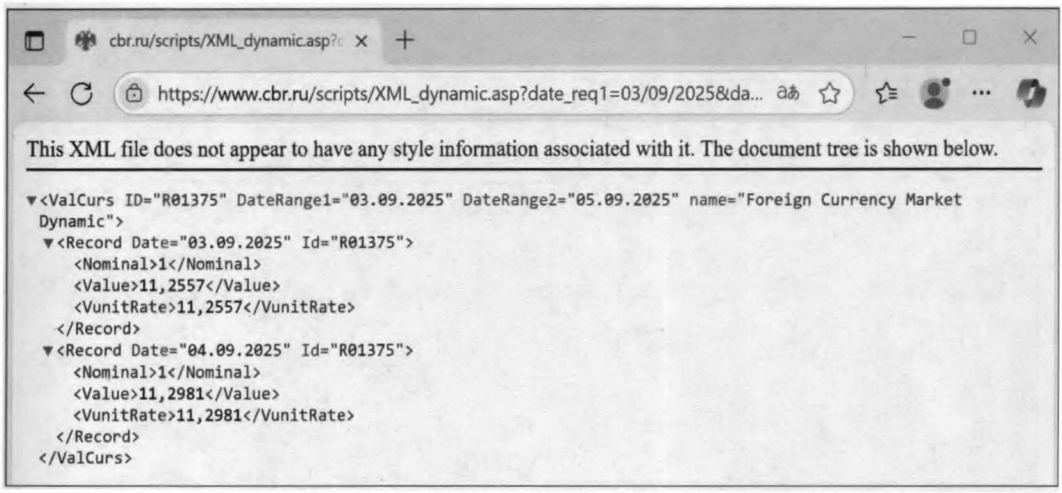


Рис. В.2. ***Ответ сервера ЦБ в браузере***

2. Команда ВЕБСЛУЖБА заходит по этой ссылке.

   Excel скачивает маленький текстовый файлик в формате XML с данными о курсе юаня.

3. Функция ФИЛЬТР.XML вытаскивает из этого файла именно нужное значение.

   » В XML много служебных строк, но нас интересуют только числа курса.

   » XPath-запрос *"//Value"* говорит: "найди все значения внутри тега <Value>".

   » Обычно это одно число — курс юаня к рублю.

4. В начале строки добавляется подпись:

`="Юань (CNY): "&` ... склеивает текст и найденный курс.

Готово! Теперь курсы валют в Excel.

В.6.2. Способ 2: визуальный конструктор (для тех, кто доверяет кнопкам)

Этот метод не требует даже копирования формул. Мы воспользуемся мощным встроенным инструментом Excel под названием Power Query. Он позволяет настроить получение данных через удобный графический интерфейс.

Шаг 1. Создайте новый файл Excel.

Перед вами пустой лист — это рабочее поле, куда мы поместим курсы валют (рис. В.3).

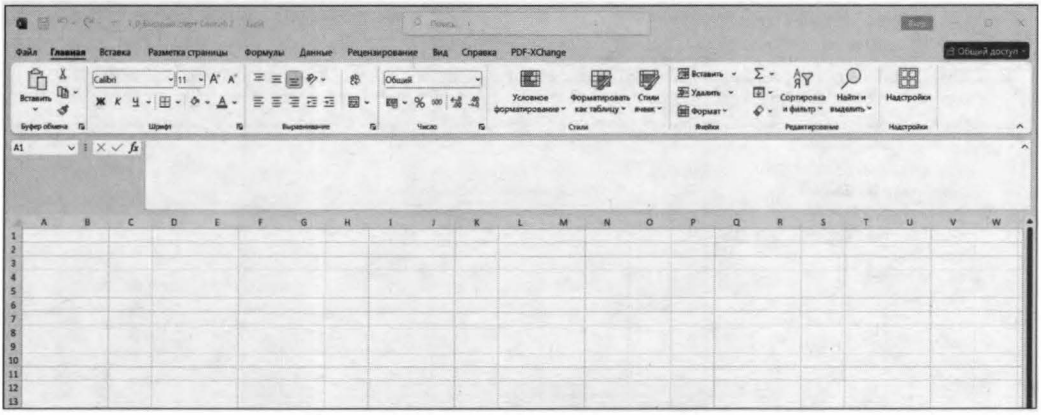


Рис. В.3. ***Чистый лист Экселя 2024***

Шаг 2. Перейдите на вкладку Данные.

В верхней ленте меню найдите и нажмите на вкладку **Данные** (рис. В.4). В левой части ленты вы увидите группу кнопок **Получить и преобразовать данные**. Нас интересует кнопка **Из Интернета**.

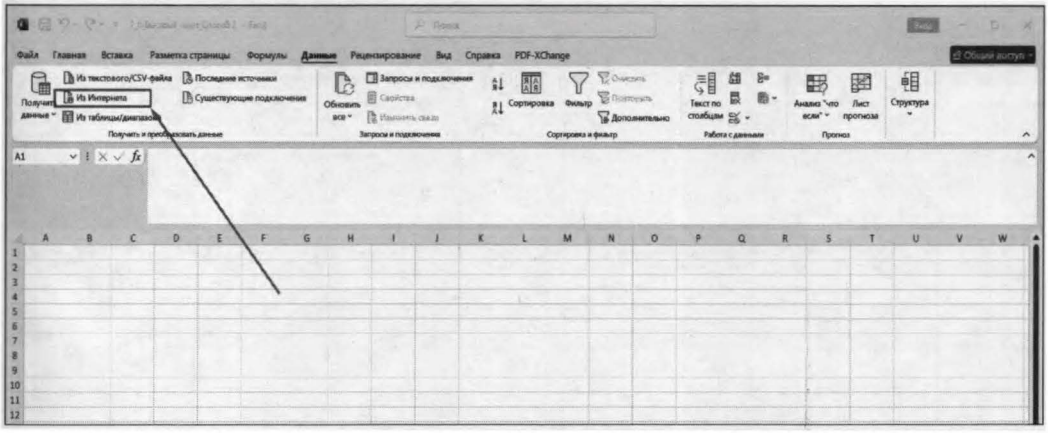


Рис. В.4. ***Лента Excel со вкладкой "Данные" и кнопкой "Из Интернета"***

Шаг 3. Вставьте ссылку на источник данных.

После нажатия на кнопку появится небольшое окно, которое попросит вас ввести URL-адрес (рис. В.5). Скопируйте и вставьте туда эту ссылку и нажмите **ОК**:

```
https://www.cbr.ru/scripts/XML_daily.asp
```

Это прямая ссылка на файл, в котором ЦБ РФ ежедневно публикует все курсы валют.

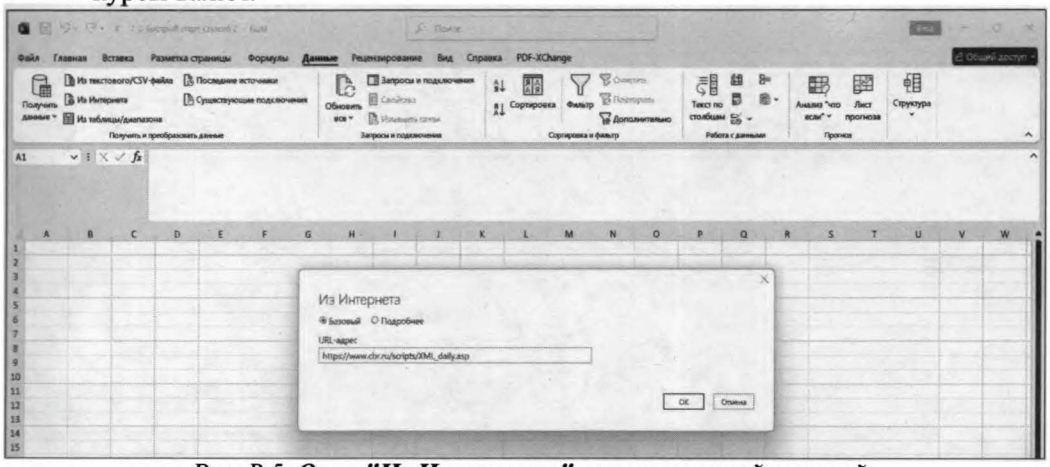


Рис. В.5. ***Окно "Из Интернета" со вставленной ссылкой***

Шаг 4. Укажите кодировку Windows-1251.

Иногда Excel сам определяет кодировку неверно, и вместо русских букв появляются "кракозябры" (рис. В.6).

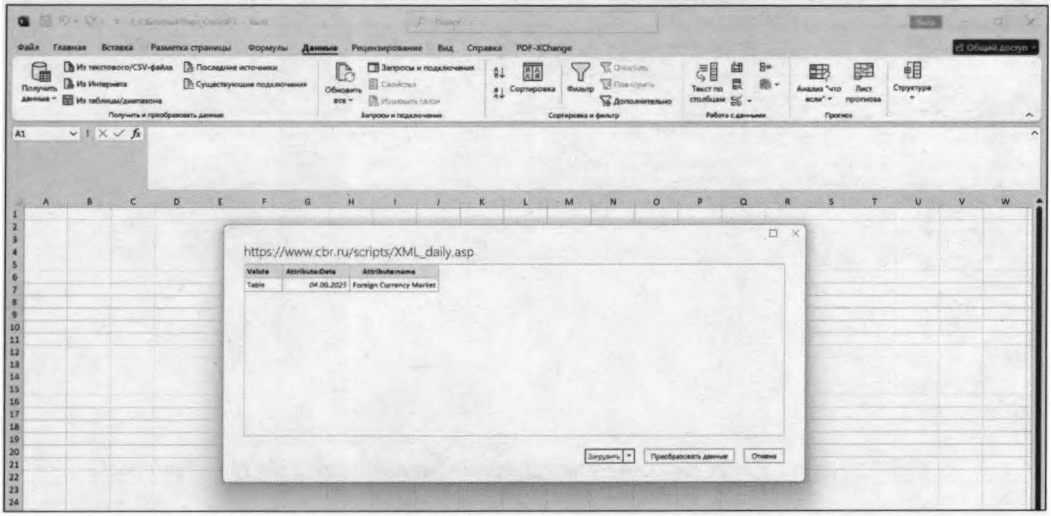


Рис. В.6. ***Окно обработки со вставленной ссылкой***

Чтобы этого избежать, нажмите **Преобразовать данные**.

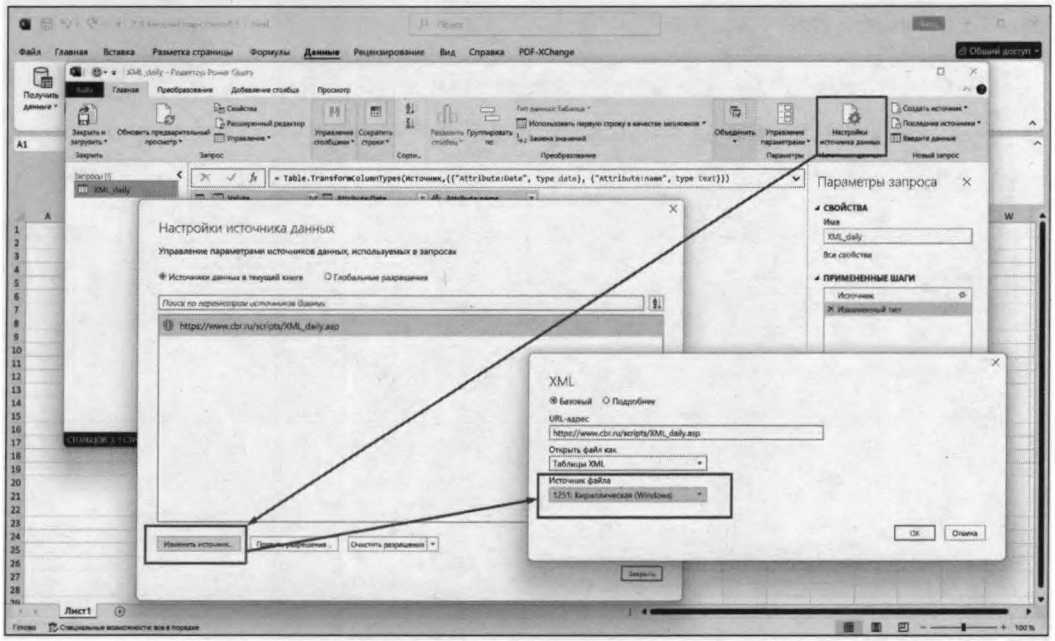


Рис. В.7. ***Настройка кодировки***

В открывшемся окне выберите **Настройка источника данных** и **Изменить источник** (рис. В.7).

В параметрах укажите кодировку: 1251: Кириллическая (Windows).

Подтвердите изменения.

Шаг 5. Выберите таблицу Valute.

Вернитесь в окно редактора Power Query. На центральной панели будет список доступных таблиц. Нажмите на **Table** под **Valute** (рис. В.8):

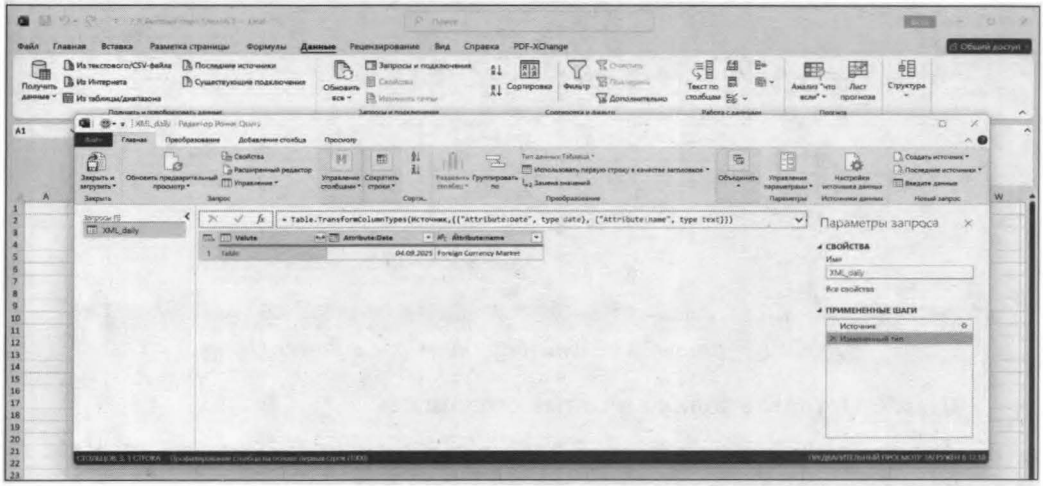


Рис. В.8. ***Переход в таблицу со значениями курсов***

По центру отобразится таблица с валютами: CharCode, Name, Value, VunitRate и т.д. (рис. В.9).

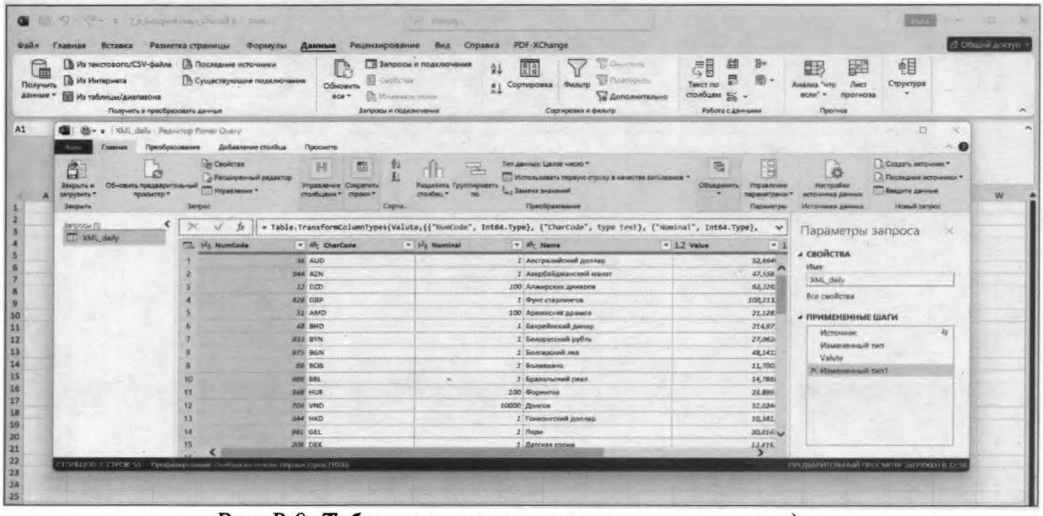


Рис. В.9. ***Таблица значения курсов валют на сегодня***

Шаг 6. Отфильтруйте только CNY и USD.

В столбце **CharCode** щелкните по фильтру и оставьте галочки только у CNY и USD (рис. В.10).

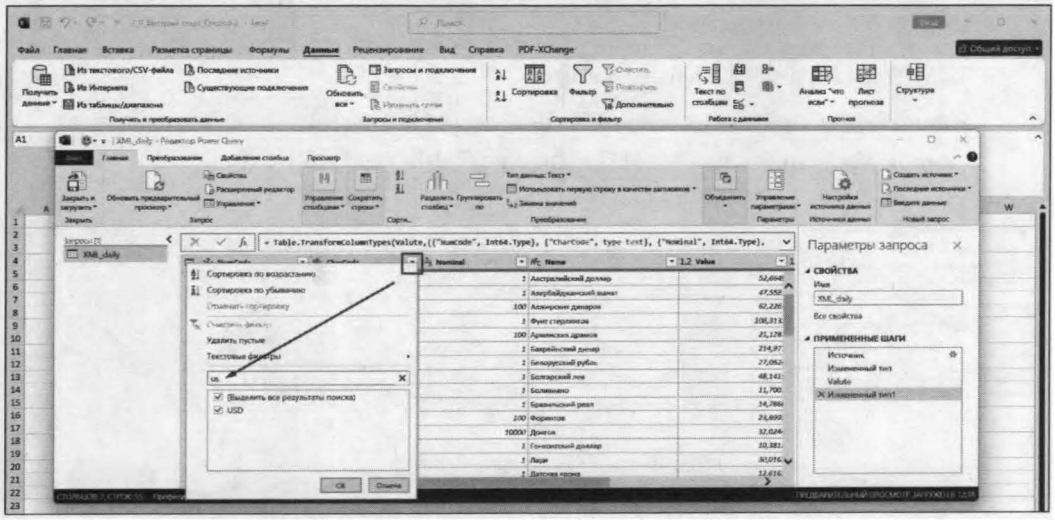


Рис. В.10. ***Фильтр по столбцу*** *CharCode* ***в Power Query***

Шаг 7. Оставьте только нужные столбцы.

Для простоты оставим только **CharCode** (код валюты), **Name** и **VunitRate** (курс). Остальные можно удалить (рис. В.11).

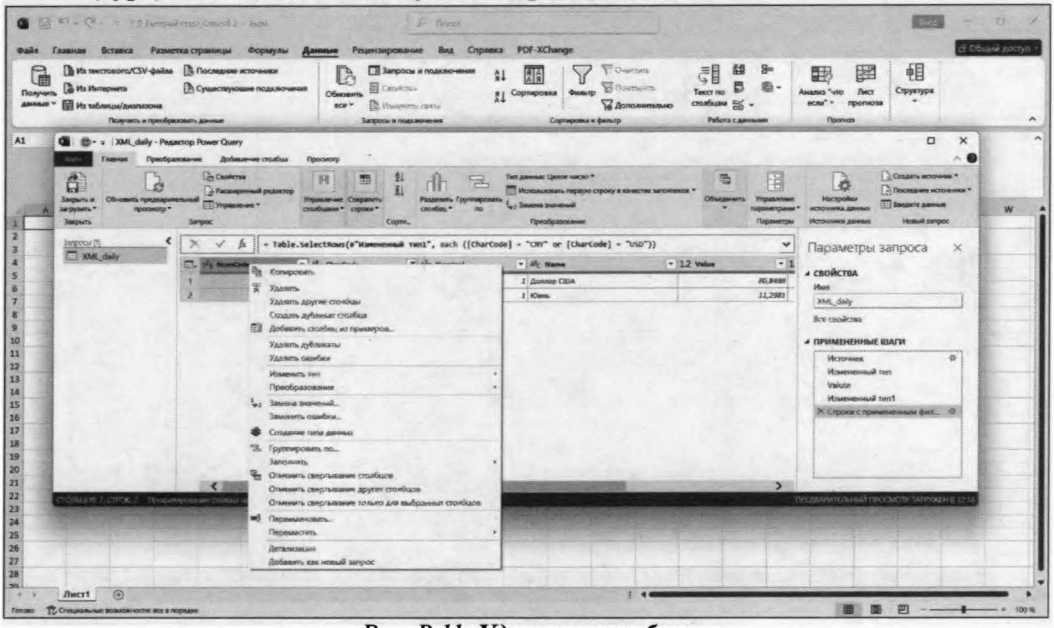


Рис. В.11. ***Удаляем столбцы***

Шаг 8. Закройте и загрузите таблицу.

Нажмите в окне редактора **Power Query** => **Закрыть и загрузить** (рис. В.12).

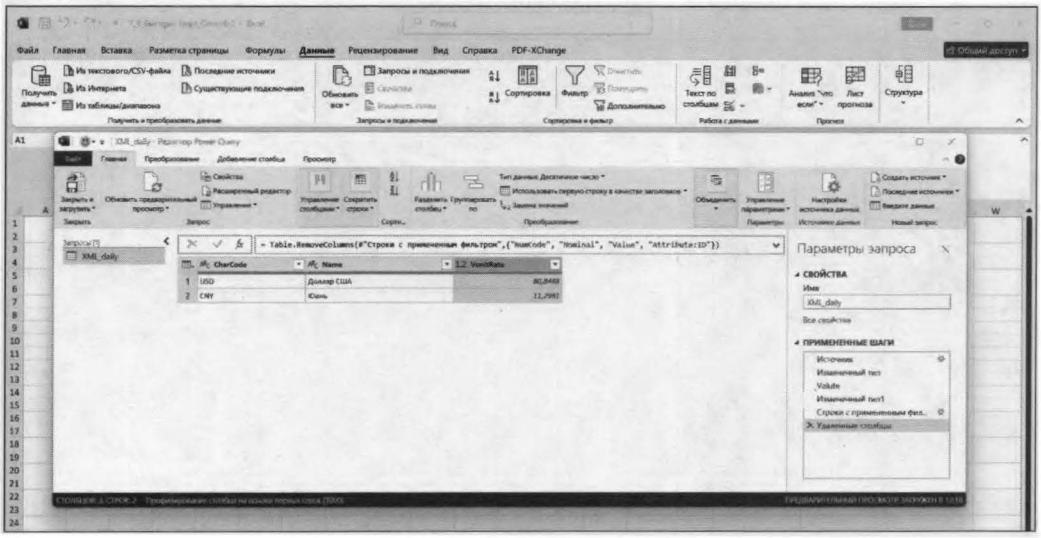


*Рис. В.12. **Окно редактора** Power Query*

На листе Excel появится аккуратная таблица (рис. В.13):

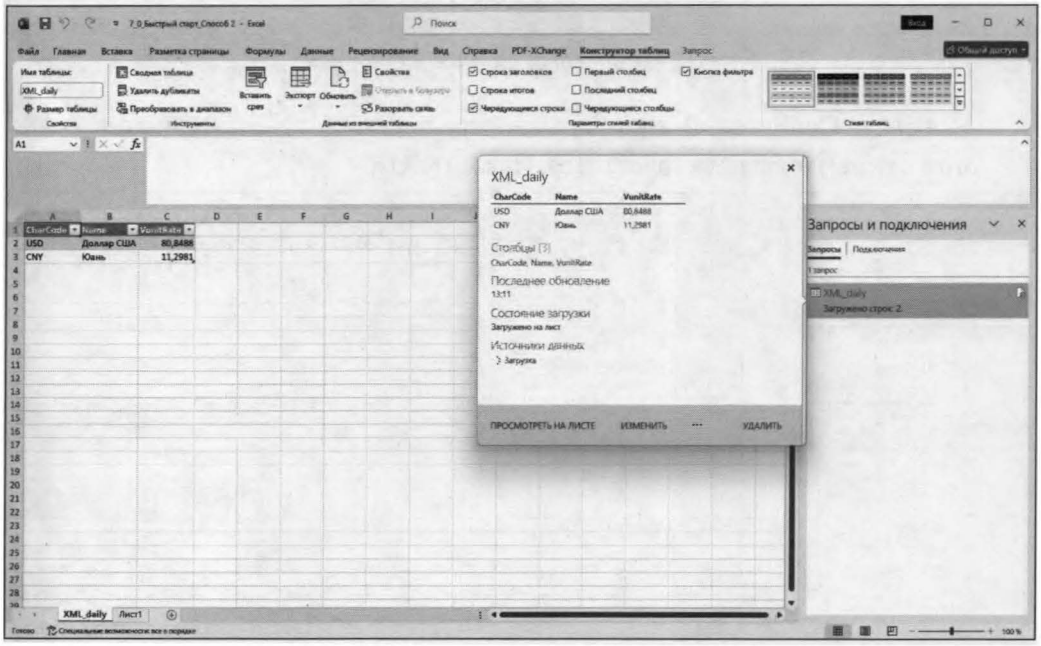


*Рис. В.13. **Загруженная таблица в Excel***

Шаг 9. Настройте автообновление.

Чтобы курсы подтягивались при каждом открытии файла, кликните правой кнопкой в свойствах таблицы (рис. В.14).

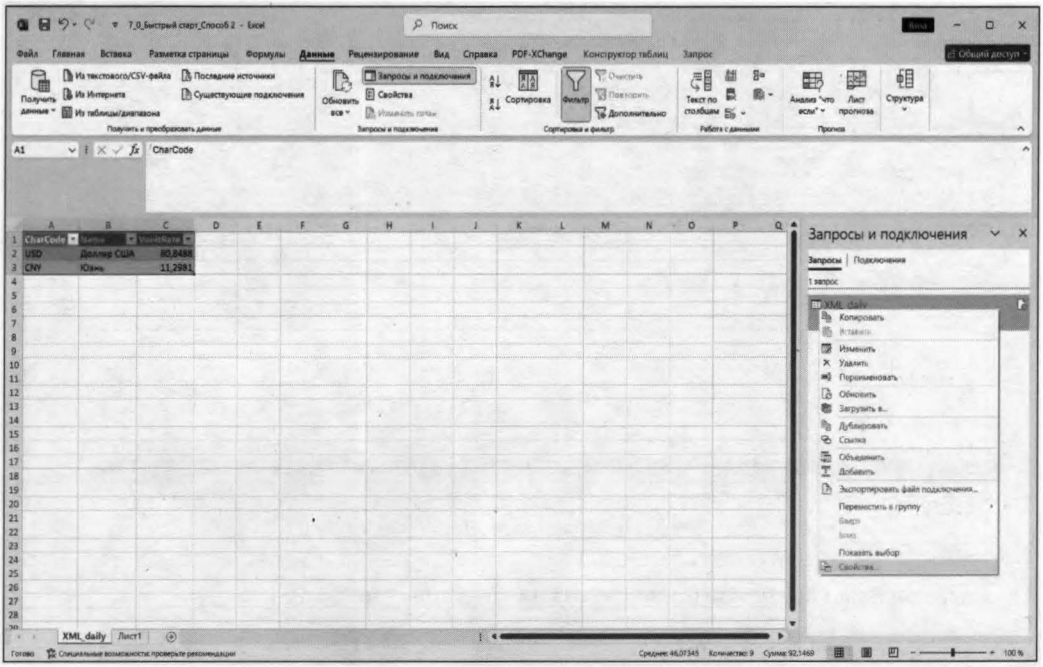


Рис. В.14. ***Свойства диапазона данных***

Выберите **Свойства**. В открывшемся окне поставьте галочку **Обновлять при открытии файла** (рис. В.15). Нажмите **ОК**.

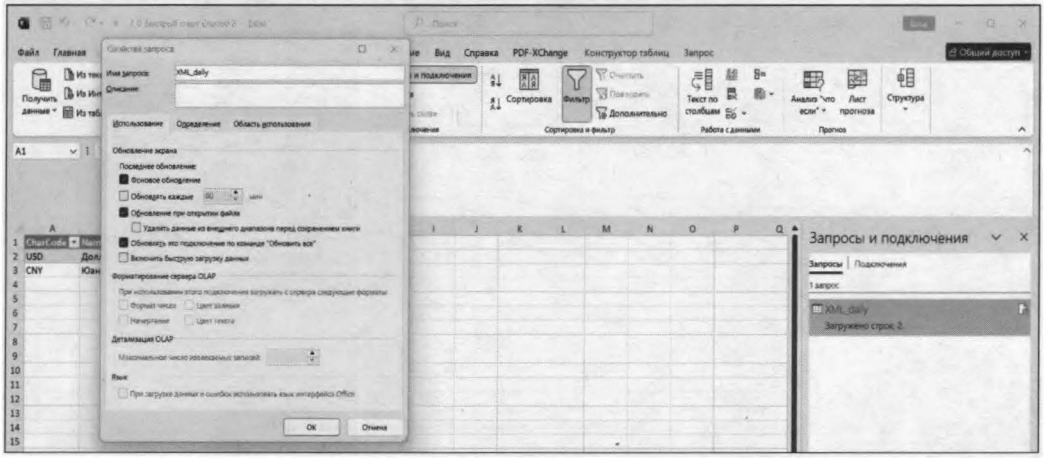


Рис. В.15. ***Настраиваем автообновление при открытии***

Теперь Excel сам будет загружать актуальные курсы юаня и доллара при каждом открытии файла.

Поздравляем! Вы только что стали автоматизатором!

Мы создали свой первый отчёт пусть и по готовым примерам, но он работает! Хотя вы не писали код, а использовались формулы, но результат ведь получен. Больше не надо копировать данные вручную – при помощи этого файла можно автоматически получать самые новые данные.

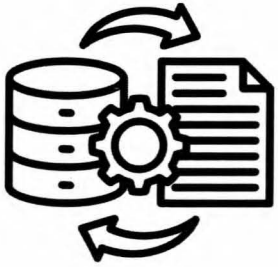
Главные принципы в данном случае – найти источник данных и затем научить приложение забирать эти данные автоматически – по расписанию или при открытии файла.

Но что если пойти дальше?

- Вместо курсов валют собирать цены конкурентов?
- Получать валюты в разных банках и искать минимальный курс?
- Объединять с другими данными по сложному запросу и отправлять отчёт в Телеграм к определенному времени?

С Excel так не получится, но нам на помощь придёт «швейцарский нож» — Python.

Дальше в книге мы разовьем похожие идеи и превратим наши рутинные процессы в автоматические сценарии.



Глава 1.

Excel и Google Таблицы на стероидах: заставляем данные работать на нас

Прежде чем мы погрузимся в API, давайте начнем с самого знакомого инструмента, который уже есть у вас под рукой, — с электронных таблиц.

Вы могли бы спросить: "Постойте, в названии книги есть Python, так почему мы начинаем с Excel?" Ответ прост: мы начинаем с родной среды обитания. Для миллионов офисных работников и менеджеров Excel и Google Таблицы — это то место, где данные рождаются, живут и обрабатываются. Прежде чем строить небоскреб автоматизации на Python, нужно заложить прочный фундамент — научиться базовым принципам на знакомой территории.

Эта глава — ваш первый и самый важный шаг от ручного труда к созданию "живых" систем. Мы возьмем привычные Excel или Google Таблицы и превратим их из пассивного хранилища цифр в активных помощников, которые сами собирают и обновляют информацию.

Анатомия "живой" таблицы

Давайте разберемся в ключевой идее.

- **Обычная таблица** — это фотография данных на определенный момент. Вы вручную скопировали цены, вбили цифры из отчета коллеги, ввели курсы валют. Эта "фотография" статична. Она устаревает в ту же секунду, как только реальные данные меняются. Любое обновление требует вашего прямого участия: найти, скопировать, вставить, проверить. Это рутина, в которой кроются ошибки.
- **"Живая" таблица** — это прямой эфир. Это не статичный снимок, а динамичное окно в мир актуальных данных. Ее главная сила — автоматическое подключение к источникам информации и самостоятельное обновление. Ваша роль меняется с исполнителя на архитектора: вы один раз настраиваете процесс, а таблица делает все сама.

Чтобы сделать этот путь понятным, мы пройдем три уровня мастерства, освоив арсенал инструментов от простого к сложному:

1. **Магия формул:** научимся получать данные из Интернета с помощью одной-единственной функции. Это наш "снайперский" инструмент для быстрых и точечных задач.
2. **Мощь визуальных конструкторов:** освоим Power Query — ваш личный "комбайн" для сбора и очистки данных из десятков источников без единой строчки кода.
3. **Гибкость скриптов:** напишем небольшие программы на VBA и Apps Script. Это "тяжелая артиллерия" для нестандартных задач, когда простые инструменты бессильны.

По итогам этой главы вы сможете автоматически получать любые данные из Интернета, превращать хаос в аккуратные отчеты. Готовы превратить свою таблицу в умного ассистента? Начнем.

1.1. Магия формул. Быстрые данные без кода

Начнем с самого простого способа "оживить" ячейку. Мы заставим ее самостоятельно забирать данные с сайта, используя лишь одну встроенную функцию.

1.1.1. Google Таблицы: погода в реальном времени с IMPORTDATA

Цель: понять базовый принцип "источник => таблица => автообновление" на простом и наглядном примере.

Задача: создать в Google Таблицах ячейку, которая всегда показывает актуальную погоду в вашем городе. Мы сделаем это без кода, используя публичный API сервиса Open-Meteo (рис. 1.1).

Почему именно он? Он бесплатен, не требует регистрации и API-ключа, а главное — умеет отдавать данные в формате CSV, идеальном для таблиц.

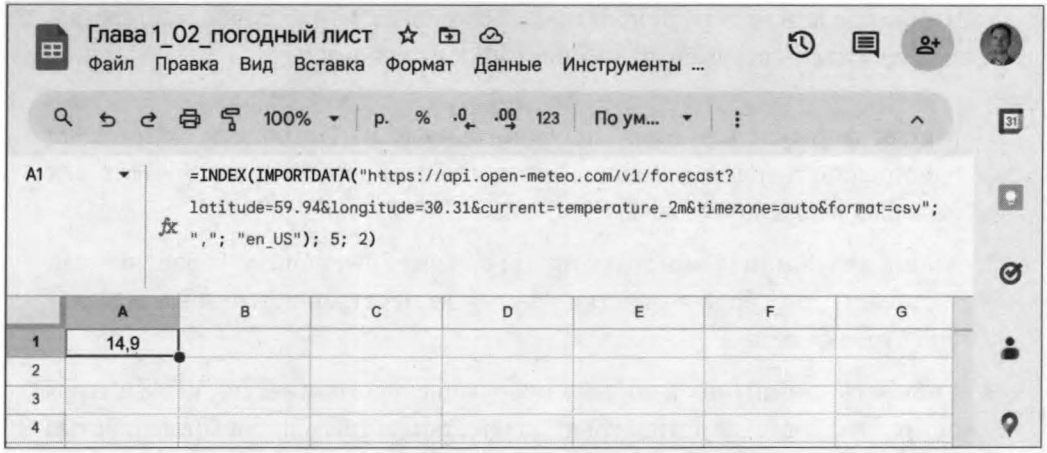


Рис. 1.1. ***Формула в*** *Google Таблицах****, получающая значение погоды в Санкт-Петербурге***

Почему выбран именно *api.open-meteo.com*?

Критерии выбора источника для нашего "живого" листа:

- **Бесплатно и без API-ключа.** Не нужно регистрироваться, не нужно хранить ключи и пароли в таблице.
- **Чистый, предсказуемый формат.** Open-Meteo умеет отдавать данные в CSV — идеальный формат для IMPORTDATA/IMPORTCSV.
- **Минимум "шума".** Это API, а не страница с HTML-шаблонами, значит, нам не придется парсить HTML и писать XPath.
- **Гибкость.** Можно запросить текущую температуру, прогноз, часовые данные и т.п. — все доступно через запрос параметров в URL.

Формула и ее подробный разбор

Вот формула, которую мы используем в Google Таблицах (в русской локали с точкой с запятой как разделителем аргументов):

```
=INDEX(
  IMPORTDATA("https://api.open-meteo.com/v1/forecast?latitude=59.94
&longitude=30.31&current=temperature_2m&timezone=auto&format=csv");
  5; 2
)
```

Пошагово, что делает формула:

- `IMPORTDATA(URL)` — эта функция отправляет запрос по указанному адресу и "всасывает" ответ в формате CSV, превращая его во временную таблицу внутри ячейки.
- `URL` — это не просто ссылка, а запрос к API. Параметрами *latitude=59.94* и *longitude=30.31* мы указываем координаты Санкт-Петербурга, *current=temperature_2m* просим вернуть текущую температуру, а *format=csv* — требуем ответ в удобном нам формате.
- `INDEX(<массив>; 5; 2)` — из всей таблицы, которую вернула IMPORTDATA, мы забираем значение из пятой строки и второго столбца. Именно там, в структуре ответа Open-Meteo, находится нужное нам число.

Почему пример в Google Таблицах, а не в Excel?

Для этой конкретной задачи Google Таблицы удобнее, так как функции IMPORTDATA, IMPORTXML, IMPORTHTML встроены и прекрасно работают с веб-источниками "из коробки".

Как адаптировать под ваш город?

Откройте Google Maps, кликните правой кнопкой по нужному месту на карте, и вы увидите его координаты. Подставьте их в URL формулы. Готово!

1.1.2. Excel: котировки акций с WEBSERVICE + FILTER.XML

Мы увидели, как легко это делается в Google. А что же Excel? В его арсенале есть своя мощная связка для работы с веб-данными в формате XML — ВЕБСЛУЖБА и ФИЛЬТР.XML. Сразу оговоримся: этот метод работает в Excel для Windows.

> **Кейс**: вы хотите отслеживать в своей таблице стоимость акций Сбербанка (тикер SBER) с Московской биржи (рис. 1.2). Вам нужно только одно значение — цена вчерашнего закрытия. API Мосбиржи умеет отдавать данные в формате XML, что идеально подходит для нашей задачи.

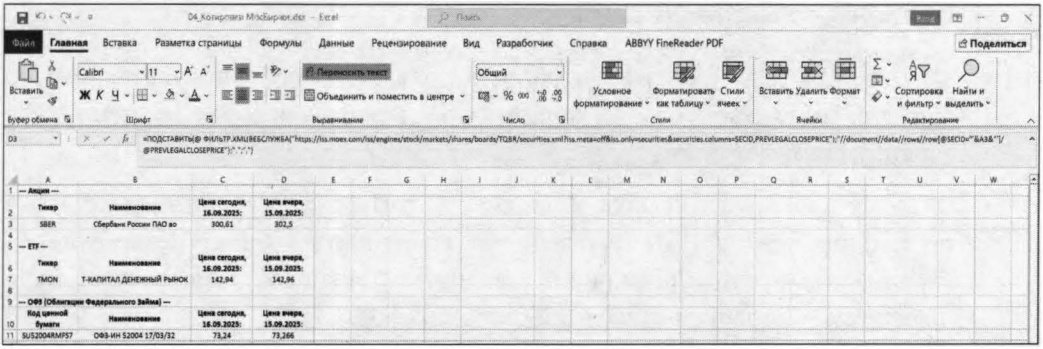


Рис. 1.2. ***Получение котировок Мосбиржи для ознакомительных целей***

Решение: создадим простой трекер котировок.

1. В ячейку A1 вводим тикер интересующей нас акции, например SBER.
2. В ячейку D3, где мы хотим видеть цену, вставляем нашу формулу:

```
=ПОДСТАВИТЬ(@ФИЛЬТР.XML(ВЕБСЛУЖБА("https://iss.moex.com/iss/engines/
stock/markets/shares/boards/TQBR/securities.xml?iss.meta=off&iss.
only=securities&securities.columns=SECID,PREVLEGALCLOSEPRICE");"//
document//data//rows//row[@SECID='"&A3&"']/@PREVLEGALCLOSEPRICE");".";",")
```

Давайте посмотрим по шагам, что делает формула.

Шаг 1. ВЕБСЛУЖБА("...") — получаем данные.

Эта функция — наш курьер. Она отправляется по указанному URL и приносит все, что найдет в ответ. Адрес, по которому она идет, — это не просто сайт, а специальный запрос к API Московской биржи (ISS MOEX API):

https://iss.moex.com/iss/engines/stock/markets/shares/boards/TQBR/securities.xml?iss.meta=off&iss.only=securities&securities.columns=SECID,PREVLEGALCLOSEPRICE

- **.../securities.xml** — мы запрашиваем список всех ценных бумаг в режиме торгов TQBR (основной для акций) и просим отдать его в формате XML.
- **?iss.meta=off** — говорим серверу: "Не присылай нам метаданные (описание полей), это лишняя информация, нам нужны только сами данные".
- **&iss.only=securities** — из всего ответа нас интересует только блок с данными о бумагах (securities).

- **&securities.columns=SECID,PREVLEGALCLOSEPRICE** — ключевая оптимизация. Мы просим биржу прислать не всю таблицу с десятками столбцов, а только два: SECID (тикер) и PREVLEGALCLOSEPRICE (цена закрытия предыдущего торгового дня). Это делает ответ сервера компактным и ускоряет загрузку.

  В результате ВЕБСЛУЖБА возвращает в ячейку большой кусок текста в формате XML, содержащий котировки всех акций.

Шаг 2. ФИЛЬТР.XML(... ; "...") — находим нужное.

Теперь в дело вступает ФИЛЬТР.XML. Это наш аналитик, который умеет читать XML и находить в нем конкретные значения с помощью языка запросов XPath.

```
"//document//data//rows//row[@SECID='"&A3&"']/@PREVLEGALCLOSEPRICE"
```

Это не абракадабра, а точный адрес нужной нам ячейки внутри XML-документа.

- **//document//data//rows//row** — идем по структуре XML вглубь: ищем корневой элемент, в нем data, затем rows и, наконец, все элементы row (каждая такая строка – это данные по одной акции).
- **[@SECID='"&A3&"']** — а вот и фильтр! Мы говорим: "Найди мне ту строку (row), у которой атрибут SECID равен значению из ячейки A3". Конструкция '"&A3&"' динамически подставляет тикер SBER (или любой другой) из нашей таблицы прямо в запрос.
- **/@PREVLEGALCLOSEPRICE** — финальный шаг. Когда нужная строка найдена, мы говорим: "Из этой строки мне не нужна вся информация, верни только значение атрибута PREVLEGALCLOSEPRICE".

  В итоге ФИЛЬТР.XML возвращает одно-единственное значение — цену закрытия для тикера из ячейки A3.

Шаг 3. ПОДСТАВИТЬ(@ ... ;".";",") — приводим в порядок.

Мы почти у цели, но есть нюанс. Большинство API отдают числа с точкой в качестве десятичного разделителя (например, 123.45), а в русской локали Excel ожидает запятую. Функция ПОДСТАВИТЬ решает эту проблему, заменяя точку запятой и превращая текст в полноценное число.

Символ @ перед функцией — это оператор неявного пересечения, который нужен для корректной работы с динамическими массивами в новых версиях Excel, но в данном случае он просто помогает формуле правильно обработать одиночное значение.

Вот и все! Мы одной формулой создали "живую" ячейку, которая сама ходит на сайт Мосбиржи, находит нужную акцию и забирает ее котировку. Поменяйте тикер в А3 — и цена тут же обновится. Это идеальный пример того, как старые, но мощные инструменты могут решать современные задачи элегантно и без единой строчки кода.

1.2. Визуальный конструктор. Собираем данные с Power Query в Excel

Формулы отлично справляются с получением одного-двух значений. Но что если данных много, они из разных источников и вдобавок идут вразнобой? Здесь на сцену выходит инструмент второго уровня — визуальный конструктор Power Query, ваш личный дата-центр внутри Excel.

Кейс: вы менеджер и каждую неделю получаете от трех подрядчиков CSV-отчеты о затраченном времени. Форматы у всех разные: заголовки на разных языках, разные разделители, даты в разном виде. Ваша задача — собрать все в единый сводный отчет и обновлять его одним кликом.

Практика по шагам

1.2.1. Подключение: создаем три запроса к трем CSV

В Excel: **Данные => Получить данные => Из файла => Из текстового/CSV-файла**. Выберите `03_contractor_1.csv` — Power Query определит разделитель и покажет предпросмотр **(рис. 1.3)**. Нажмите **Преобразовать данные** (не **Загрузить в лист**).

Повторите для `03_contractor_2.csv` и `03_contractor_3.csv` (рис. 1.4).

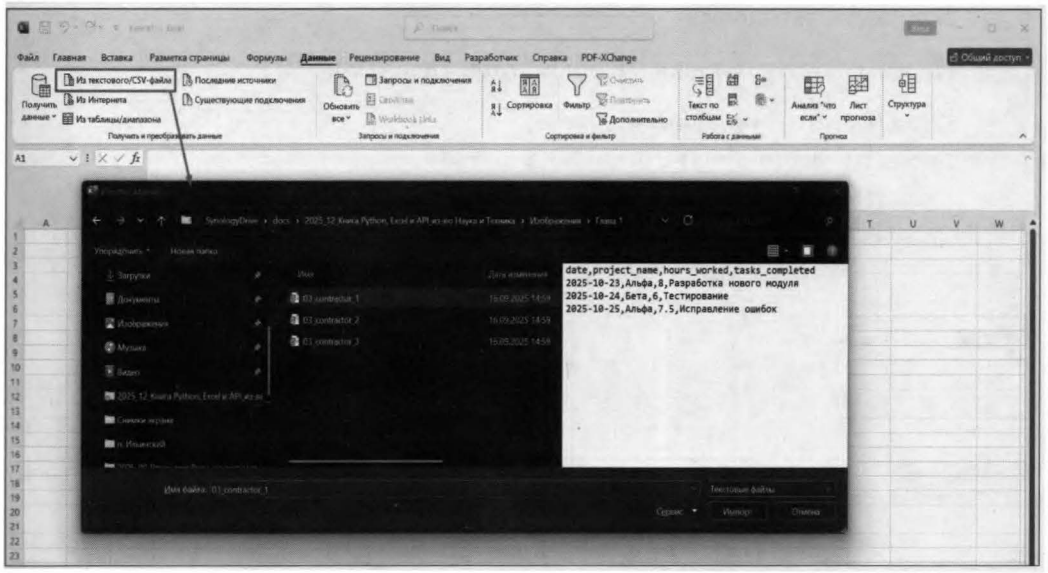


Рис. 1.3. ***Импорт из текстового/CSV-файла***

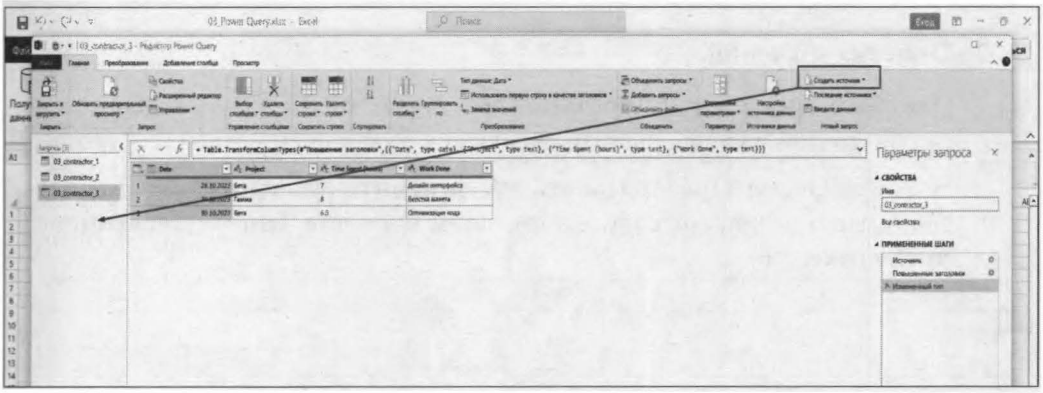


Рис. 1.4. ***Добавление файлов в редакторе Power Query***

1.2.2. Трансформация: приводим столбцы к единому виду

У каждого файла свои заголовки и форматы. Пройдем по каждому из запросов и выполним одинаковые операции.

Общие шаги (их делаем в интерфейсе Power Query — они будут записаны как последовательность шагов):

1. **Привести заголовки.**

   Переименовать колонки в едином стиле (рис. 1.5):

   `Дата` (вместо `date`, `Период`, `Date`),
   `Проект` (вместо `project_name`, `Проект`, `Project`),
   `Часы` (вместо `hours_worked`, `Затраченное время`, `Time Spent (hours)`),
   `Задача` (вместо `tasks_completed`, `Выполненные задачи`, `Work Done`).

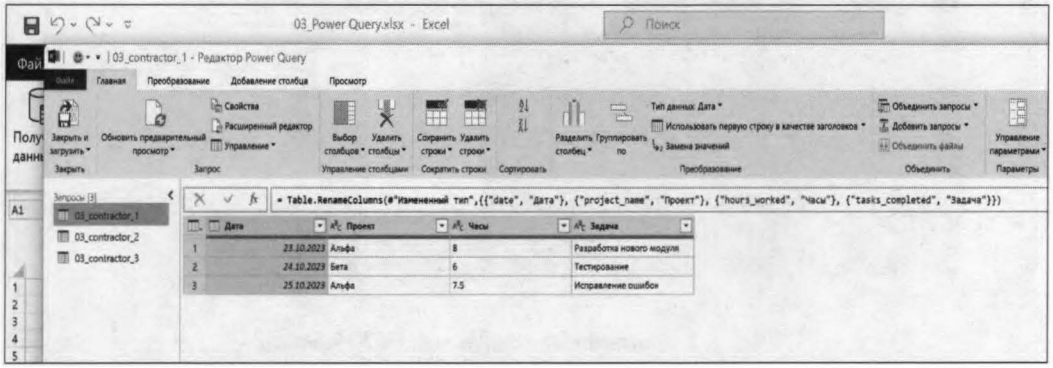


Рис. 1.5. ***Переименование заголовков***

2. **Очистка значений.**

   Для столбца `Часы`: если встречается десятичная запятая (`7,5`), замените запятую точкой или используйте смену типа с указанием локали (см. ниже).

   В Power Query: **Преобразовать => Заменить значения** — замените точку запятой (для чисел в тексте), затем выберите **Тип => Десятичное число (рис. 1.6)**.

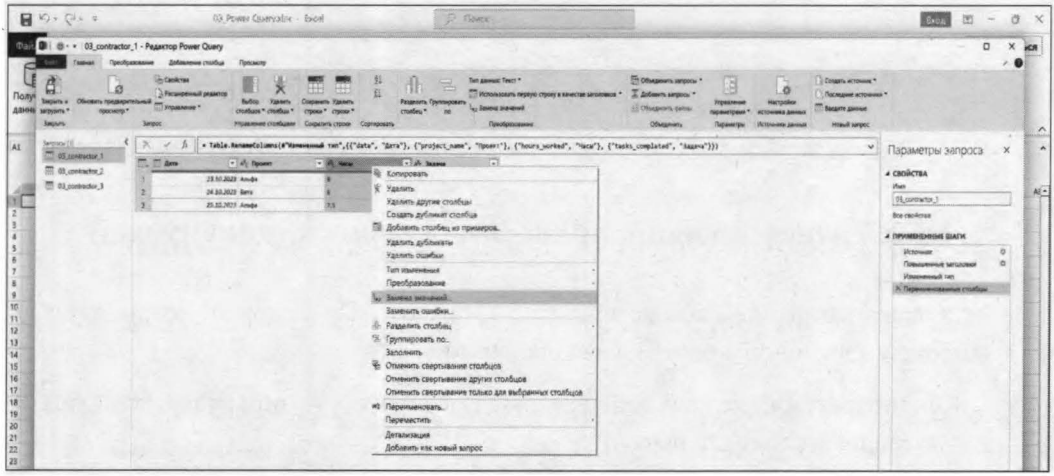


Рис. 1.6. ***Очистка значений***

3. **Даты в едином формате.**

   У файлов даты бывают `2025-10-23`, `25.10.2025`, `2025/10/28`. Для безопасного преобразования: **Изменить тип => Используя локаль => Дата** и выбрать подходящую локаль/формат. Если формат смешан, используйте `Date.FromText` с явным преобразованием или добавьте предварительный шаг, нормализующий строку (`Text.Replace("/", "-")`, затем `Date.FromText`).

   Пример расширенного редактора. Это только иллюстрация — Power Query сам сгенерирует похожий код:

```
let
    Источник =
Csv.Document(File.Contents("D:\SynologyDrive\docs\2025_12_Книга
Python, Excel и API_из-во Наука и Техника\Изображения\Глава 1\03_
contractor_3.csv"),[Delimiter=",", Columns=4, Encoding=65001,
QuoteStyle=QuoteStyle.None]),
    #"Повышенные заголовки" = Table.PromoteHeaders(Источник,
[PromoteAllScalars=true]),
    #"Измененный тип" = Table.TransformColumnTypes(#"Повышенные
заголовки",{{"Date", type date}, {"Project", type text}, {"Time
Spent (hours)", type text}, {"Work Done", type text}}),
    #"Переименованные столбцы" = Table.RenameColumns(#"Измененный
тип",{{"Date", "Дата"}, {"Project", "Проект"}, {"Time Spent (hours)",
"Часы"}, {"Work Done", "Задача"}}),
    #"Замененное значение" = Table.ReplaceValue(#"Переименованные
столбцы",".",",",Replacer.ReplaceText,{"Часы"}),
    #"Измененный тип1" = Table.TransformColumnTypes(#"Замененное
значение",{{"Часы", type number}})
in
    #"Измененный тип1"
```

1.2.3. Объединение: соединяем три очищенных запроса

Когда все три запроса имеют одинаковые имена колонок и корректные типы:

- В одном из запросов: **Главная => Объединить => Добавить запросы =>** выбрать все три таблицы. Получим одну таблицу с колонками `Дата`, `Проект`, `Часы`, `Задача` (это рис. 1.7).

- Дополнительно:
  - » отсортировать по `Дата` (по убыванию или возрастанию);
  - » удалить дубликаты, если подрядчики иногда дублируют строки.

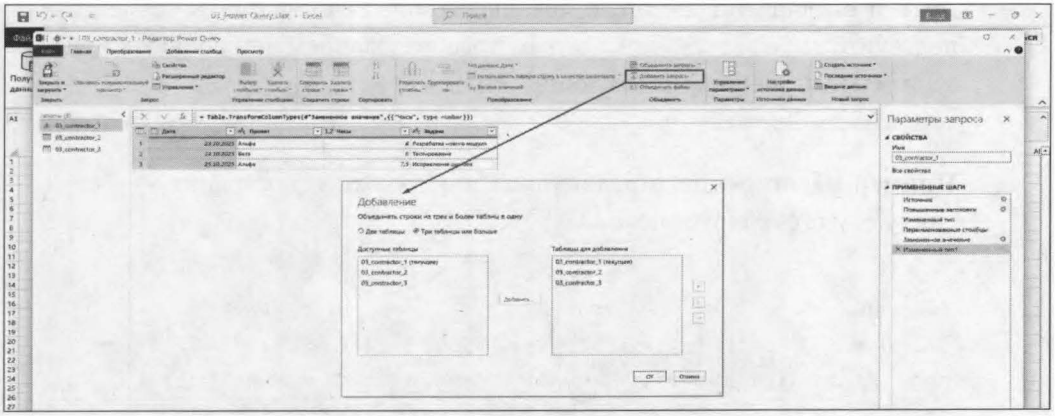


Рис. 1.7. ***Объединенный запрос***

1.2.4. Загрузка и обновление

Нажмите **Закрыть и загрузить => Закрыть и загрузить в...** — выберите **Таблица на лист Excel** (рис. 1.8). Теперь обновление простое: **Данные => Обновить все** или правой кнопкой по таблице **=> Обновить** (рис. 1.9).

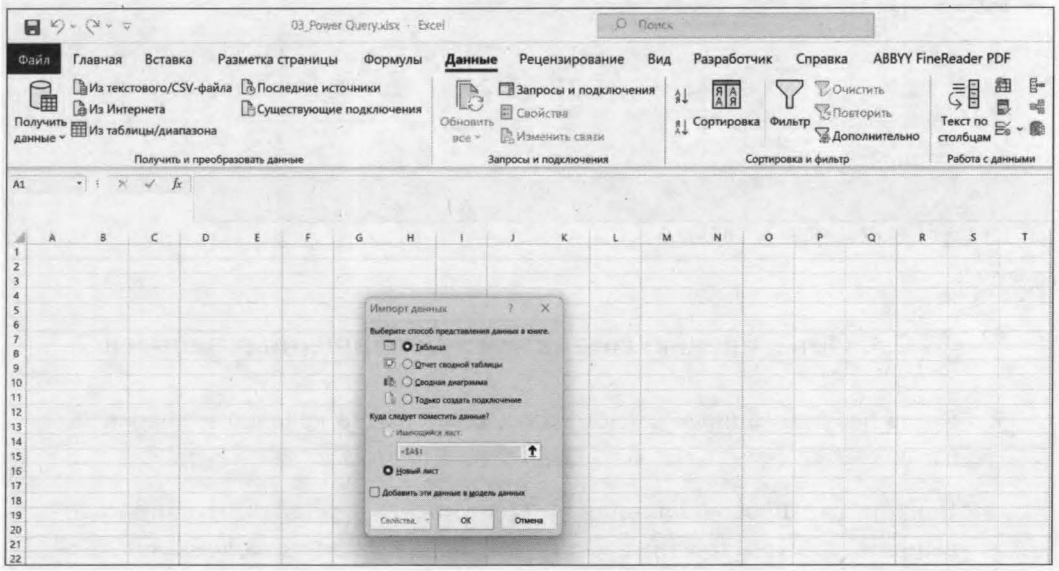


Рис. 1.8. ***Загрузка в таблицу***

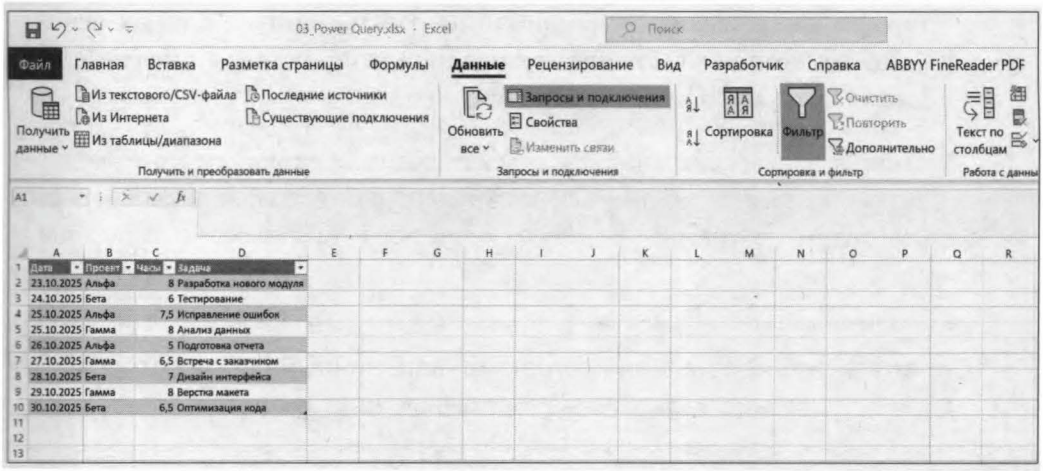


Рис. 1.9. ***Отображение в таблице***

Теперь, когда на следующей неделе придут новые файлы, вам достаточно будет нажать одну кнопку: **Данные => Обновить все**. Power Query сам повторит все шаги и обновит итоговый отчет. Вы только что построили свой первый автоматизированный конвейер по обработке данных!

1.3. Пишем скрипты. Когда формул и конструкторов недостаточно

Вы проделали огромную работу. Научились "оживлять" ячейки и строить мощные конвейеры в Power Query. Эти инструменты идеально подходят для работы с "цивилизованными" источниками — аккуратными файлами и API.

Но что делать, когда цифровой мир перестает быть дружелюбным?

- **Что если нужные данные есть только на старом сайте,** где нет никакого API, а информация запрятана глубоко в HTML-коде, рекламных баннерах и меню? Формулы здесь бессильны, а Power Query может не справиться со сложной структурой.
- **Что если для доступа к API нужно не просто перейти по ссылке,** а пройти аутентификацию, отправить специальные заголовки или выполнить серию запросов в строгом порядке?

- **Что если ваша задача — не просто получить данные, а на их основе создать документ, отправить персонализированное письмо или поставить задачу в календаре?**

Именно для таких ситуаций, когда мы выходим за рамки простого "получить и преобразовать", нам нужен инструмент с безграничной гибкостью. Мы переходим на третий уровень мастерства в таблицах — написание скриптов.

Скрипт — это небольшая программа, ваш личный робот, которому можно дать точные и сложные инструкции.

В следующих двух разделах мы познакомимся с двумя такими "роботами":

1. **VBA (Visual Basic for Applications)** — классическая "рабочая лошадка" внутри Microsoft Excel. Идеально подходит, чтобы "соскрести" данные со сложных веб-страниц и автоматизировать задачи прямо в вашем Excel.
2. **Google Apps Script** — облачный "швейцарский нож" для экосистемы Google. Его сила — не только в работе с таблицами, но и в умении дирижировать целым оркестром сервисов: GMail, Google Docs, Календарем и внешними API.

Не пугайтесь слова "код". Мы будем двигаться шаг за шагом. Вы увидите, что скрипты — это не магия, а логичный способ решать задачи, которые раньше казались невыполнимыми.

1.3.1. Скрапинг в Excel с VBA: парсим Википедию

Кейс: нам нужен актуальный список крупнейших городов России. Идеальный источник — страница в Википедии. Проблема: страница огромная, и стандартные функции Excel не смогут ее обработать.

Задача: написать VBA-скрипт (макрос), который самостоятельно зайдет на страницу Википедии, найдет таблицу с городами, отфильтрует только города-миллионники и выведет их на лист Excel.

Решение:

Прежде чем мы начнем писать код, нужно подготовить наш Excel.

Шаг 0. Включаем режим разработчика.

По умолчанию вкладка для работы с макросами скрыта. Чтобы ее включить:

1. Нажмите **Файл => Параметры**.
2. В открывшемся окне выберите **Настроить ленту**.
3. В правом списке ("Основные вкладки") поставьте галочку напротив пункта **Разработчик** и нажмите **ОК** (рис. 1.10).

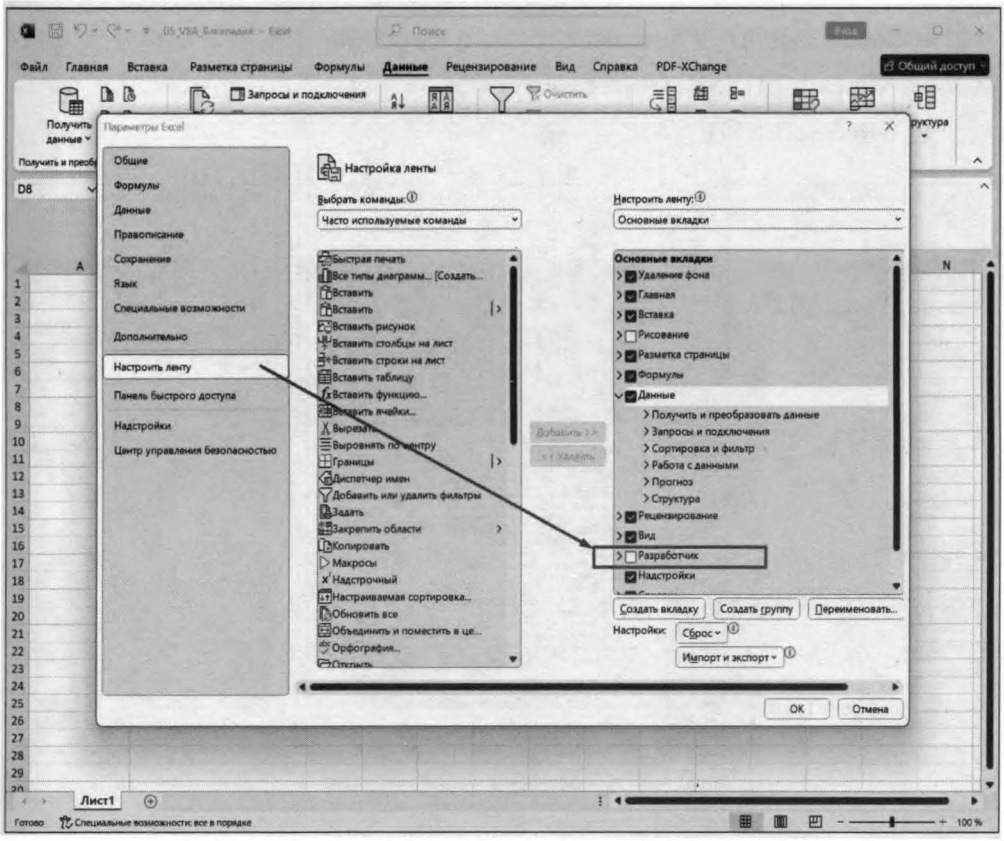


Рис. 1.10. ***Включаем режим разработчика в Microsoft Office 2024***

Теперь у вас на главной ленте Excel появилась новая вкладка, которая открывает доступ ко всем инструментам для создания макросов.

Шаг 1. Открываем редактор VBA и создаем модуль.

Перейдите на вкладку **Разработчик** и нажмите самую левую кнопку — **Visual Basic** (или используйте горячие клавиши **Alt + F11**). Откроется редактор кода. Чтобы было куда писать наш скрипт, создадим для него "чистый лист" — модуль. В меню редактора выберите **Insert => Module**.

Шаг 2. Подключаем необходимые библиотеки.

Наш скрипт будет работать с веб-запросами (HTTP) и структурой HTML-документов. Чтобы Excel "понимал" эти технологии, нужно подключить две встроенные библиотеки (рис. 1.11):

1. В редакторе VBA выберите **Tools => References...**.
2. В списке найдите и поставьте галочки напротив:
   - **Microsoft HTML Object Library**,
   - **Microsoft XML, v6.0**.

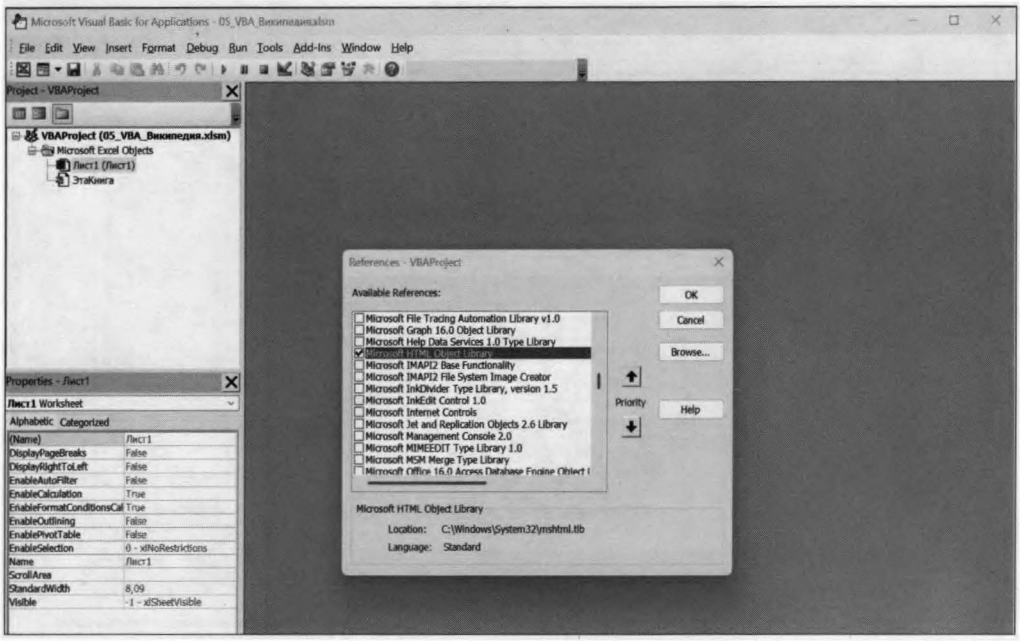


Рис. 1.11. ***Подключаем необходимые библиотеки***

Это даст нашему коду доступ к объектам для работы с Интернетом и HTML.

Шаг 3. Пишем и разбираем код.

Теперь вставьте следующий код в окно созданного модуля Insert/Module (рис. 1.12):

```
Option Explicit
' Принуждает объявлять все переменные — защищает от опечаток в именах
переменных


Sub ParseWikipediaCities()
    ' НАЗНАЧЕНИЕ ПРОЦЕДУРЫ:
    ' Парсит список российских городов с населением более миллиона человек
    ' с Википедии и выводит города-миллионники в Excel-таблицу

    ' --- 1. ОБЪЯВЛЕНИЕ ПЕРЕМЕННЫХ ---
    ' Переменные для работы с HTTP-запросами и HTML-парсингом
    Dim http As Object          ' Объект для HTTP-запроса к веб-серверу
    Dim htmlDoc As Object       ' Объект HTML-документа для парсинга
страницы
    Dim cityTable As Object     ' Объект таблицы с данными о городах
    Dim row As Object           ' Объект строки таблицы для перебора данных

    ' Переменные для работы с Excel
    Dim ws As Worksheet         ' Рабочий лист Excel для вывода результатов
    Dim rowNum As Long          ' Номер текущей строки для записи данных

    ' Переменные для хранения данных
    Dim url As String           ' URL-адрес веб-страницы для парсинга
    Dim cityName As String      ' Название города
    Dim population As Long      ' Население города (в тысячах человек)

    ' --- 2. ИНИЦИАЛИЗАЦИЯ И НАСТРОЙКА ---
    ' URL страницы Википедии со списком российских городов
    url = "https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%B
A_%D0%B3%D0%BE%D1%80%D0%BE%D0%B4%D0%BE%D0%B2_%D0%A0%D0%BE%D1%81%D1%81%D0%B8%
D0%B8_%D1%81_%D0%BD%D0%B0%D1%81%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5%D0%BC_%D
0%B1%D0%BE%D0%BB%D0%B5%D0%B5_100_%D1%82%D1%8B%D1%81%D1%8F%D1%87_%D1%87%D0%B5
%D0%BB%D0%BE%D0%B2%D0%B5%D0%BA"
```

```
    ' Создание COM-объектов для работы с веб-данными
    Set http = CreateObject("MSXML2.XMLHTTP")       ' HTTP-клиент для
запросов
    Set htmlDoc = CreateObject("htmlfile")          ' Парсер HTML-документов
    Set ws = ThisWorkbook.ActiveSheet               ' Ссылка на активный
лист

    ' Подготовка рабочего листа
    ws.Cells.Clear                      ' Очищаем все ячейки на листе
    rowNum = 2                          ' Начинаем запись данных со 2-й
строки (1-я для заголовков)

    ' Показываем пользователю прогресс выполнения
    Application.StatusBar = "Отправка запроса на сервер..."

    ' --- 3. ПОЛУЧЕНИЕ HTML-КОДА СТРАНИЦЫ ---
    ' Выполняем HTTP GET-запрос к серверу
    http.Open "GET", url, False         ' GET — метод запроса, False —
синхронный режим
    http.send                           ' Отправляем запрос

    ' Проверяем успешность запроса (код 200 = OK)
    If http.Status = 200 Then
        ' Загружаем полученный HTML в объект документа для парсинга
        htmlDoc.body.innerHTML = http.responseText

        ' --- 4. ПАРСИНГ HTML И ИЗВЛЕЧЕНИЕ ДАННЫХ ---
        Application.StatusBar = "Поиск и обработка таблицы..."

        ' Находим первую таблицу с классом "wikitable" на странице
        ' getElementsByClassName возвращает коллекцию, (0) — берем первый
элемент
        Set cityTable = htmlDoc.getElementsByClassName("wikitable")(0)

        ' --- СОЗДАНИЕ ЗАГОЛОВКОВ ТАБЛИЦЫ ---
        ' На Википедии таблица имеет сложную структуру заголовков:
        ' - Первая строка: "№", "Город", "Население, тыс. человек"
        ' - Вторая строка: годы для колонок населения
```

```
        Dim headerRow1 As Object, headerRow2 As Object
        Set headerRow1 = cityTable.getElementsByTagName("tr")(0) ' Первая
строка заголовков
        Set headerRow2 = cityTable.getElementsByTagName("tr")(1) ' Вторая
строка заголовков (с годами)

        ' ВАЖНО: индексация в HTML начинается с 0!
        ' Получаем "Город" из второй ячейки (индекс 1, так как нулевая
ячейка = "№")
        ws.Range("A1").Value = headerRow1.Cells(1).innerText

        ' Формируем заголовок колонки населения
        ' Получаем "Население, тыс. человек" из третьей ячейки (индекс 2)
        Dim mainHeaderPart As String
        mainHeaderPart = headerRow1.Cells(2).innerText

        ' Получаем год из последней колонки (индекс 15 = 16-я ячейка)
        ' В таблице несколько колонок с разными годами, берем последнюю
(актуальную)
        Dim yearText As String
        yearText = headerRow2.Cells(15).innerText

        ' Объединяем части заголовка: "Население, тыс. человек (2025)"
        ws.Range("B1").Value = mainHeaderPart & " (" & yearText & ")"

        ' --- ПЕРЕБОР СТРОК ДАННЫХ ---
        ' Начинаем с индекса 2, пропуская 2 строки заголовков (индексы 0 и 1)
        Dim i As Long
        For i = 2 To cityTable.getElementsByTagName("tr").Length - 1
            Set row = cityTable.getElementsByTagName("tr")(i)

            ' Проверяем, что в строке достаточно ячеек (больше 16)
            ' Это защищает от ошибок при парсинге неполных строк
            If row.Cells.Length > 16 Then
                ' ИЗВЛЕЧЕНИЕ ДАННЫХ ИЗ ЯЧЕЕК:
                cityName = row.Cells(2).innerText      ' Название города
(3-я колонка, индекс 2)
                population = Val(row.Cells(16).innerText) ' Население в
последней колонке (индекс 15)
```

```
                ' ОБЪЯСНЕНИЕ Val():
                ' Val() преобразует текст в число, игнорируя нечисловые
символы
                ' Например: Val("1 500") = 1500, Val("2,3 млн") = 2.3

                ' --- 5. ФИЛЬТРАЦИЯ И ВЫВОД В EXCEL ---
                ' Отбираем только города-миллионники (население > 1000 тыс.
= 1 млн)
                If population > 1000 Then
                    ws.Cells(rowNum, 1).Value = cityName     ' Записываем
город в колонку A
                    ws.Cells(rowNum, 2).Value = population  ' Записываем
население в колонку B
                    rowNum = rowNum + 1                     ' Переходим к
следующей строке
                End If
            End If
        Next i

        ' Показываем результат пользователю
        Application.StatusBar = "Готово! Найдено городов-миллионников: " &
rowNum - 2
        ' rowNum - 2, потому что начали с 2-й строки, а rowNum указывает на
следующую пустую строку

    Else
        ' Обработка ошибки HTTP-запроса
        MsgBox "Не удалось загрузить страницу. Статус: " & http.Status & " "
& http.statusText
        Application.StatusBar = False  ' Очищаем строку состояния
    End If

    ' --- 6. ОЧИСТКА ПАМЯТИ ---
    ' ВАЖНО: всегда освобождаем ресурсы объектов для предотвращения утечек
памяти
    Set http = Nothing          ' Освобождаем HTTP-объект
    Set htmlDoc = Nothing       ' Освобождаем HTML-документ
    Set cityTable = Nothing     ' Освобождаем объект таблицы
    Set row = Nothing           ' Освобождаем объект строки
```

```
    Set ws = Nothing                ' Освобождаем ссылку на рабочий лист

    ' ФИНАЛЬНОЕ ФОРМАТИРОВАНИЕ:
    ' Автоматически подгоняем ширину колонок под содержимое
    ThisWorkbook.ActiveSheet.Columns("A:B").AutoFit

End Sub
```

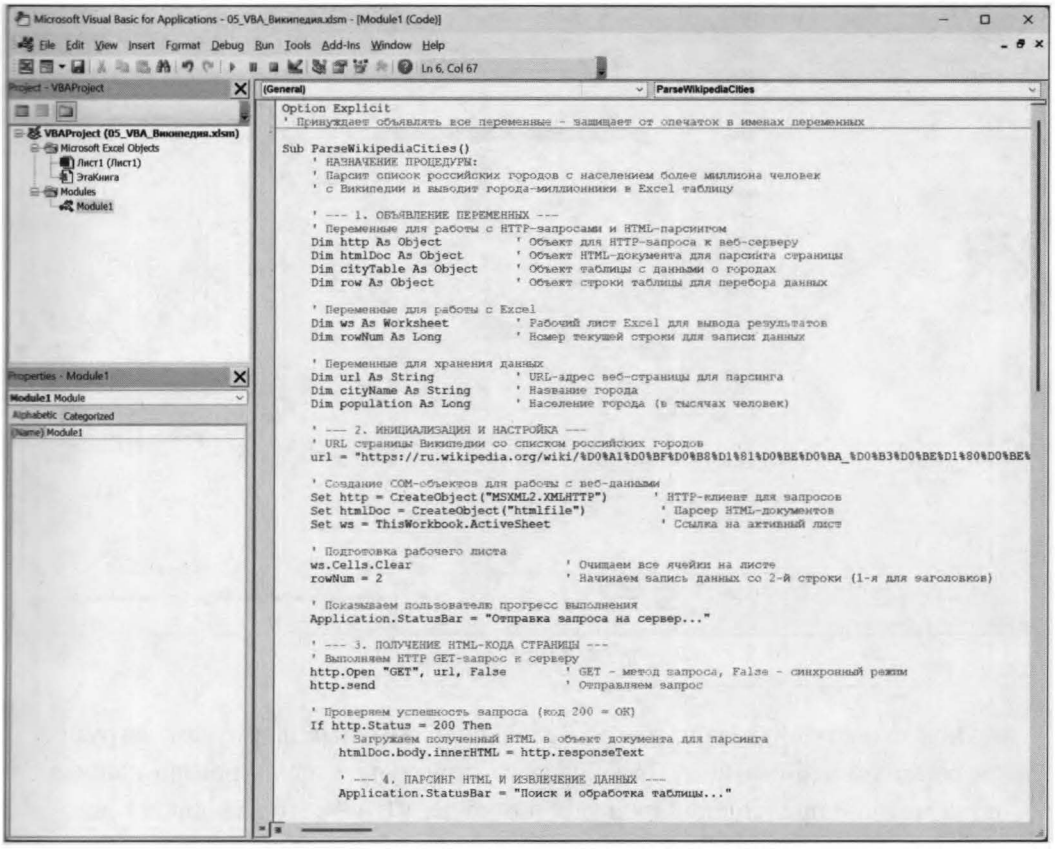


Рис. 1.12. ***Вставляем код***

Шаг 4. Запускаем макрос.

Вернитесь в окно Excel. Перейдите на вкладку **Разработчик**, нажмите **Макросы**, выберите в списке `ParseWikipediaCities` и нажмите **Выполнить**. Через несколько секунд ваш лист заполнится аккуратной таблицей с городами-миллионниками (это рис. 1.13).

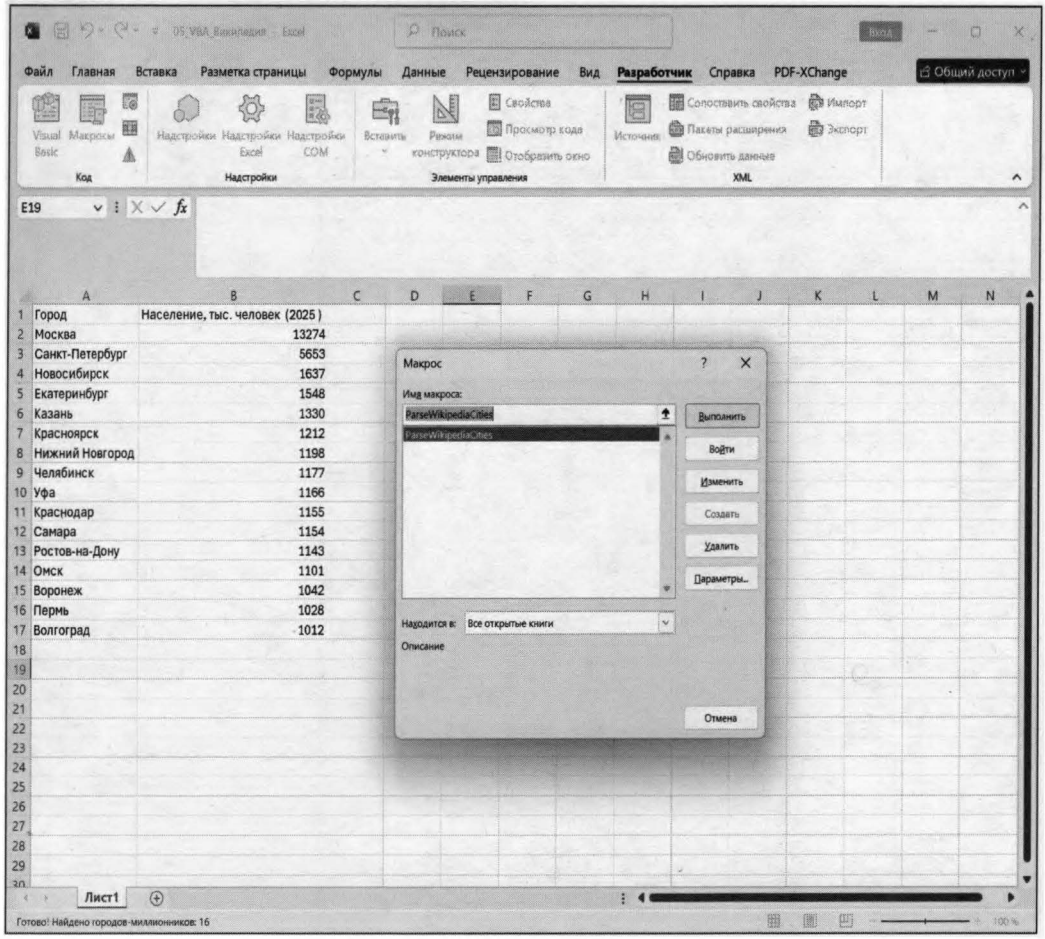


Рис. 1.13. ***Запускаем макрос***

Мы только что сделали то, что не под силу обычным формулам: загрузили сложную веб-страницу, разобрали ее структуру, отфильтровали данные по условию и представили их в удобном виде. VBA — это ваш ключ к автоматизации нестандартных задач, когда другие, более простые инструменты, достигают своего предела.

1.3.2. Облачная автоматизация в Google Таблицах с Apps Script: создаем ассистента по поиску цен

Теперь перенесемся в облачную вселенную Google. Если VBA — это для работы внутри Excel, то Apps Script — это пульт управления вашим личным

центром автоматизации, способный дирижировать целым оркестром веб-сервисов.

> **Кейс**: вы готовите смету для тендера. Нужно быстро узнать рыночные цены на сотни позиций. Ручной поиск — это часы монотонной работы и риск ошибок.

Задача: создать в Google Таблицах "умного помощника", который возьмет на себя самую нудную часть работы. Скрипт должен сам пробежаться по списку товаров, найти в Интернете их цены и подставить в таблицу рядом с названием товара кликабельную ссылку на источник. Роль человека смещается с механического исполнителя на контролера: остается лишь проверить адекватность найденных цен и принять финальное решение.

Решение: мы напишем небольшой скрипт на Google Apps Script, который будет использовать Google Custom Search API — специальный инструмент для разработчиков, позволяющий встроить поиск Google в свои приложения.

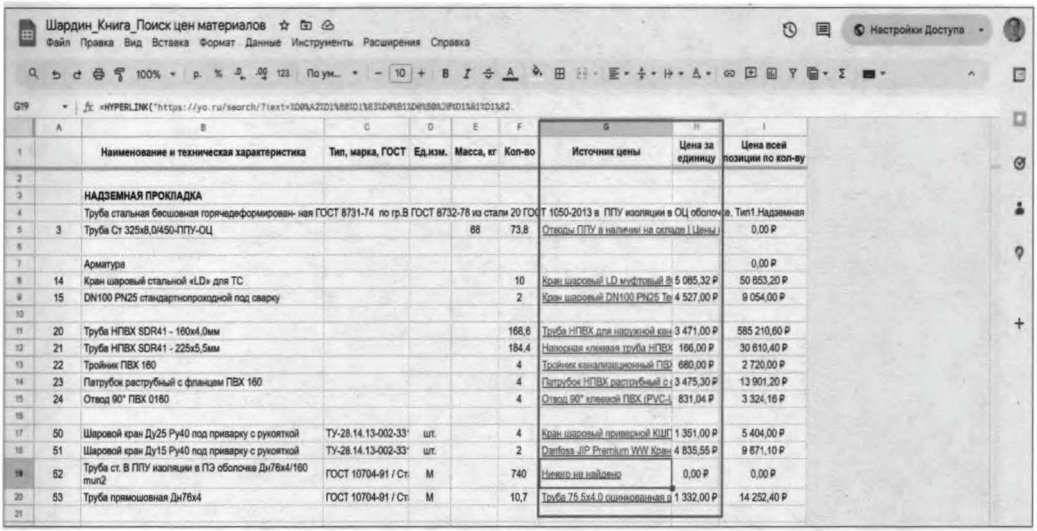


Рис. 1.14. ***Выделенные красным столбцы заполняются автоматически***

Вот как выглядит итоговый результат. Столбцы, выделенные на скриншоте, заполняются полностью автоматически после запуска скрипта (рис. 1.14).

Ссылка на рабочий пример:

https://docs.google.com/spreadsheets/d/1Xw5lm8YuUCr68TirViRL7SPd3707c_14XNR9yQHZiJA/edit?usp=sharing

Код и его анатомия

Откройте вашу Google Таблицу, перейдите в меню **Расширения => Apps Script** и вставьте в редактор следующий код:

```
const secrets = {
  apiKey_token: 'xxxxxxx',  // Замените своим ключом API
  cx_token: 'xxxxxxx', // Замените идентификатором Google's Custom Search Engine (CSE) ID
  // https://cse.google.com/cse?cx=xxxxxxx&cr=countryRU
  initial_phrase: '' // Начинать все запросы со слов // Пермь купить цена
};

function processSearchTab() {
  var sheet = SpreadsheetApp.getActiveSpreadsheet().getSheetByName('Поиск');
  var data = sheet.getDataRange().getValues();
  var results = [];

  for (var i = 1; i < data.length; i++) { // Начнем с 1-й, чтобы пропустить строку заголовка
    var productName = data[i][1]; // Колонка B
    var flagValue = data[i][5]; // Колонка F

    if (productName && Number.isFinite(flagValue)) {
      var result = searchProductPrice(productName);
      var hyperlink = '=HYPERLINK("' + result.link + '"; "' + result.title + '")';
      results.push([hyperlink, result.price]);
    } else {
      results.push(['', '']); // Если условия не выполняются, оставим ячейки пустыми
    }
  }

  // Запишем результаты обратно в столбцы G и H.
```

```
  var range = sheet.getRange(2, 7, results.length, 2); // Начиная со 2-го
столбца, столбец G(7) и H(8)
  range.setValues(results);
}

function test() {
  searchProductPrice("Труба НПВХ SDR41 - 160х4,0мм");
}

function searchProductPrice(productName) {
  var apiKey = secrets.apiKey_token;
  var cx = secrets.cx_token;
  productName = productName //.replace('-', ' ');
  var query = encodeURIComponent(secrets.initial_phrase + productName);
  var url = 'https://www.googleapis.com/customsearch/v1?q=' + query +
'&cr=countryRU&cx=' + cx + '&key=' + apiKey;
  // https://developers.google.com/custom-search/v1/reference/rest/v1/cse/
list?hl=ru
  console.log('url: ${url}');

  var response = UrlFetchApp.fetch(url);
  var json = JSON.parse(response.getContentText());

  if (json.items && json.items.length > 0) {
    for (var i = 0; i < json.items.length; i++) {
      if (json.items[i].pagemap && json.items[i].pagemap.offer && json.
items[i].pagemap.offer[0].price) {
        var price = json.items[i].pagemap.offer[0].price;
        price = price.replace('.', ',');
        var link = json.items[i].link;
        var title = json.items[i].title;
        console.log('${productName}:\n${title}: ${price} в ${link}');

        return {
          price: price,
          link: link,
          title: title
        };
```

```
      }
    }
  }

  console.log('Ничего не найдено для ${productName}');
  return {
    price: '0',
    link: 'https://ya.ru/search/?text=' + query,
    title: 'Ничего не найдено'
  };
}
```

Как это работает

- **Функция *processSearchTab()* — дирижер оркестра.** Это главная функция, которую мы будем запускать. Она, как усердный менеджер, берет лист "Поиск", проходит по нему строка за строкой и для каждой позиции решает, нужно ли искать цену. Условие простое: если в колонке B есть название товара, а в колонке F стоит любое число (мы используем это как флаг-команду "Искать!"), она передает название товара своему помощнику (рис. 1.15).

	A	B	C	D	E	F	G	H	I
1		Наименование и техническая характеристика	Тип, марка, ГОСТ	Ед.изм.	Масса, кг	Кол-во	Источник цены	Цена за единицу	Цена всей позиции по кол-ву
2									
3		НАДЗЕМНАЯ ПРОКЛАДКА							
4		Труба стальная бесшовная горячедеформирован- ная ГОСТ 8731-74 по гр.В ГОСТ 8732-78 из стали 20 ГОСТ 1050-2013 в ППУ изоляции в ОЦ оболочке. Тип1.Надземная тепл'к							
5	3	Труба Ст 325х8,0/450-ППУ-ОЦ			68	73,8	Ничего не найдено	0,00 ₽	0,00 ₽
6									
7		Арматура							0,00 ₽
8	14	Кран шаровый стальной «LD» для ТС				10	Кран шаровой стальной фланце	1 810,00 ₽	18 100,00 ₽
9	15	DN100 PN25 стандартнопроходной под сварку				2	LD КШЦП GAS DN40 PN40 Шар	3 108,00 ₽	6 216,00 ₽
10									

Рис. 1.15. ***Условие поиска*** *if (productName && Number.isFinite(flagValue))*

- **Функция *searchProductPrice()* — детектив-исполнитель.** Эта функция получает название товара и делает всю "грязную" работу. Она формирует специальный URL-запрос к Google API, подставляя в него наши секретные ключи и сам поисковый запрос (например, *Пермь купить цена Труба НПВХ SDR41* — я ищу эти позиции в своем городе). Затем она отправляет этот запрос в Интернет и ждет ответа.

- **Анализ ответа.** Google возвращает результаты поиска в удобном формате JSON. Скрипт пробегается по найденным ссылкам и ищет специальные микроразметки (pagemap.offer), где сайты обычно указывают цену товара. Как только он находит первый же результат с ценой, он "хватает" ее, а также ссылку на страницу и ее заголовок, и возвращает дирижеру. Если ничего не найдено, он честно сообщает об этом.
- **Запись результатов.** Главная функция получает от помощника найденные данные и аккуратно складывает их в нужные ячейки (G и H). Причем в ячейку G она вставляет не просто ссылку, а красивую кликабельную гиперссылку с названием страницы — все благодаря формуле `=HYPERLINK()`.

Как заставить это работать у себя?

Чтобы этот скрипт ожил в вашей таблице, нужно сделать три вещи: получить "пропуск" для доступа к API Google, вставить их в код и дать скрипту разрешение на работу.

Шаг 1. Получаем ключи (API Key и CSE ID).

Google тщательно контролирует доступ к своим сервисам, поэтому нам нужны два ключа.

1. **API Key (ключ API)** — это ваш универсальный пропуск в мир сервисов Google.
   - » Зайдите в Google Cloud Console *https://console.cloud.google.com/* и создайте новый проект (например, My Price Checker).
   - » Внутри проекта перейдите в раздел **API и сервисы => Библиотека**.
   - » Найдите в поиске **Custom Search API** и включите его.
   - » После этого перейдите в **Учетные данные**, нажмите **Создать учетные данные** и выберите **Ключ API**. Сохраните полученный ключ — он нам понадобится.
2. **CSE ID (ID поисковой системы)** — этот ключ говорит Google, как именно искать. Мы создадим поисковую систему, настроенную на весь Интернет.
   - » Перейдите на страницу Google Programmable Search Engine: *https://programmablesearchengine.google.com/controlpanel/all*

» Нажмите **Добавить**, дайте имя вашей поисковой системе (например, Price Searcher). В настройках выберите опцию **Искать во всем Интернете**.

» После создания системы перейдите в ее панель управления. В разделе **Основная информация** вы найдете **Идентификатор поисковой системы**. Скопируйте его.

Шаг 2. Вставляем ключи в код.

Вернитесь в редактор Apps Script. В самом верху кода найдите блок secrets и вставьте ваши ключи вместо 'ВАШ_API_КЛЮЧ' и 'ВАШ_CSE_ID'. Сохраните проект (иконка дискеты).

Шаг 3. Запускаем и даем разрешения.

» Убедитесь, что в редакторе выбрана функция `processSearchTab`.

» Нажмите на кнопку **Выполнить** (значок треугольника).

» При первом запуске Google попросит вас предоставить разрешения. Это стандартная процедура безопасности. Нажмите **Просмотр разрешений**, выберите свой аккаунт. Вы увидите предупреждение, что приложение не проверено Google (ведь его написали вы сами). Смело нажимайте **Дополнительные настройки**, а затем **Перейти на страницу** (название вашего проекта). На последнем шаге нажмите **Разрешить**. Не бойтесь, что скрипт как-то повредит вам — в любой момент вы можете посмотреть список выданных вами разрешений на специальной странице https://myaccount.google.com/connections и в один клик их отозвать.

Важные нюансы и возможные улучшения

- **Квоты API:** бесплатный тариф Google Custom Search API ограничен 100 запросами в день. Для личных нужд этого часто хватает. Если нужно больше, можно создать несколько проектов Google Cloud с разными ключами и научить скрипт переключаться между ними.
- **Надежность данных:** скрипт полагается на то, как сайты структурируют информацию о ценах. Он не всегда может найти цену, даже если она есть на странице. Поэтому ручная проверка самых важных позиций все равно необходима.

- **Гибкость:** вы можете легко адаптировать скрипт под себя. Например, изменить `initial_phrase`, чтобы искать цены в другом регионе, или добавить парсинг дополнительных данных, если они есть в ответе API.

Этот пример — лишь вершина айсберга возможностей Apps Script. Он превращает обычную Google Таблицу из пассивного хранилища данных в активный инструмент, который самостоятельно выполняет рутинную работу, экономя ваше время и силы для более творческих и важных задач.

Заключение

В этой главе мы совершили настоящую революцию в работе с таблицами. Мы превратили статичные "фотографии" данных в "живые" потоки, которые обновляются сами. Вы прошли путь от одной простой формулы до создания полноценных мини-программ, которые выполняют за вас рутинную работу.

> Главный вывод: **для каждой задачи есть свой лучший инструмент**. Иногда достаточно быстрой формулы, иногда нужен мощный Power Query, а для самых сложных случаев у вас теперь есть гибкость скриптов. Вы научились мыслить, как автоматизатор: не "я сделаю", а "как сделать, чтобы оно делалось само".

Вы получили мощнейший арсенал для работы с Excel и Google Таблицами. Но это только начало. Таблицы великолепны, но, когда дело доходит до анализа больших объемов данных, создания сложных приложений или интеграции с тысячами других сервисов, их возможностей может не хватать.



Глава 2.

Магия одного клика: работа с документами и файлами

В предыдущей главе мы превратили наши таблицы в "живые организмы", которые сами собирают данные. Данные сами по себе — это еще не продукт, а лишь сырье. Их ценность проявляется только тогда, когда они превращаются в документы, отчеты и таблицы, которые можно использовать в работе. Представьте цех: в одном углу лежат заготовки, в другом — готовые детали. Между ними стоят станки, которые превращают одно в другое.

В этой главе мы посмотрим, как менялись такие "станки" для работы с документами.

- **Сначала** — Word, который напоминает ручной "станок": прост, надежен, но требует много ручного труда.
- **Потом** — Google Apps Script, уже "фабрика с конвейером", где процессы идут автоматически и в потоке.
- **Наконец** — Python, универсальный "инструмент инженера": с ним вы сами решаете, какой цех выбрать, и не зависите от чужих ограничений.

2.1. Word — ручной станок для документов

Microsoft Word — настоящий "дедушка" автоматизации. Это ручной станок, на котором еще можно наладить мелкосерийное производство. Нужен комплект одинаковых документов? Word справится: вы загружаете заготовки (таблицу с данными), подключаете форму-шаблон — и получаете пачку готовых писем, наклеек или договоров.

Но, как у любого станка, есть ограничения. Он требует присутствия человека: нужно щелкать кнопки, открывать меню, подготавливать шаблон. Производительность ограничена — сотни документов можно сделать, тысячи уже проблематично. И станок жестко привязан к одному цеху — вашему компьютеру с установленным Word.

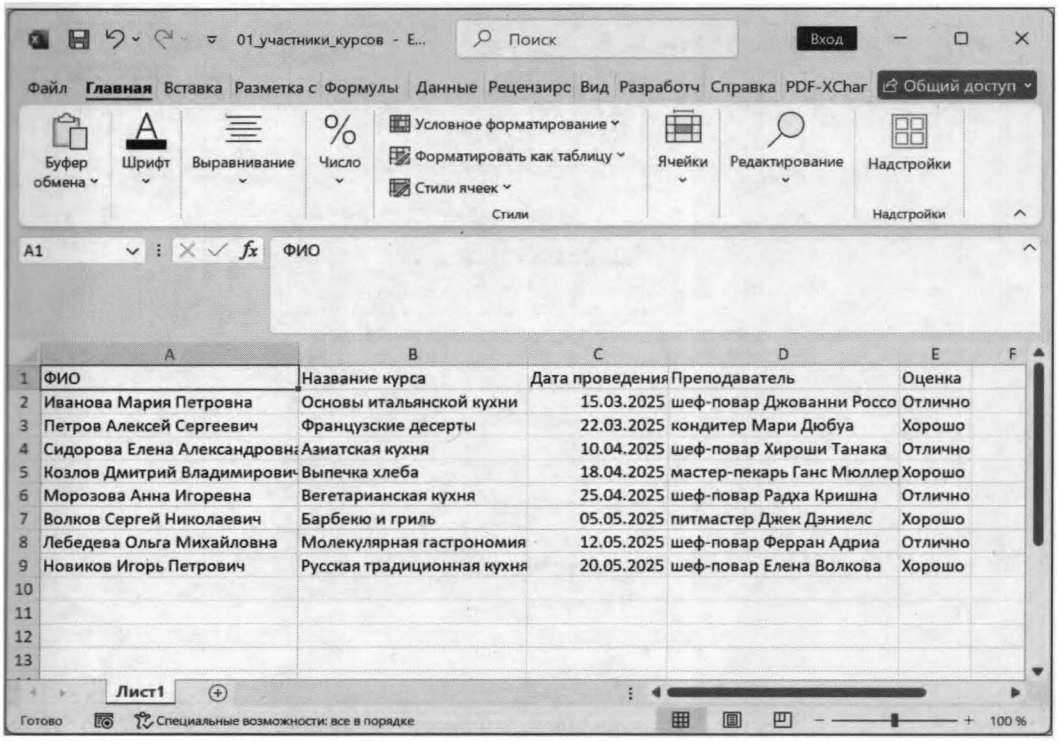
2.1.1. Простой пример: генерация именных сертификатов для участников кулинарных курсов

Сценарий. Провели серию мастер-классов, и нужно выдать сертификаты участникам: 100 человек, каждому — персонализированный документ с ФИО, названием курса и датой.

Источник данных

Подготовьте простую Excel-таблицу со столбцами (рис. 2.1):

- `ФИО` — обязательно в одной колонке, так, как вы хотите, чтобы оно отображалось на сертификате;
- `Название курса` — строка, может содержать подзаголовок;
- `Дата проведения` — предпочтительно в формате ДД.ММ.ГГГГ (ДД — день, число от 01 до 31; ММ — месяц, число от 01 до 12; ГГГГ — год, четыре цифры, например, 2026) или в виде строки, чтобы Word не менял стиль.



	A	B	C	D	E
1	ФИО	Название курса	Дата проведения	Преподаватель	Оценка
2	Иванова Мария Петровна	Основы итальянской кухни	15.03.2025	шеф-повар Джованни Россо	Отлично
3	Петров Алексей Сергеевич	Французские десерты	22.03.2025	кондитер Мари Дюбуа	Хорошо
4	Сидорова Елена Александровна	Азиатская кухня	10.04.2025	шеф-повар Хироши Танака	Отлично
5	Козлов Дмитрий Владимирович	Выпечка хлеба	18.04.2025	мастер-пекарь Ганс Мюллер	Хорошо
6	Морозова Анна Игоревна	Вегетарианская кухня	25.04.2025	шеф-повар Радха Кришна	Отлично
7	Волков Сергей Николаевич	Барбекю и гриль	05.05.2025	питмастер Джек Дэниелс	Хорошо
8	Лебедева Ольга Михайловна	Молекулярная гастрономия	12.05.2025	шеф-повар Ферран Адриа	Отлично
9	Новиков Игорь Петрович	Русская традиционная кухня	20.05.2025	шеф-повар Елена Волкова	Хорошо
10					
11					
12					
13					

Рис. 2.1. ***Подготовленная таблица с участниками***

Советы по таблице:

- У первой строки должны быть заголовки (именно они станут именами полей в Word).
- Удалите лишние пробелы и скрытые символы (функция `СЖПРОБЕЛЫ` в Excel помогает).
- Сохраните файл в .xlsx (файл для программы Microsoft Excel или ее аналогов) или .csv (простой текстовый файл, который хранит только данные, разделенные запятыми или другими символами). Если используете CSV — убедитесь в правильной кодировке (UTF-8 с BOM для совместимости Windows).

Шаблон

Создайте в Microsoft Word аккуратный .docx-шаблон сертификата: логотип, рамка, место для подписи и текст вроде "Настоящий сертификат подтверждает, что <<ФИО>> прошел курс <<Название курса>> от <<Дата проведения>>" (рис. 2.2). Вставьте плейсхолдеры там, где нужна персонализация, — Word заменит их значениями из Excel.

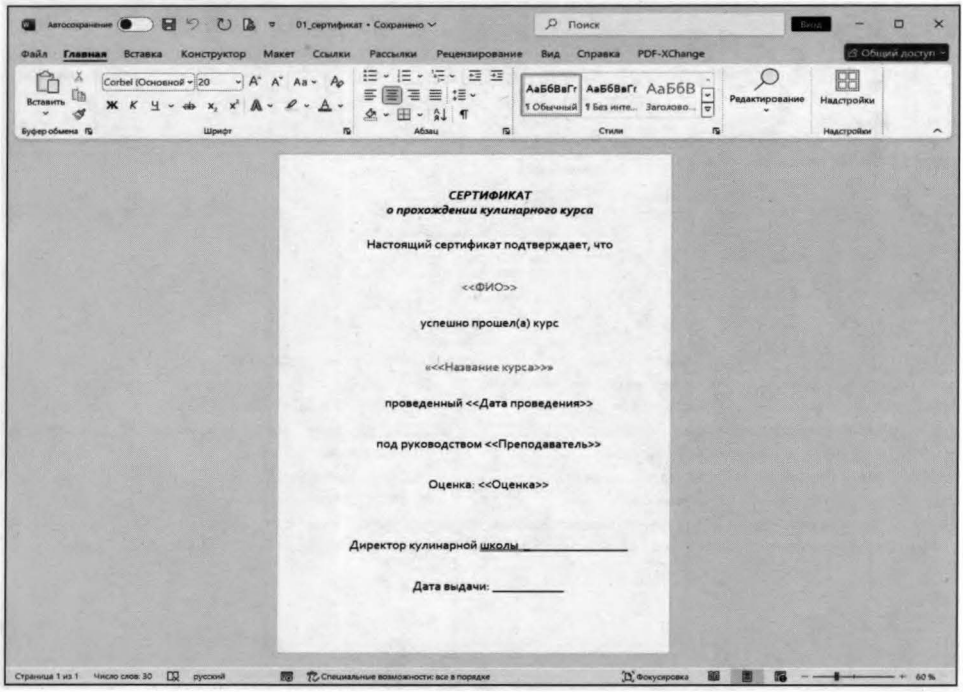


Рис. 2.2. ***Подготовленный простой шаблон***

Советы по шаблону:

- Используйте таблицы или текстовые поля для точного позиционирования текста на сертификате.
- Если документ будет печататься — ориентируйтесь на реальный размер бумаги и поля принтера.
- Заблокируйте шаблон (сохраните копию без полей) на случай ошибок.

2.1.2. Процесс: пошаговая инструкция

1. Откройте шаблон в Microsoft Word.
2. Перейдите на вкладку **Рассылки (рис. 2.3)**.

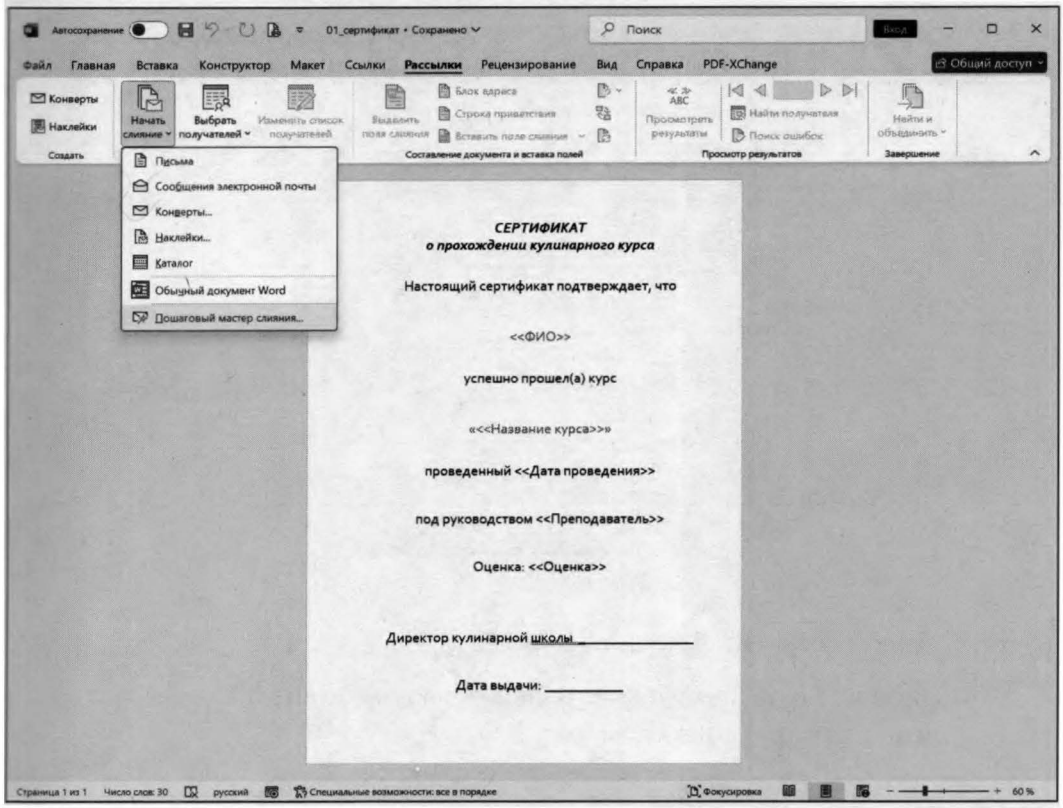


Рис. 2.3. ***Вкладка "Рассылки" в Word. Поля из базы данных придется добавлять повторно***

3. Нажмите **Начать слияние => Пошаговый мастер слияния...** — это удобный режим для первого раза (рис. 2.4).

4. В мастере выберите тип документа — **Письма**.

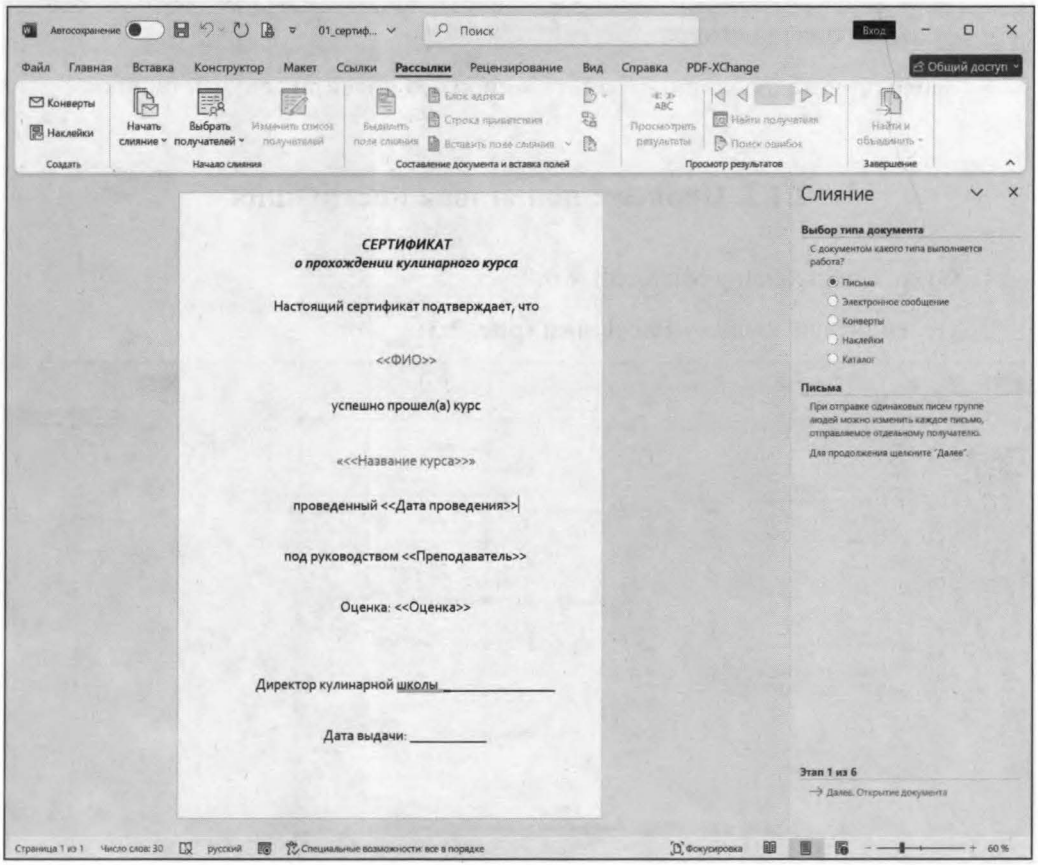


Рис. 2.4. ***Выбор типа слияния***

5. Выберите документ: **Текущий документ.**

6. Подключите источник данных: **Выбрать получателей => Использовать список** и укажите файл Excel (рис. 2.5).

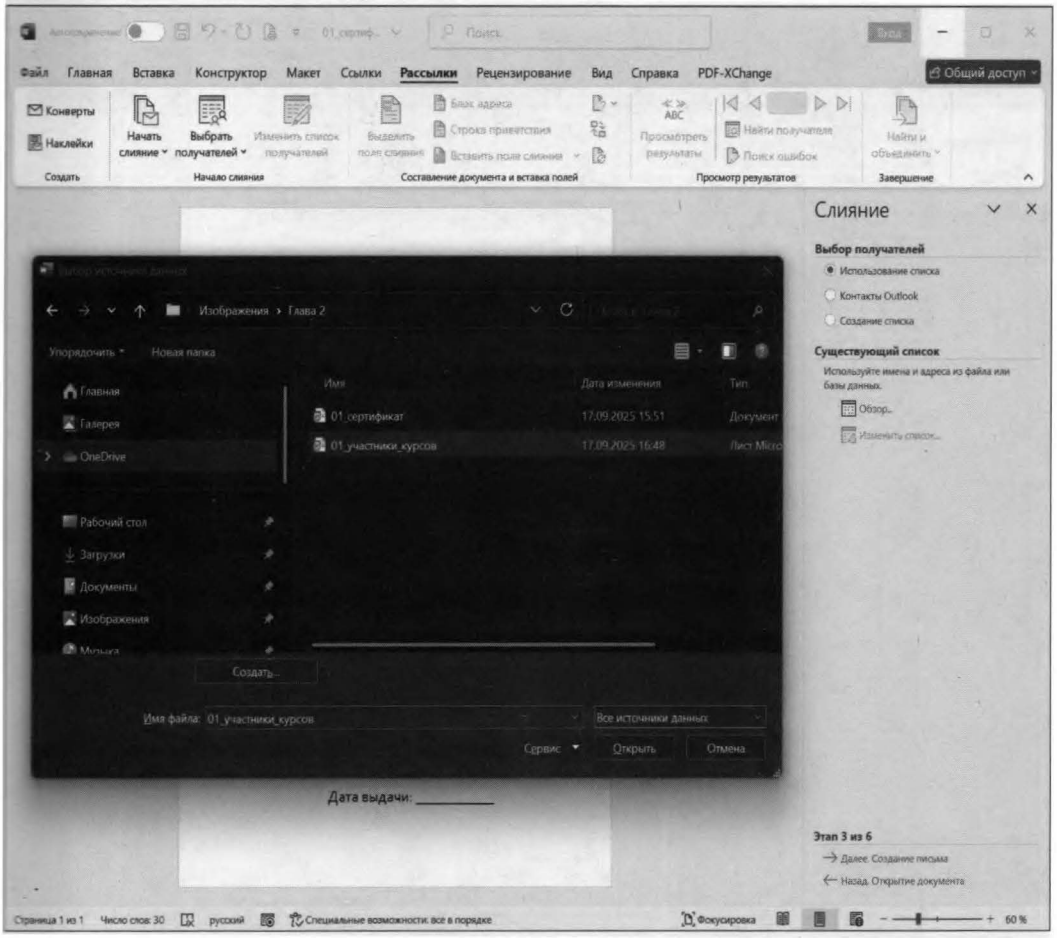


Рис. 2.5. ***Окно выбора файла Excel***

7. После подключения выберите лист и диапазон, если нужно. Word подхватит заголовки столбцов как поля.

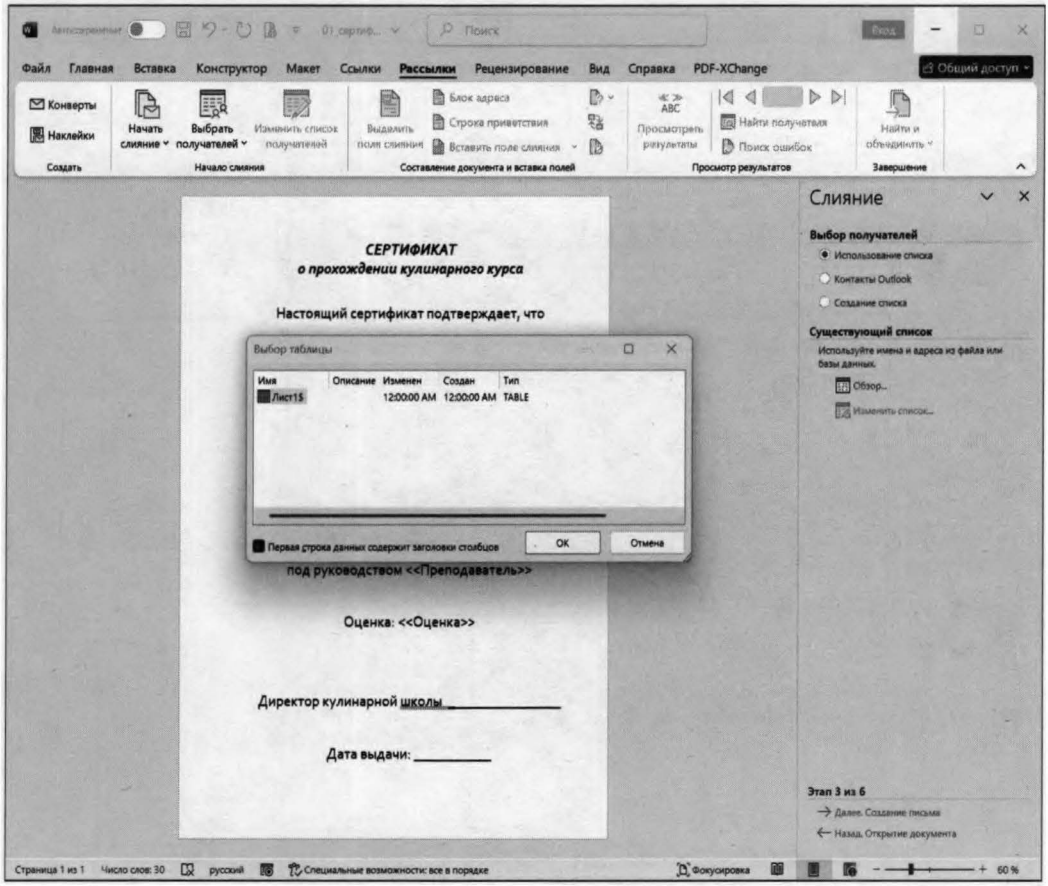


Рис. 2.6. ***Выбор листа Excel***

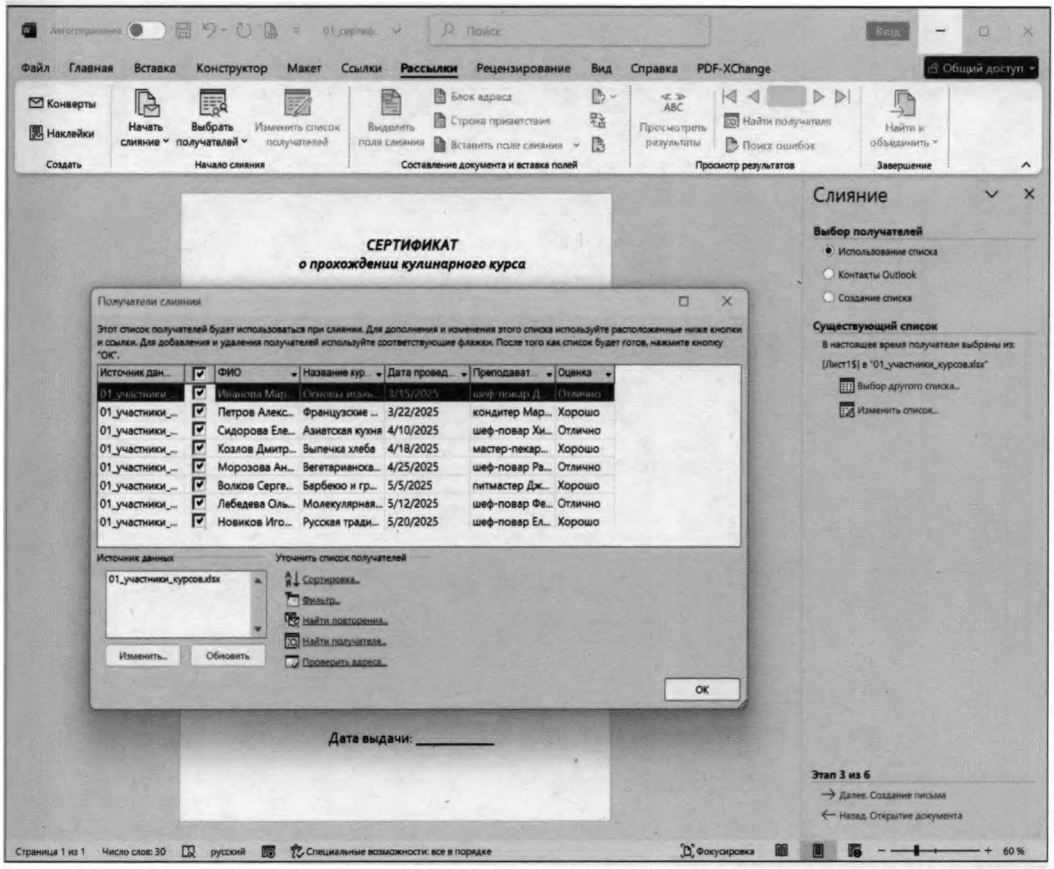


Рис. 2.7. ***Проверка данных***

8. В шаблоне нажмите **Вставить поле слияния** и выберите, например, `ФИО`, `Название курса`, `Дата проведения`. Разместите поля в нужных местах (рис. 2.7).

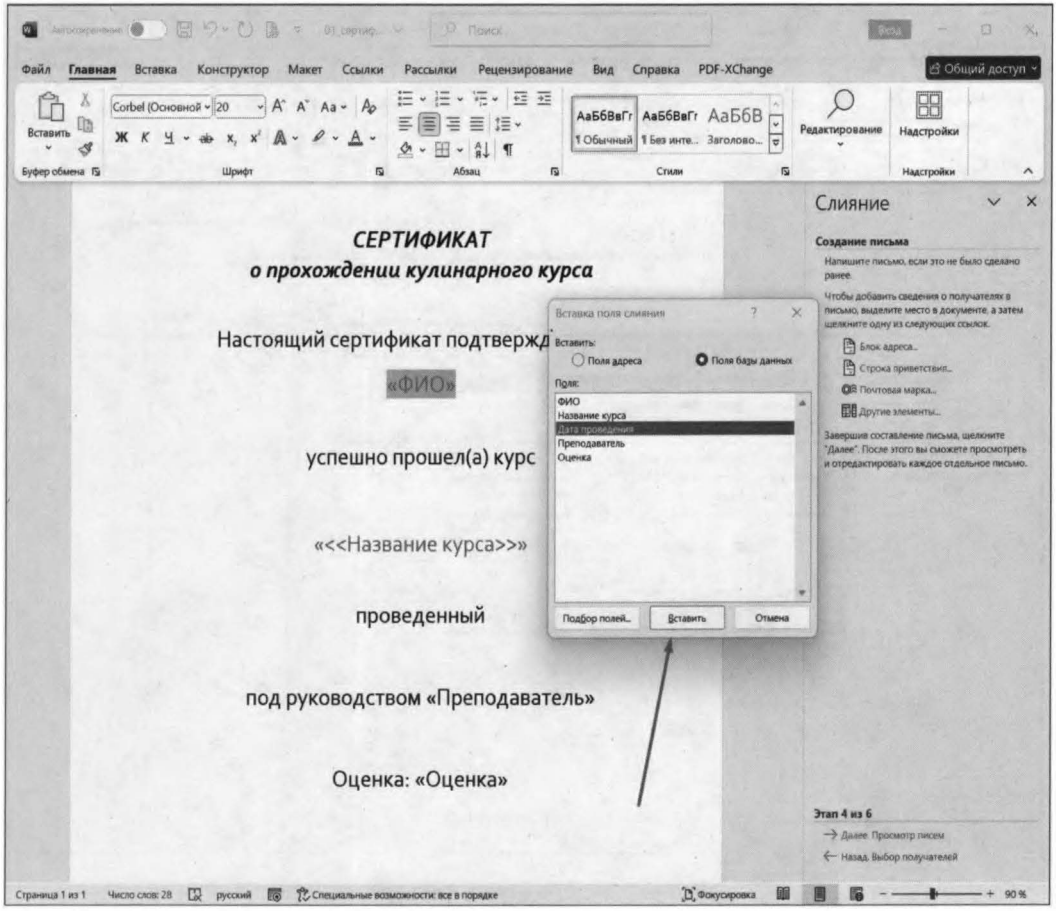


Рис. 2.8. ***Поля слияния в шаблоне***

9. Проверьте превью — кнопка **Просмотреть результаты** покажет, как будут выглядеть реальные сертификаты. Переключайтесь между записями, чтобы убедиться в корректности (рис. 2.8).

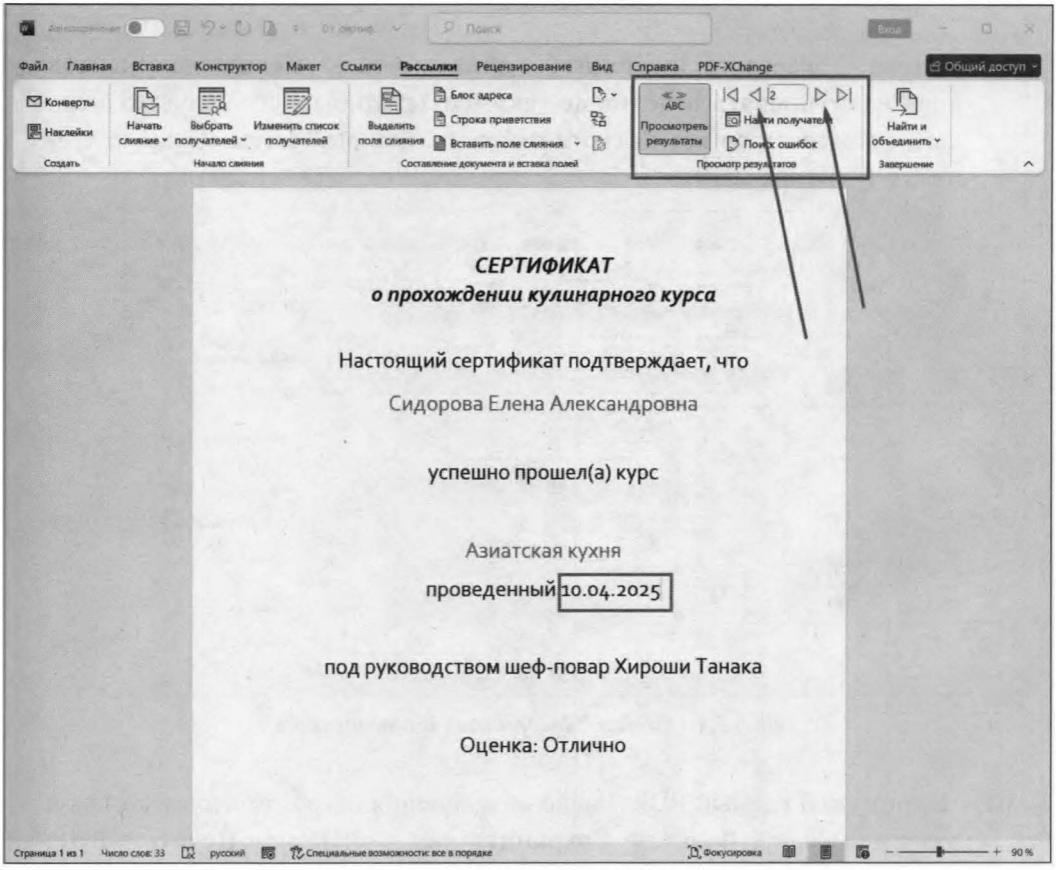


Рис. 2.9. ***Предпросмотр сертификатов и проблема с датами***

Если у вас возникает проблема с отображением даты в формате ММ.ДД.ГГГГ вместо ДД.ММ.ГГГГ — это типичная ситуация, связанная с региональными настройками системы. Самый надежный способ — преобразовать даты в текст:

- создайте дополнительный столбец рядом с датами;
- используйте формулу: =ТЕКСТ(C2;"ДД.ММ.ГГГГ"), где C2 — ячейка с исходной датой;
- скопируйте формулу для всех строк;

» скопируйте получившиеся значения и вставьте как "значения" в основной столбец дат;

» удалите вспомогательный столбец.

10. Готово — выберите **Завершить и объединить => Печать документов** или **Редактировать отдельные документы**, чтобы получить один большой файл .docx со всеми сертификатами, который затем можно сохранить в PDF (рис. 2.10).

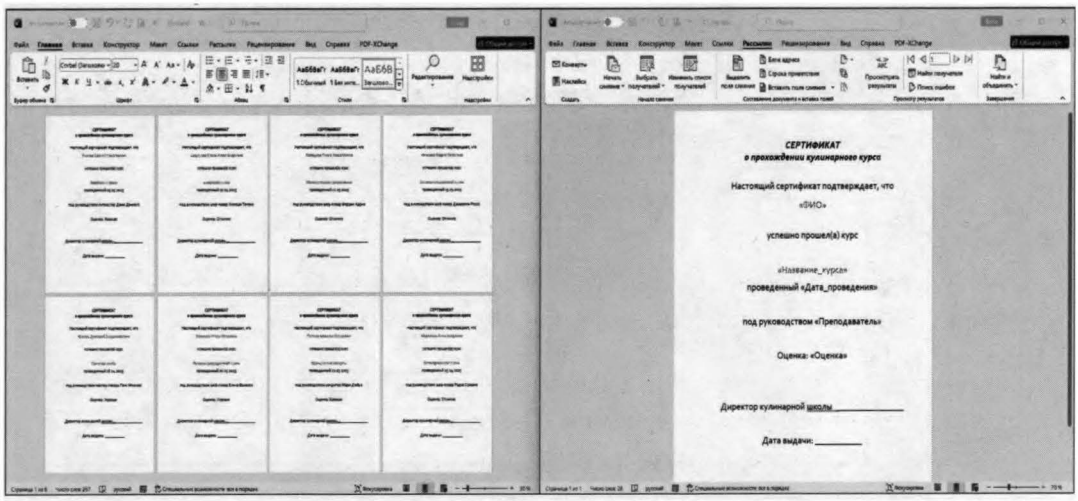


Рис. 2.10. ***Меню "Завершить и объединить"***

11. Если нужен единый PDF: после объединения откройте итоговый документ и выберите **Файл => Сохранить как => PDF** или **Печать в PDF**, чтобы получить один файл с 100 страницами.

2.1.3. Практические хитрости

- Если нужно разный текст в зависимости от значения поля (например, Участник/Лектор), используйте поле `IF` в Word — это простая условная логика прямо в шаблоне.
- Для длинных списков (тысячи записей) Word может тормозить: разбивайте по частям или используйте CSV и внешние инструменты.
- Если даты в Excel обрабатываются некорректно, заранее приведите их к текстовому формату в нужном виде.

- Проведите тест на 3–5 записей перед генерацией всех 100 — это сэкономит кучу времени.

> **Рассылки** — отличный старт для тех, кто хочет получить результат быстро и без кода. Это "дедушка" автоматизации: надежный, понятный и встроенный в инструмент, которым большинство уже пользуется.

Но у метода есть ограничения:

- Сложная логика (вложенные условия, циклы, выбор шаблона по условию) реализуется тяжело и неэлегантно.
- Ограничения по масштабируемости: при больших объемах Word может замедляться.
- Трудно интегрировать в облачные пайплайны и запланированные задачи без дополнительных средств (VBA или скриптов).

Если вам нужен следующий уровень — автоматизация в облаке, сложная логика или интеграция с сервисами (подпись, отправка писем, хранение в облаке) — в следующих разделах мы плавно перейдем к сценариям с VBA, Python и облачными фабриками документов. Но, прежде чем бежать за кодом, — попробуйте Рассылки: это чаще всего решает задачу быстрее и с наименьшими усилиями.

2.2. Облачная фабрика документов с Google Apps Script

Если Word — это ручной станок, на котором все держится на операторе, то **Google Apps Script** — уже целая фабрика. Здесь есть конвейер: данные подаются автоматически, шаблоны документов создаются и заполняются без вашего участия, а результаты складываются прямо в облако.

Главное отличие фабрики в том, что она работает потоком. Нужно сделать сотню договоров? Тысячу сертификатов? Скрипт пробежит по таблице Google Sheets и в пару минут выпустит готовую продукцию — документы

в Google Docs или PDF-файлы. Вам остается только забрать результат или дать доступ коллегам.

2.2.1. Пример: автоматизация договоров дарения

Рассмотрим реалистичный **кейс**: бухгалтерии нужно регулярно готовить однотипные договоры дарения для сотрудников. Ручное копирование ФИО, паспортных данных и сумм из ведомости в шаблон договора — это долго, утомительно и чревато ошибками. Наша задача — автоматизировать процесс так, чтобы пачка готовых договоров создавалась за один клик.

Разбор решения

1. Подготовка: таблица-источник и документ-шаблон.

Первым шагом мы готовим "сырье" и "форму для отливки".

- **Источник данных (Google Таблица):** создаем таблицу, где каждая строка — это один будущий договор, а каждый столбец — переменная (рис. 2.11). Заголовки столбцов должны быть понятными: Дата, ФИО, Должность, Сумма, Подарок, Паспорт, ИНН. Эта таблица становится нашим единым центром управления.

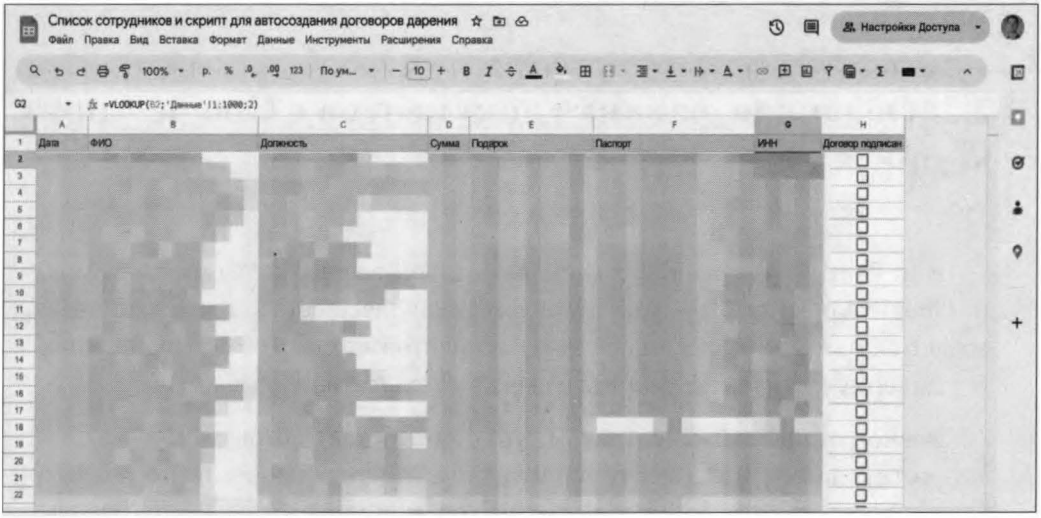


Рис. 2.11. ***Таблица с исходными данными***

- **Шаблон (Google Документ):** создаем стандартный договор в Google Docs со всем необходимым форматированием: логотипы, реквизиты, юридический текст. В местах, куда должны подставляться данные из таблицы, мы размещаем плейсхолдеры — уникальные метки (рис. 2.12, 2.13). Удобнее всего использовать двойные фигурные скобки, например: {{ФИО}}, {{Должность}}, {{Сумма}}. Эти метки легко идентифицировать, и скрипт будет находить и заменять их.

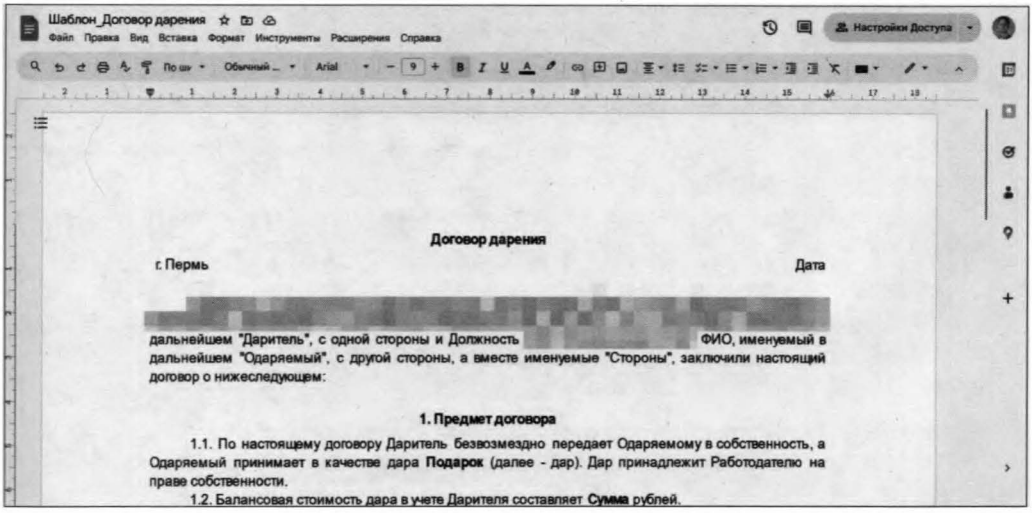


Рис. 2.12. ***Верхняя часть однотипного шаблона документа Google***

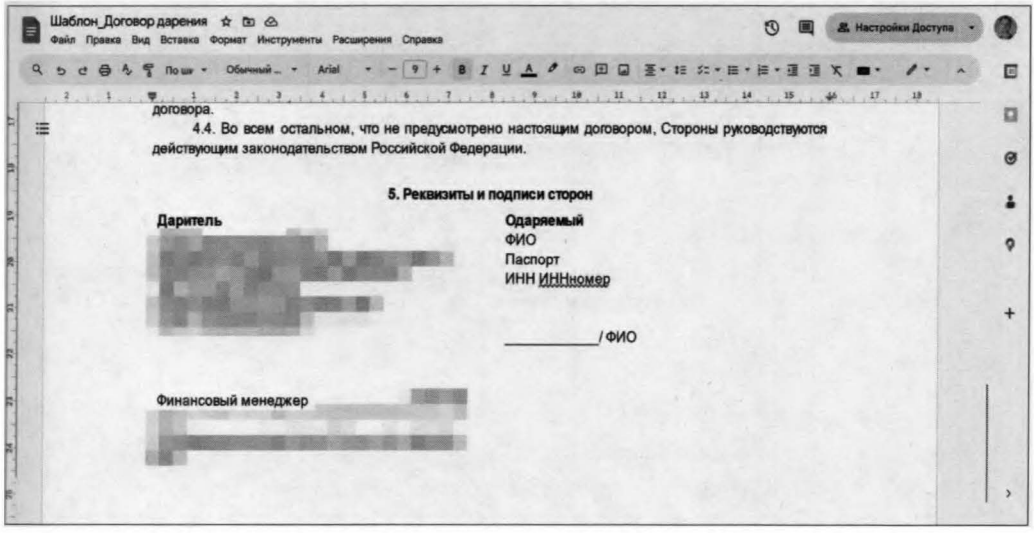


Рис. 2.13. ***Нижняя часть однотипного шаблона документа Google***

2. Анатомия скрипта: как это работает "под капотом".

Теперь самое интересное — код, который приводит механизм в движение. Чтобы добавить его, в меню Google Таблицы выберите **Расширения => Apps Script**. Откроется редактор, куда нужно вставить скрипт. Разберем его ключевые команды на простом языке.

```
function generateDocumentsFromSpreadsheet() {
    console.log('Функция generateDocumentsFromSpreadsheet начала работу
в ${(new Date()).toLocaleString("ru-RU")}');
    // Доступ к таблице и получение данных
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    const sheet = ss.getSheetByName("Автозаполнение"); // Измените на
название вашего листа
    const data = sheet.getDataRange().getValues(); // Получаем все
данные с таблицы


    // Открытие шаблона Google Документа
    const templateDocId = "1y2XQhZBXXXXXXXXXXXXXzyfnXh8yUQ"; // Измените
на настоящий идентификатор вашего шаблона
    const templateDoc = DriveApp.getFileById(templateDocId);
    const templateFolder = templateDoc.getParents().next(); // Получаем
папку, в которой находится шаблон


    //  Проходим по каждой строке (начиная со второй, чтобы пропустить
заголовки)
    for (let i = 1; i < data.length; i++) {
        const [rawDate, fullName, position, rawAmount, gift, passport,
tin] = data[i]; // Получаем данные из строки таблицы
        console.log('Работаем с ${fullName}.');


        let formattedAmount = '';
        formattedAmount = rawAmount + " (" + FloatToSamplesInWordsRus
(rawAmount) + ")" // Форматируем сумму как текст (включая пропись)


        // Форматируем дату
        let formattedDate = '';
        if (rawDate instanceof Date) {
            formattedDate = Utilities.formatDate(rawDate, Session.
```

```
getScriptTimeZone(), "dd.MM.yyyy"); // Преобразуем дату в формат "дд.мм.гггг"
        }

        // Создание нового документа на основе шаблона
        const copyDoc = templateDoc.makeCopy('Заполненный -
${fullName}'); // Копируем шаблон и называем новый документ по ФИО
        const copyDocId = copyDoc.getId();

        // Открываем новый документ
        const copyDocAsDoc = DocumentApp.openById(copyDocId);
        const copyDocBody = copyDocAsDoc.getBody();

        // Замена шаблонных полей данными из таблицы
        copyDocBody.replaceText('Дата', formattedDate);
        copyDocBody.replaceText('ФИО', fullName);
        copyDocBody.replaceText('Должность', position);
        copyDocBody.replaceText('Сумма', formattedAmount);
        copyDocBody.replaceText('Подарок', gift);
        copyDocBody.replaceText('Паспорт', passport);
        copyDocBody.replaceText('ИННномер', tin);
        // Замена текста полей соответствующими данными

        // Сохранение и закрытие заполненного документа
        copyDocAsDoc.saveAndClose();

        // Преобразование документа в PDF, если надо
        // const pdfBlob = DriveApp.getFileById(copyDocId).
getAs("application/pdf");

        // Сохранение копии PDF на Google Диске
        // const pdfFile = templateFolder.createFile(pdfBlob).
setName('Заполненный - ${fullName}.pdf');

        // Дополнительно: удалите промежуточный Google Документ, если
нужен только PDF
        // DriveApp.getFileById(copyDocId).setTrashed(true);
    }
```

```
    console.log('Функция generateDocumentsFromSpreadsheet закончила
работу в ${(new Date()).toLocaleString("ru-RU")}');

}

// Карта для преобразования чисел в слова (рубли) отсюда https://habr.
com/ru/articles/104057/
var mapNumbers = {
    0: [2, 1, "ноль"],
    1: [0, 2, "один", "одна"],
    2: [1, 2, "два", "две"],
    3: [1, 1, "три"],
    4: [1, 1, "четыре"],
    5: [2, 1, "пять"],
    6: [2, 1, "шесть"],
    7: [2, 1, "семь"],
    8: [2, 1, "восемь"],
    9: [2, 1, "девять"],
    10: [2, 1, "десять"],
    11: [2, 1, "одиннадцать"],
    12: [2, 1, "двенадцать"],
    13: [2, 1, "тринадцать"],
    14: [2, 1, "четырнадцать"],
    15: [2, 1, "пятнадцать"],
    16: [2, 1, "шестнадцать"],
    17: [2, 1, "семнадцать"],
    18: [2, 1, "восемнадцать"],
    19: [2, 1, "девятнадцать"],
    20: [2, 1, "двадцать"],
    30: [2, 1, "тридцать"],
    40: [2, 1, "сорок"],
    50: [2, 1, "пятьдесят"],
    60: [2, 1, "шестьдесят"],
    70: [2, 1, "семьдесят"],
    80: [2, 1, "восемьдесят"],
    90: [2, 1, "девяносто"],
    100: [2, 1, "сто"],
    200: [2, 1, "двести"],
```

```
    300: [2, 1, "триста"],
    400: [2, 1, "четыреста"],
    500: [2, 1, "пятьсот"],
    600: [2, 1, "шестьсот"],
    700: [2, 1, "семьсот"],
    800: [2, 1, "восемьсот"],
    900: [2, 1, "девятьсот"]
};

var mapOrders = [{
        _Gender: true,
        _arrStates: ["рубль", "рубля", "рублей"]
    },
    {
        _Gender: false,
        _arrStates: ["тысяча", "тысячи", "тысяч"]
    },
    {
        _Gender: true,
        _arrStates: ["миллион", "миллиона", "миллионов"]
    },
    {
        _Gender: true,
        _arrStates: ["миллиард", "миллиарда", "миллиардов"]
    },
    {
        _Gender: true,
        _arrStates: ["триллион", "триллиона", "триллионов"]
    }
];

var objKop = {
    _Gender: false,
    _arrStates: ["копейка", "копейки", "копеек"]
};

function Value(dVal, bGender) {
    var xVal = mapNumbers[dVal];
```

```
    if (xVal[1] == 1) {
        return xVal[2];
    } else {
        return xVal[2 + (bGender ? 0 : 1)];
    }
}

function From0To999(fValue, oObjDesc, fnAddNum, fnAddDesc) {
    var nCurrState = 2;
    if (Math.floor(fValue / 100) > 0) {
        var fCurr = Math.floor(fValue / 100) * 100;
        fnAddNum(Value(fCurr, oObjDesc._Gender));
        nCurrState = mapNumbers[fCurr][0];
        fValue -= fCurr;
    }

    if (fValue < 20) {
        if (Math.floor(fValue) > 0) {
            fnAddNum(Value(fValue, oObjDesc._Gender));
            nCurrState = mapNumbers[fValue][0];
        }
    } else {
        var fCurr = Math.floor(fValue / 10) * 10;
        fnAddNum(Value(fCurr, oObjDesc._Gender));
        nCurrState = mapNumbers[fCurr][0];
        fValue -= fCurr;

        if (Math.floor(fValue) > 0) {
            fnAddNum(Value(fValue, oObjDesc._Gender));
            nCurrState = mapNumbers[fValue][0];
        }
    }

    fnAddDesc(oObjDesc._arrStates[nCurrState]);
}

function FloatToSamplesInWordsRus(fAmount) {
    var fInt = Math.floor(fAmount + 0.005);
```

```
    var fDec = Math.floor(((fAmount - fInt) * 100) + 0.5);

    var arrRet = [];
    var iOrder = 0;
    var arrThousands = [];
    for (; fInt > 0.9999; fInt /= 1000) {
        arrThousands.push(Math.floor(fInt % 1000));
    }
    if (arrThousands.length == 0) {
        arrThousands.push(0);
    }

    function PushToRes(strVal) {
        arrRet.push(strVal);
    }

    for (var iSouth = arrThousands.length - 1; iSouth >= 0; --iSouth) {
        if (arrThousands[iSouth] == 0) {
            continue;
        }
        From0To999(arrThousands[iSouth], mapOrders[iSouth], PushToRes,
PushToRes);
    }

    if (arrThousands[0] == 0) {
        //  Handle zero amount
        if (arrThousands.length == 1) {
            PushToRes(Value(0, mapOrders[0]._Gender));
        }

        var nCurrState = 2;
        PushToRes(mapOrders[0]._arrStates[nCurrState]);
    }

    if (arrRet.length > 0) {
        // Capitalize first letter
        arrRet[0] = arrRet[0].match(/^(.)/)[1].toLocaleUpperCase() +
arrRet[0].match(/^.(.*)$/)[1];
```

```
  }

    arrRet.push((fDec < 10) ? ("0" + fDec) : ("" + fDec));
    From0To999(fDec, objKop, function () {}, PushToRes);

    return arrRet.join(" ");
}
```

- `SpreadsheetApp.getActiveSpreadsheet()` — эта команда говорит скрипту: "Работай с той таблицей, которая сейчас открыта". Это делает код универсальным.
- `DriveApp.getFileById()` — у каждого файла на Google Диске есть уникальный идентификатор (вы видите его в адресной строке браузера). Эта функция позволяет скрипту безошибочно найти наш шаблон, где бы он ни находился.
- Цикл `for` — это сердце автоматизации. Скрипт последовательно "пробегает" по каждой строке таблицы (начиная со второй, чтобы проигнорировать заголовки) и выполняет для нее одинаковый набор действий.
- `templateDoc.makeCopy()` — ключевой принцип: мы никогда не трогаем исходный шаблон. Для каждой строки данных создается его точная копия, которой сразу присваивается осмысленное имя, например: "Договор дарения — Иванов И. И.".
- `copyDocBody.replaceText()` — а вот и сама магия. Эта команда находит в тексте созданной копии плейсхолдер (например, {{ФИО}}) и заменяет его значением из соответствующей ячейки текущей строки (fullName).
- Особый акцент — `FloatToSamplesInWordsRus`, которая делает скрипты на порядок мощнее простого слияния. Во многих юридических документах требуется дублировать сумму прописью. Стандартными инструментами это сделать невозможно. Но с помощью кода мы можем подключить или написать функцию любой сложности. `FloatToSamplesInWordsRus` берет число (например, 15000) и превращает его в текст "Пятнадцать тысяч рублей 00 копеек" с учетом всех правил русского языка. Это показывает главное преимущество Apps Script: возможность реализовать любую бизнес-логику, будь то сложные вычисления, форматирование данных или, как здесь, лингвистические преобразования.

3. Результат: папка готовых документов за один запуск.

После того как скрипт написан и сохранен, достаточно нажать кнопку **Выполнить** в редакторе. Apps Script запросит разрешения на доступ к вашим Таблицам и Документам (это нужно сделать только один раз). После этого он за несколько секунд обработает все строки и создаст в той же папке, где лежит шаблон, аккуратно именованные и полностью заполненные договоры. При необходимости скрипт можно легко доработать, чтобы он сразу сохранял документы в формате PDF.

2.2.2. Apps Script — "швейцарский нож" для автоматизации в Google

Этот пример — лишь верхушка айсберга.

Google Apps Script — это мощнейший и при этом бесплатный инструмент для автоматизации документооборота и других рутинных задач в экосистеме Google.

Освоив базовый принцип "данные в Таблице => шаблон в Документе => скрипт-соединитель", вы сможете применять его для самых разных целей, например:

- **Генерация счетов и актов** — автоматическое создание пакета закрывающих документов для клиентов по итогам месяца.
- **Персонализированные письма** — создание индивидуальных коммерческих предложений или писем для HR.
- **Сертификаты и дипломы** — массовая генерация наградных документов для участников курсов или соревнований.
- **Отчеты** — автоматическое формирование еженедельных отчетов по стандартной форме на основе данных из аналитической таблицы.

Переход от Word к Google Apps Script — это шаг от простой подстановки данных к созданию умных, гибких и полностью автоматических "фабрик документов", работающих в облаке.

2.3. Python — универсальный инструмент инженера

Этот раздел посвящен второй частой проблеме — файловому хаосу, огда у вас не один шаблон, а сотни или тысячи файлов, с которыми нужно что-то сделать.

2.3.1. Простой пример: скрипт-сортировщик на Python

Word — это станок, Google Apps Script — фабрика. Но и станок, и фабрика работают в заданных условиях: первый привязан к вашему компьютеру, вторая — к облаку Google. А что, если вам нужен инструмент, который можно применить к любой задаче?

Таким инструментом становится Python. Представьте чемодан инженера: в нем набор отверток, ключей и даже мини-станки. С их помощью можно собрать что угодно — от простой полки до целого станочного парка. Python именно такой: он работает и с документами, и с таблицами, и с базами данных, и с файлами на жестком диске.

Где особенно полезен Python?

Там, где ни "станок", ни "фабрика" не справятся.

- **Наведение порядка в хаосе файлов**. У вас папка с сотнями договоров и сканов? Python переберет их, переименует, рассортирует по датам или клиентам.
- **Массовая обработка данных**. Нужно из десяти Excel собрать один большой отчет? "станок" и "фабрика" тут бессильны, а Python справится за секунды.
- **Интеграция с любыми системами**. API-брокеры, CRM, домашний NAS — все это можно "подключить" к Python-скрипту.

Проблема: цифровая свалка в папке "Загрузки"

Папка **Downloads** — это, пожалуй, самое хаотичное место на любом компьютере.

В нее по умолчанию валится все: рабочие документы, смешные картинки, архивы с файлами, установочные пакеты программ, билеты на самолет и другое. Спустя пару месяцев она превращается в цифровую "черную дыру", где найти что-либо практически невозможно. Ручная сортировка — дело неблагодарное: долго, скучно, и уже через неделю все возвращается на круги своя.

Решение: "цифровой дворецкий" на Python

Давайте напишем простой скрипт, который будет выполнять работу умного сортировщика. Его задача — заглянуть в папку **Загрузки**, посмотреть на тип каждого файла и аккуратно положить его в соответствующую тематическую папку.

Вот его план действий:

1. Проверить содержимое папки **Downloads**.
2. Создать подпапки, если их еще нет: **Изображения**, **Документы**, **Архивы** и **Прочее** для всего остального.
3. Взять каждый файл, посмотреть на его расширение (например, .jpg, .pdf, .zip).
4. Переместить файл в нужную папку. Картинку — в **Изображения**, документ — в **Документы** и так далее.

Все это происходит за доли секунды и без единого клика мыши с нашей стороны.

Код: 20 строк, которые сэкономят часы

Для этой задачи нам понадобятся две встроенные в Python библиотеки: **pathlib** для удобной работы с путями к файлам и папкам (она избавляет от головной боли с разными слешами в Windows и Linux) и **shutil** (сокращение от *shell utilities*) для высокоуровневых операций, таких как перемещение файлов.

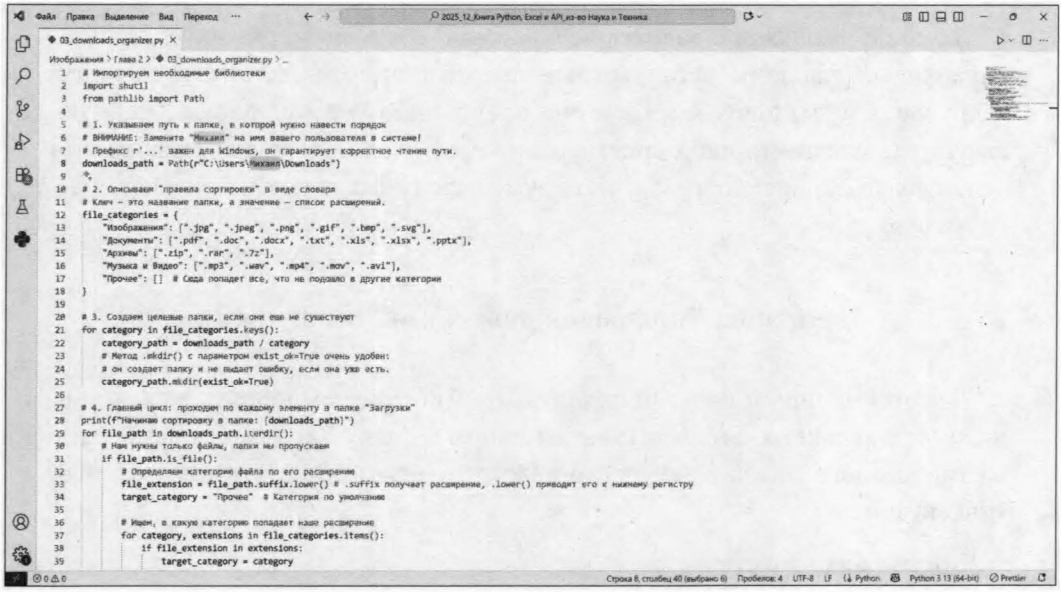


Рис. 2.14. ***Код в Visual Studio Code***

Вот как выглядит сам скрипт. Он короткий, и каждая строка снабжена комментариями, объясняющими, что происходит.

```
# Импортируем необходимые библиотеки
import shutil
from pathlib import Path

# 1. Указываем путь к папке, в которой нужно навести порядок
# ВНИМАНИЕ: Измените "Михаил" на имя вашего пользователя в системе!
# Префикс r'...' важен для Windows, он гарантирует корректное чтение пути.
downloads_path = Path(r"C:\Users\Михаил\Downloads")

# 2. Описываем "правила сортировки" в виде словаря
# Ключ - это название папки, а значение - список расширений.
file_categories = {
    "Изображения": [".jpg", ".jpeg", ".png", ".gif", ".bmp", ".svg"],
    "Документы": [".pdf", ".doc", ".docx", ".txt", ".xls", ".xlsx", ".pptx"],
    "Архивы": [".zip", ".rar", ".7z"],
    "Музыка и Видео": [".mp3", ".wav", ".mp4", ".mov", ".avi"],
    "Прочее": []  # Сюда попадет все, что не подошло в другие категории
```

```
}

# 3. Создаем целевые папки, если они еще не существуют
for category in file_categories.keys():
    category_path = downloads_path / category
    # Метод .mkdir() с параметром exist_ok=True очень удобен:
    # он создает папку и не выдает ошибку, если она уже есть.
    category_path.mkdir(exist_ok=True)

# 4. Главный цикл: проходим по каждому элементу в папке "Загрузки"
print(f"Начинаю сортировку в папке: {downloads_path}")
for file_path in downloads_path.iterdir():
    # Нам нужны только файлы, папки мы пропускаем
    if file_path.is_file():
        # Определяем категорию файла по его расширению
        file_extension = file_path.suffix.lower() # .suffix получает расширение,
.lower() приводит его к нижнему регистру
        target_category = "Прочее"  # Категория по умолчанию

        # Ищем, в какую категорию попадает наше расширение
        for category, extensions in file_categories.items():
            if file_extension in extensions:
                target_category = category
                break # Нашли подходящую категорию, выходим из внутреннего
цикла

        # 5. Перемещаем файл в нужную папку
        target_folder = downloads_path / target_category
        shutil.move(str(file_path), str(target_folder))
        print(f"Файл '{file_path.name}' перемещен в папку '{target_category}'")

print("Сортировка завершена! В папке 'Загрузки' теперь идеальный порядок.")
```

Разбор полетов: как это работает?

Шаги 1 и 2: подготовка. Мы просто говорим скрипту, где работать (downloads_path) и по каким правилам играть (file_categories).

Прелесть такого подхода в том, что вы легко можете адаптировать код под себя: добавить новые папки (например, "Книги" для .epub и .fb2) или расширить существующие списки.

Шаг 3: создание. Скрипт проверяет, есть ли нужные папки, и, если нет, создает их. Это делает его "самодостаточным".

Шаг 4 и 5: магия. Главный цикл `for file_path in downloads_path.iterdir()` — это сердце скрипта. Он "осматривает" каждый файл, определяет его "судьбу" по расширению и с помощью команды `shutil.move()` мгновенно отправляет его в правильное место (рис. 2.15).

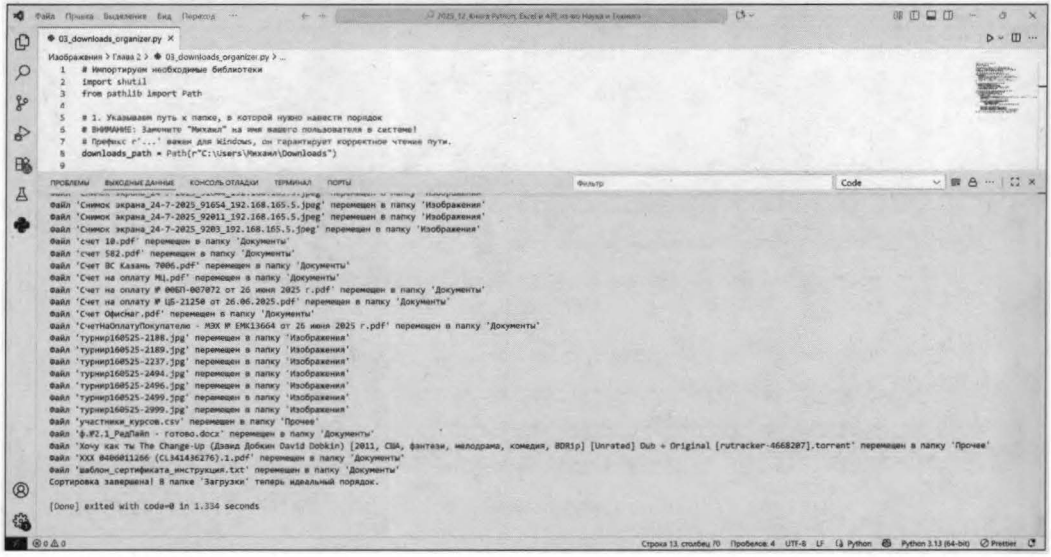


Рис. 2.15. ***Результат выполнения скрипта на моем компьютере: 584 файла обработаны за 1,3 секунды***

Вывод: от простого скрипта к целой системе автоматизации

Этот простой, но невероятно полезный скрипт — идеальная иллюстрация мощи Python для бытовой автоматизации. Вы написали утилиту, которая решает реальную проблему.

Но самое главное — это только начало. Этот скрипт можно улучшать и развивать:

- **Добавить логирование:** записывать в текстовый файл информацию о том, что и куда было перемещено.

- **Обрабатывать вложенные папки:** научить скрипт "заглядывать" в другие папки внутри **Загрузок**.
- **Запускать по расписанию:** с помощью Планировщика заданий в Windows или даже **cron** в macOS/Linux вы можете настроить запуск этого скрипта при включении компьютера. Вы включаете его, а в папке **Загрузки** уже идеальный порядок.

2.3.2. Сложный пример: сбор сводного отчета из сотен Excel-файлов

Если наш Python-скрипт был "цифровым дворецким", который аккуратно раскладывал вещи по полкам, не заглядывая в коробки, то теперь мы перейдем к задачам "цифрового архивариуса". Его работа сложнее: он должен не просто перемещать папки, а вскрывать каждый документ, находить в нем нужные строчки и заносить их в общую ведомость.

Идея: программный рейд по файлам

Представьте себе типичную задачу в бухгалтерии, отделе продаж или у инженеров-сметчиков. У вас есть папка, в которой лежат сотни однотипных файлов: отчеты по продажам за каждый день, акты выполненных работ по десяткам объектов или, как в нашем примере, комплекты сметной документации (рис. 2.16).

Имя	Тип	Размер	Дата
..		<Папка>	12.09.2025 10:56
Ягодная, 24		<Папка>	17.09.2025 08:38
Южная, 90		<Папка>	28.03.2025 07:58
Центральная, 88		<Папка>	28.03.2025 07:57
Трактовая, 23		<Папка>	28.03.2025 07:57
Тепличная, 476		<Папка>	28.03.2025 07:57
Солнечная, 16а		<Папка>	28.03.2025 07:57
Советская, 14а		<Папка>	28.03.2025 07:57
Снежная, 12		<Папка>	28.03.2025 07:57
Северная, 6		<Папка>	28.03.2025 07:57
Садовая, 30		<Папка>	28.03.2025 07:57
Рублевская, 43		<Папка>	28.03.2025 07:57
Подлесная, 9		<Папка>	28.03.2025 07:56
Подлесная, 2		<Папка>	28.03.2025 07:56
Новогодняя, 21		<Папка>	28.03.2025 07:56
Малый, 3		<Папка>	28.03.2025 07:56
Малый, 1		<Папка>	28.03.2025 07:56
Лесная, 11		<Папка>	28.03.2025 07:56
Кедровая, 21а		<Папка>	28.03.2025 07:56
Запрудная, 26		<Папка>	28.03.2025 07:56
Запрудная, 23		<Папка>	28.03.2025 07:56
Дорожная, 18а		<Папка>	28.03.2025 07:56
Встречная, 17		<Папка>	28.03.2025 07:55

Рис. 2.16. ***Часть папок с проектами, из которых надо сделать сводную таблицу с описанием и ценой***

Каждый файл — это отдельный документ Excel.

Задача — собрать из них всех сводный реестр: вытащить из каждого файла, скажем, название объекта и итоговую сумму, чтобы получить общую картину.

Вручную это адский труд: открыть файл, найти глазами ячейку A10, скопировать, вставить в итоговую таблицу, найти ячейку I36, скопировать, вставить, закрыть файл, не сохранив случайных изменений. И так — сотни раз. Это не просто долго, это гарантированный источник ошибок за счет потери концентрации.

Наше решение — научить Excel делать эту работу за нас с помощью скрипта на VBA.

Почему именно VBA?

Мы только что познакомились с мощью Python для работы с файлами.

Так почему бы не использовать его?

Можно, и для некоторых задач это будет правильным решением. Но когда речь идет о глубокой и сложной работе внутри документов Microsoft Office, VBA (Visual Basic for Applications) часто оказывается самым быстрым и нативным путем.

Представьте, что экосистема **Microsoft Office** — это большой завод. **Python** — это мощный внешний подрядчик, которому для работы нужен пропуск и специальные инструменты (библиотеки). А **VBA** — это главный инженер этого завода. Он здесь "свой", знает каждый станок (кнопку, ячейку, лист) и имеет ключи от всех дверей. Для задач, где нужно постоянно открывать, читать и обрабатывать содержимое Excel-файлов, VBA часто работает быстрее и требует меньше подготовительных шагов (рис. 2.17).

	A	B	C	D	E	F	G	H	I	J	K
1							Унифицированная форма № КС-3				
2							Утверждена постановлением Госкомстата России				
3							от 11.11.99 № 100				
4											
5										Код	
6								Форма по ОКУД		0322001	
7								по ОКПО			
8	Заказчик: АО "Газпром газораспределение Пермь", 614000, Пермский край, г.Пермь, ул.Петропавловская,43							по ОКПО			
9	Подрядчик - ООО "Спецмонтажсервис", 614060, Пермский край, г.Пермь, ул.Крупской, 35-121.							по ОКПО			
10	Стройка: «Газопровод-ввод низкого давления для подключения жилого							по ОКПО			
11	дома по адресу: Пермский край, г.Краснокамск, ул. Южная, 90» Код объекта: 159-21-137-33381										
12								Вид деятельности по ОКДП			
13							Договор подряда (контракт)	номер		14-583-ПД/2024	
14								дата	11	10	2024
15								Вид операции			
16											
17					Номер	Дата		Отчетный период			
18					документа	составления		с		по	
19				СПРАВКА	1	24.03.2025		09.01.2025		24.03.2025	
20											
21				О СТОИМОСТИ ВЫПОЛНЕННЫХ РАБОТ И ЗАТРАТ							
22											
23								Стоимость выполненных работ и затрат, руб.			
24	Номер п/п	Наименование пусковых комплексов, этапов, объектов, видов выполненных работ, оборудования, затрат			Код			с начала проведения работ	с начала года	в том числе за отчетный период	
25	1	2			3			4	5	6	
26		Всего работ и затрат, включаемых в стоимость работ в том числе:						53 632,39	53 632,39	53 632,39	
27											
28	1	Акт №1						43 873,68	43 873,68	43 873,68	
29											
30	2	Акт №2						9 758,71	9 758,71	9 758,71	
31											
32											
33											
34									Итого		53 632,39
35									Сумма НДС		10 726,48
36									Всего с учетом НДС		64 358,87
37											
38											
39											
40		Заказчик (генподрядчик)									
41	Директор Филиала в г.Краснокамск АО "Газпром газораспределение Пермь"									Э.В. Жигалов	
42				М П						*(Доверенность №2067 от 19.12.2023г.)*	
43											

Страница 1

Рис. 2.17. ***Из каждого Excel-файла надо взять всего две ячейки***

Разбор скрипта: как заставить Excel работать на себя

Давайте разберем скрипт, который превращает Excel в неутомимого архивариуса. Чтобы его использовать, нужно открыть редактор VBA (Alt + F11 в Excel), вставить код и запустить.

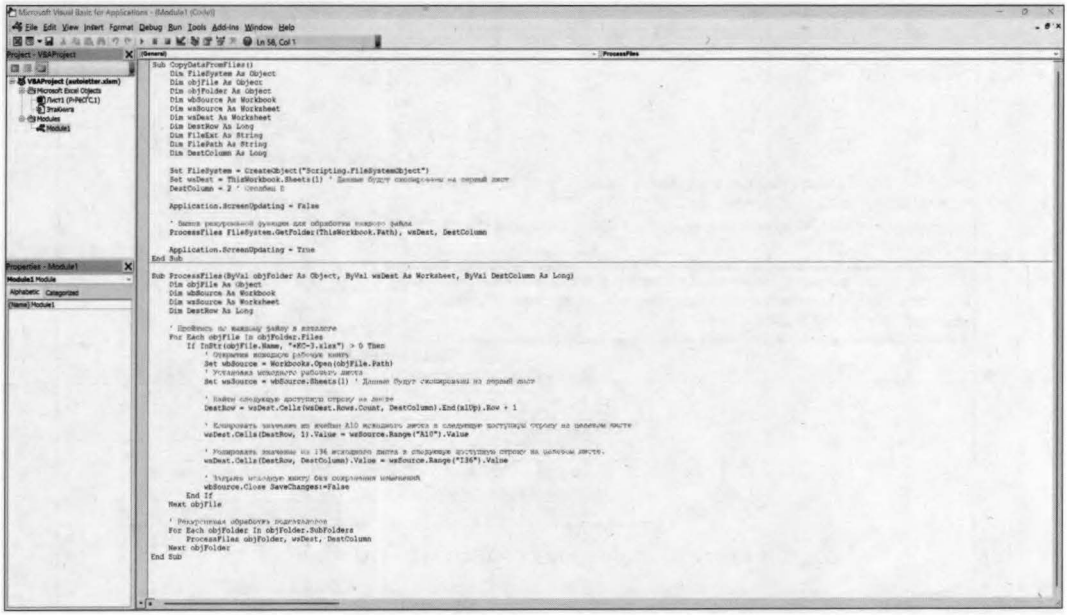


Рис. 2.18. ***Сводная из КС-3***

```
Sub CopyDataFromFiles()
    Dim FileSystem As Object
    Dim objFile As Object
    Dim objFolder As Object
    Dim wbSource As Workbook
    Dim wsSource As Worksheet
    Dim wsDest As Worksheet
    Dim DestRow As Long
    Dim FileExt As String
    Dim FilePath As String
    Dim DestColumn As Long

    Set FileSystem = CreateObject("Scripting.FileSystemObject")
    Set wsDest = ThisWorkbook.Sheets(1) ' Данные будут скопированы на
первый лист
    DestColumn = 2 ' Столбец B

    Application.ScreenUpdating = False
```

```
    ' Вызов рекурсивной функции для обработки каждого файла
    ProcessFiles FileSystem.GetFolder(ThisWorkbook.Path), wsDest,
DestColumn

    Application.ScreenUpdating = True
End Sub

Sub ProcessFiles(ByVal objFolder As Object, ByVal wsDest As Worksheet,
ByVal DestColumn As Long)
    Dim objFile As Object
    Dim wbSource As Workbook
    Dim wsSource As Worksheet
    Dim DestRow As Long

    ' Пройтись по каждому файлу в каталоге
    For Each objFile In objFolder.Files
        If InStr(objFile.Name, "+КС-3.xlsx") > 0 Then
            ' Открытие исходную рабочую книгу
            Set wbSource = Workbooks.Open(objFile.Path)
            ' Установка исходного рабочего листа
            Set wsSource = wbSource.Sheets(1) ' Данные будут скопированы
на первый лист

            ' Найти следующую доступную строку на листе
            DestRow = wsDest.Cells(wsDest.Rows.Count, DestColumn).
End(xlUp).Row + 1

            ' Копировать значение из ячейки A10 исходного листа в
следующую доступную строку на целевом листе
            wsDest.Cells(DestRow, 1).Value = wsSource.Range("A10").Value

            ' Копировать значение из I36 исходного листа в следующую
доступную строку на целевом листе.
            wsDest.Cells(DestRow, DestColumn).Value = wsSource.
Range("I36").Value

            ' Закрыть исходную книгу без сохранения изменений
            wbSource.Close SaveChanges:=False
        End If
```

```
    Next objFile

    ' Рекурсивная обработка подкаталогов
    For Each objFolder In objFolder.SubFolders
        ProcessFiles objFolder, wsDest, DestColumn
    Next objFolder
End Sub
```

Вот его логика:

- **`CreateObject("Scripting.FileSystemObject")` — получаем доступ к файловой системе.**

  По умолчанию VBA "видит" только тот файл, в котором он запущен. Эта команда выдает ему пропуск за пределы одного документа. С ее помощью скрипт возможность "видеть" и "трогать" файлы и папки на вашем компьютере, точно так же как это делал наш Python-скрипт. Это наш первый шаг к выходу за рамки одной таблицы.

- **`ProcessFiles` — рекурсивный обход папок.**

  Это самая умная часть скрипта. Слово "рекурсия" звучит страшно, но идея проста. Мы даем скрипту команду: "Зайди в эту папку и обработай все нужные файлы. А если увидишь внутри другие папки — зайди в каждую из них и сделай то же самое". Скрипт вызывает сам себя для каждой найденной подпапки, пока не обойдет все "закоулки" указанной директории. Это гарантирует, что мы не пропустим ни одного файла, как бы глубоко он ни был спрятан.

- **`If InStr(objFile.Name, "+КС-3.xlsx") > 0` — фильтр по имени файла.**

  Чтобы не открывать вообще все файлы подряд (например, временные файлы Excel или какие-то другие документы), мы устанавливаем фильтр. Эта строчка проверяет, содержит ли имя файла определенный текст — в нашем случае +КС-3.xlsx. Это "фейс-контроль" для файлов: скрипт будет работать только с теми, кто соответствует нашему критерию. Вы легко можете поменять это условие на любое другое, например, искать файлы, начинающиеся со слова "Отчет".

- **`Workbooks.Open(...)` и `wbSource.Close` — аккуратная работа с файлами.**

Здесь происходит основное действие. Скрипт находит нужный файл, открывает его в фоновом режиме (благодаря команде Application. ScreenUpdating = False вы даже не увидите мелькания окон), делает свою работу и немедленно закрывает его без сохранения изменений (SaveChanges:=False). Это критически важно: наш архивариус не оставляет следов и не рискует случайно повредить исходные данные.

- **`wsDest.Cells(DestRow, 1).Value = wsSource.Range("A10").Value` — извлечение данных.**

  Это кульминация всего процесса. Вот та самая магия, ради которой все затевалось. Эта команда говорит: "Возьми значение из ячейки A10 открытого исходного файла (wsSource) и скопируй его в первую колонку (1) следующей свободной строки (DestRow) нашего главного отчета (wsDest)". Следующая строка делает то же самое для ячейки I36 и второго столбца. Скрипт не "смотрит" на весь лист, он точно знает, где лежат нужные ему данные, и забирает только их.

Результат: реестр за минуту

После запуска скрипта вы просто наблюдаете, как на ваших глазах строки итоговой таблицы начинают заполняться одна за другой. Там, где вы бы потратили часы на монотонный копипаст, скрипт справляется за минуту. В результате вы получаете идеально собранный реестр из сотен разрозненных файлов, готовый для анализа, печати или дальнейшей работы.

Вывод: универсальный паттерн пакетной обработки

Самое ценное в этом примере — не решение конкретной задачи с формами КС-3, а сам подход, сам паттерн. Вы только что освоили один из самых мощных приемов автоматизации — пакетную обработку файлов.

Этот шаблон универсален и используется в таких формах, как:

- отчеты о продажах — собрать данные о выручке из ежедневных отчетов, хранящихся в разных файлах;
- анкеты сотрудников — вытащить ФИО и ответы на ключевые вопросы из сотен файлов-анкет;

- технические данные — собрать итоговые показатели из файлов с результатами лабораторных тестов.

Научившись программно "заглядывать" в файлы, извлекать из них крупицы нужной информации и собирать ее в одном месте, вы переходите на новый уровень владения данными. Вы перестаете быть заложником структуры файлов и начинаете управлять информацией напрямую.

Заключение

Мы прошли путь эволюции инструментов автоматизации документов.

- **Word** — ручной станок в цехе. Он прост и надежен, помогает наладить первое "производство", но требует вашего постоянного участия и ограничен масштабом.
- **Google Apps Script** — фабрика в облаке. Она работает потоком, умеет выпускать сотни и тысячи документов без вашего присутствия, но зависит от Интернета и правил чужой площадки.
- **Python** — универсальный инструмент инженера. С его помощью можно строить собственные системы: от маленьких автоматов для сортировки файлов до целых конвейеров, работающих с базами данных, API и облачными сервисами.

"Станок", "фабрика", "универсальный инструмент" — это не конкуренты, а ступени одной лестницы. Начать можно с простого — освоить слияние в Word. Затем перейти к фабрике Google, если задач становится больше. А когда вы почувствуете вкус к настоящей свободе и захотите, чтобы "ваши машины" работали именно так, как нужно вам, — самое время открыть "чемодан инженера".

Глава 3.

Ваш личный секретарь: автоматизируем почту и мессенджеры

Каждый день вы получаете десятки писем и сообщений. Большая часть из них рутинные: счета, уведомления, повторяющиеся вопросы и другие. На ответы уходит время и внимание — а они могли бы работать на автопилоте.

Чтобы показать, как это выглядит в жизни, представим два сценария:

1. **Анна, маркетолог**. Ей приходится отвечать клиентам на одни и те же вопросы о ценах и сроках. Пока она пишет вручную, работа стоит.
2. **Олег, председатель ТСЖ**. Жильцы постоянно напоминают ему про воду, парковку и квитанции. Почта и мессенджеры трещат, нужно отвечать, а на все не хватает рук.

Общее у этих историй одно: рутинная переписка мешает сосредоточиться на важном. Автоматизация снимает этот груз: письма сортируются, дублирующиеся вопросы получают автоответ, а люди вовремя получают напоминания.

От курьера к диспетчеру: автоматизация коммуникаций

И Анна, и Олег сталкиваются с одной проблемой: рутинные коммуникации крадут у них часы жизни. Решение — автоматизация. Только теперь мы автоматизируем не сбор данных или создание документов, а их доставку до получателей.

Представьте, что Анна просто запускает скрипт, который сам читает Excel-файл, генерирует персональные отчеты для каждого региона и отправляет их менеджерам с корректными именами в приветствии. Время выполнения: 50 секунд. Вероятность ошибки: близка к нулю.

Олег настраивает простую систему, которая за пять минут отправляет уведомления всем жильцам через WhatsApp, дублирует анонс ВКонтакте и даже напоминает ему самому о необходимости повесить объявления в подъездах.

Ваша роль кардинально меняется: вместо курьера, который бегает с каждой посылкой, вы становитесь диспетчером, который один раз настраивает маршруты доставки. Компьютер берет на себя рутину, а вы контролируете процесс.

Арсенал цифрового коммуникатора

В этой главе мы рассмотрим три основных канала автоматической коммуникации:

1. **Электронную почту** — классику делового общения. Надежно, официально, поддерживает вложения любого размера. Идеальный выбор для отчетов, договоров, детальных инструкций. Работает везде, принимается всеми, архивируется автоматически. Наш "швейцарский нож" для корпоративной переписки.
2. **Мессенджеры (Telegram, WhatsApp)** — скорость плюс личный контакт. Сообщение доходит мгновенно, читается с телефона, не теряется в почтовом спаме. Telegram порадует нас официальным API, а для WhatsApp мы изучим хитрости автоматизации через веб-интерфейс. Отлично подходят для напоминаний, коротких уведомлений, срочных сообщений.
3. **Социальные сети (ВК)** — инструмент репутации и неформальных связей. Здесь живет ваша аудитория, здесь формируется общественное мнение. Автоматический постинг поможет поддерживать активность без ежедневных заходов в соцсети.

Теперь, когда мы определили три основных канала автоматизации, пора переходить от теории к практике. Мы начнем с самого консервативного и предсказуемого инструмента — электронной почты. Здесь меньше всего подводных камней, а результат можно увидеть сразу. Освоив почтовую автоматизацию, мы получим базовые навыки работы с API, понимание принципов безопасности и уверенность для перехода к более сложным и рискованным инструментам. Электронная почта станет нашим полигоном для отработки автоматизации коммуникаций — безопасным пространством, где можно экспериментировать, не боясь заблокировать аккаунт или испортить репутацию.

3.1. Почтовый автомат без кода

Прежде чем погружаться в код и API, давайте признаем честно: большинству из нас не нужен космический корабль, чтобы добраться до ближайшего магазина. Иногда велосипеда вполне достаточно. В мире автоматизации коммуникаций таким велосипедом становятся встроенные возможности почтовых сервисов и мессенджеров.

Цель этого раздела проста: показать, что базовую автоматизацию можно настроить за пять минут, используя инструменты, которые уже есть у каждого. Никакого программирования, никаких сторонних сервисов – только то, что лежит на поверхности Gmail и Telegram. Это точка входа для самых нетехнических читателей и одновременно фундамент для более сложных решений.

Gmail: фильтры и шаблоны как первая линия обороны

Знакомая проблема: ваша почта превратилась в свалку однотипных писем. Уведомления от интернет-провайдера, счета от управляющей компании, чеки из мобильных приложений, напоминания о продлении подписок — все это перемешано с важной рабочей перепиской. Найти нужное письмо становится квестом, а пропустить важное уведомление — обычным делом.

Простое решение: Gmail уже содержит мощную систему автоматизации — фильтры. Они работают как умные почтальоны: получают письмо, анализируют отправителя и содержимое, а затем выполняют заранее настроенные действия. Чтобы применить фильтр, остается создать его.

Практический пример: создадим "умный" фильтр для автоматической обработки бытовых счетов и уведомлений. Представьте, что вы живете в Перми (как я) и хотите, чтобы все счета автоматически попадали в отдельную папку.

Настраиваем фильтр:

1. **Заходим в настройки Gmail: Шестеренка => Настройки => Фильтры и заблокированные адреса => Создать новый фильтр.**

2. **Создаем "умное" условие отбора:** в поле **От кого** вводим сложное условие:

```
(from:(epd.online OR prg.perm.ru OR krc-prikam.ru OR novogor.perm.
ru OR googleplay-noreply@google.com OR 1trener@trirun.ru OR info@
tykwa.ru OR uk-astra.ru OR echeck@1-ofd.ru OR energos.perm.ru) OR
subject:("УПРАВЛЯЮЩАЯ КОМПАНИЯ" OR "Счет от ПАО "Ростелеком""))
```

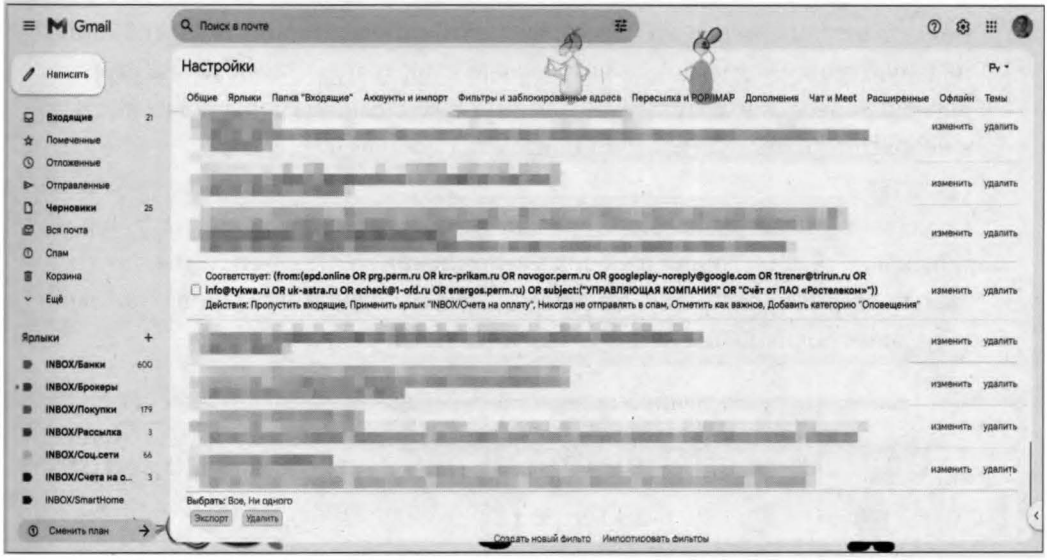


Рис. 3.1. ***Так выглядит настроенный фильтр в интерфейсе Gmail***

Это условие "поймает" письма от:

- коммунальных служб Перми (epd.online, prg.perm.ru);
- управляющих компаний (krc-prikam.ru, novogor.perm.ru);
- интернет-провайдеров и мобильных операторов;
- Google Play (для счетов за подписки);
- фитнес-клубов и других сервисов;
- любые письма с ключевыми словами в теме (рис. 3.1).

В вашем городе это будут другие адреса, но принцип тот же: соберите их один раз, и фильтр будет работать годами.

3. **Настраиваем автоматические действия:**

» пропустить входящие (не загромождаем основную папку);

» применить ярлык "INBOX/Счета на оплату" (создаем удобную категорию);

» никогда не отправлять в спам (важные письма не потеряются);

» отметить как важное (выделяем среди остальных);

» добавить категорию "Оповещения".

Результат: теперь все бытовые счета автоматически организуются в отдельную папку, помечаются как важные и сразу пересылаются на второй адрес. Вы экономите минимум 10 минут ежедневно на сортировке почты, а вероятность пропустить счет снижается до нуля.

Создавайте отдельные фильтры для разных категорий писем. Например, отдельный фильтр для рабочих уведомлений от CRM-системы, другой — для новостных рассылок, третий — для интернет-магазинов. Каждый фильтр делает одну вещь, но делает ее идеально.

3.2. Ваш личный почтовый робот на Google Apps Script

Вы держите в руках книгу о Python, и это совершенно справедливо. Python — наш основной инструмент, мощный, гибкий и идеально подходящий для большинства задач автоматизации. Однако эта книга не просто о языке, она о философии прагматичной автоматизации: использовать лучший инструмент для конкретной задачи.

В этой главе мы дважды отступим от Python в пользу его "коллеги" — JavaScript. И вот почему:

- **Google Apps Script** — это и есть JavaScript. Для автоматизации внутри экосистемы Google (Gmail, Google Docs, Sheets) без внешних серверов Google создал специальную среду Apps Script. Ее "родной" и единственный язык — JavaScript. Это самый простой и прямой путь к созданию нашего первого почтового робота, тот самый "велосипед", о котором мы

говорили. Изучив его, вы поймете, как работают облачные скрипты изнутри.

- **Веб-автоматизация** — стихия JavaScript. Когда речь заходит об управлении браузером и взаимодействии с веб-сайтами (например, веб-версиями мессенджеров), JavaScript находится в своей естественной среде. Для некоторых задач, как в случае с WhatsApp и ВКонтакте, именно в экосистеме Node.js (JavaScript вне браузера) существуют самые стабильные, активно поддерживаемые и мощные библиотеки. Мы могли бы решить эти задачи на Python, но это было бы сложнее и менее надежно. Показывая вам решение на Node.js, я знакомлю вас с промышленным стандартом для веб-автоматизации.

Не пугайтесь нового синтаксиса. Логика программирования универсальна: циклы, переменные, функции работают по тем же принципам и в Python, и в JavaScript. Воспринимайте это не как отклонение от курса, а как расширение вашего арсенала.

Фильтры в **Gmail** — мощный инструмент. Они помогают навести порядок в ящиках и превратить хаос в аккуратные папки. Но у фильтров есть предел: они умеют сортировать, маркировать, пересылать, но не способны вести диалог.

А ведь именно диалог часто нужен. Представьте: вы выложили в открытый доступ полезный файл — электронную книгу, шаблон бюджета или архив с кодом. Ссылкой поделились в блоге или в комментарии на форуме. Через какое-то время в почту начинают приходить десятки однотипных писем с темой **"Запрос доступа к файлу"**.

Можно, конечно, вручную открывать каждое письмо, копировать приветствие и отвечать "Здравствуйте, файл доступен по новой ссылке". Но спустя двадцать писем это превращается в чистую рутину. Автоответчик Gmail не спасает: его возможности слишком примитивны, он не понимает, какой именно файл запрашивают, не умеет подставлять имя человека, и в итоге письмо выглядит как сухая отписка.

Решение? Сделать собственного почтового робота.

Сценарий простой: в почте появляются письма с запросом доступа, робот раз в день их просматривает, отвечает каждому персонально и убирает в корзину, чтобы не мешали. Человек получает письмо, где:

- его приветствуют по имени,
- благодарят за интерес к файлу,
- объясняют, что доступ пока не выдан, но есть инструкция или новая ссылка,
- и все это — вежливо и аккуратно.

Так вы экономите себе часы времени и при этом не выглядите бездушным автоответчиком.

Шаг 1. Создаем скрипт

Google Apps Script встроен прямо в экосистему Google, и начать работать с ним очень просто.

Прежде чем мы вставим код, давайте разберем его логику. Скрипт будет делать три вещи: находить все непрочитанные письма с определенной темой; для каждого такого письма извлекать имя отправителя; формировать и отправлять вежливый ответ, а затем перемещать исходное письмо в корзину, чтобы оно не мешалось.

1. Откройте **Google Диск => Создать => Еще => Google Apps Script**.
2. В редакторе удалите заготовку function myFunction() {} и вставьте следующий код:

```
function autoReply() {
    for (const filename of [
            'Имя файла 1',
            'Имя файла 2',
            'Имя файла 3'
        ]) {
        var threads = GmailApp.search('is:inbox is:unread Запрос доступа
к файлу "${filename}"');
```

```
        for (var i = 0; i < threads.length; i++) {
            var thread = threads[i];
            console.log('Результат поиска: "${filename}": ${i} из
${threads.length}:\n---- ${thread.getMessages()[thread.getMessageCount()
- 1].getReplyTo()};')
            from = thread
                .getMessages()[thread.getMessageCount() - 1].getFrom()
                .split(' (через')[0]
                .replace(/\"/g, '')
            thread.reply('Привет, ${from}!\nБлагодарю Вас за интерес к
моему файлу "${filename}". Бла бла бла \n\nС уважением, Михаил Шардин.');
            GmailApp.moveThreadsToTrash(threads.slice(i, i + 1));

            console.log('Был обработан и удален запрос от ${thread.
getMessages()[thread.getMessageCount() - 1].getReplyTo()} к ${filename}')
        }
    }
}
```

Шаг 2. Разбираем код

Чтобы скрипт не выглядел черным ящиком, разберем ключевые моменты:

- **GmailApp.search()** — ищет письма в почте по условию. В нашем случае фильтр "is:inbox is:unread Запрос доступа к файлу" ловит все непрочитанные письма с нужной темой.
- **getFrom()** — извлекает адрес отправителя. Небольшая обработка .split() и .replace() позволяет убрать хвосты от имени сервера и оставить аккуратное имя.
- **thread.reply()** — отправляет ответ прямо в цепочку. Получатель видит письмо от вас, а не от бездушного робота.
- **moveThreadsToTrash()** — наводит порядок: после ответа цепочка переезжает в корзину и больше не отвлекает.

Шаг 3. Настраиваем и запускаем

1. Сохраните проект под любым именем, например **Gmail auto-responder**.
2. В первый раз нажмите **Выполнить** и выберите функцию **autoReply**. Google попросит подтвердить доступ к почте — соглашайтесь.
3. Чтобы скрипт работал без вашего участия, настройте триггер:
   - » **Триггеры текущего проекта => Добавить**.
   - » Выберите тип **Управляемый временем**.
   - » Настройте запуск — например, один раз в день утром или каждые 15 минут.

Теперь почтовый робот работает автономно.

Искусство автоответа

Техника — это половина дела. Вторая половина — содержание письма. Хороший автоответ должен быть:

- **Персонализированным.** "Здравствуйте, Иван!" звучит лучше, чем "Уважаемый пользователь".
- **Благодарственным.** Простое "Спасибо за интерес к моему файлу" создает позитивное впечатление.
- **Прозрачным.** Укажите, что это автоматический ответ, и дайте понять, что при необходимости вы ответите лично.
- **Полезным.** Вместо сухой отписки приложите аннотацию, инструкцию или новую ссылку.

Подводные камни и зачем это вообще нужно

Возникает вопрос: зачем отвечать письмом, если человек все равно хочет доступ к файлу? Разве не проще просто выдать права?

На практике автоответ решает несколько задач:

- **Управление ожиданиями.** Получатель сразу понимает, что его запрос не потерялся и что ответ будет.
- **Очистка почты.** Запросы не висят сотнями во входящих.

- **Маркетинг.** В ответ можно приложить ссылки на другие материалы, платные версии продукта или ваши контакты.
- **Фильтрация.** Если кто-то просит права на редактирование, можно сразу отправить инструкцию "сделайте копию файла", как делают многие авторы.

Примеры использования:

- Отправка аннотации к файлу (чтобы человек понимал, что скачивает).
- Перенаправление на актуальные ресурсы.
- Автоответ в стиле "редактирование не даем, но копию можно сделать".

Вы научились выходить за рамки простых фильтров Gmail и строить настоящего цифрового помощника внутри почты. Теперь письма не просто сортируются, а ведут осмысленный диалог без нашего участия.

Но электронная почта — это лишь один канал коммуникации. Что если нужно мгновенное уведомление — за минуту до собрания или прямо в телефон человека? Здесь на сцену выходят мессенджеры. В следующем разделе вы узнаете, как превратить Telegram и WhatsApp в ваши автоматизированные каналы связи.

3.3. Диалог с миром (мессенджеры)

Если электронная почта — это протокол делового мира, то мессенджеры — язык повседневной жизни. Здесь решаются бытовые вопросы, организуется семейный досуг, координируется работа небольших команд и другие задачи.

Переход от почтовой автоматизации к мессенджерам требует понимания двух принципиально разных философий работы с API.

Отложенная отправка в мессенджерах: цифровой этикет и здравый смысл

Прежде чем вы погрузитесь в код для мессенджеров, стоит упомянуть один простой, но мощный инструмент — **отложенную отправку**.

Такие функции, как в Telegram, уже приучают нас мыслить не только "что" отправить, но и "когда" (рис. 3.2). Это первый шаг от ручного управления к автоматическому. А теперь давайте сделаем следующий шаг и заставим компьютер отправлять сообщения полностью самостоятельно.

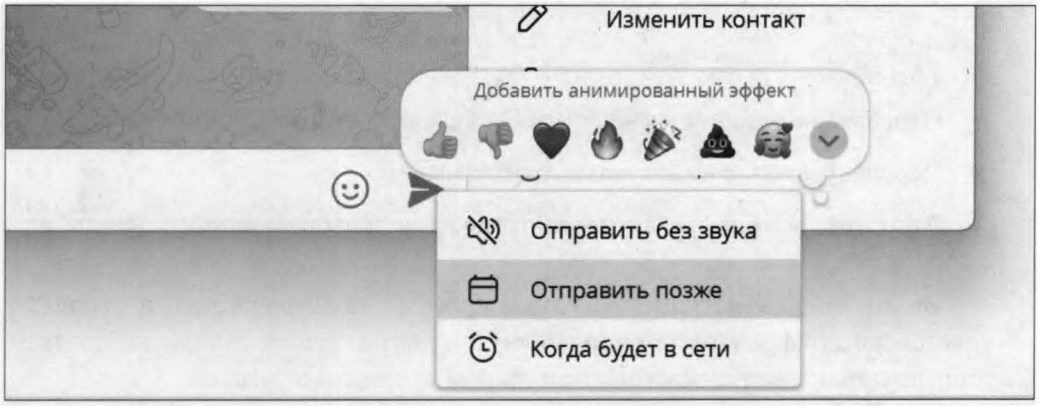


Рис. 3.2. ***Настройка отправки отложенного сообщения через Telegram на десктопе***

> **Отложенная отправка** — это мостик между ручным управлением и полной автоматизацией. Вы все еще контролируете каждое сообщение, но уже начинаете думать в терминах "когда отправить" вместо "что отправить". Это ментальная подготовка к более серьезным инструментам автоматизации, которые вы изучили далее.

Следующий уровень потребует от нас немного программирования, но результат того стоит: полностью автоматические рассылки, персонализированные сообщения и интеграция с внешними данными.

Два пути автоматизации мессенджеров

- **"Белый" путь:** ***использование официальных API***

  Разработчики Telegram и некоторых других платформ предоставляют официальные инструменты для автоматизации. Telegram Bot API позволяет создавать ботов для групп и каналов.

Клиентские библиотеки вроде Telethon дают доступ к функциям обычного пользователя — отправке личных сообщений, работе с контактами, планированию отложенной отправки.

Преимущества очевидны: стабильность, документация, техническая поддержка. Код не ломается после обновления приложения. Риск блокировки аккаунта минимален, если соблюдать официальные лимиты. Разработчики даже заинтересованы в том, чтобы их API использовали — это расширяет экосистему платформы.

Ограничения тоже есть: не все мессенджеры предоставляют API для личной переписки. WhatsApp, например, официально позволяет только создание бизнес-ботов через WhatsApp Business API, что требует верификации и подходит скорее для интернет-магазинов, чем для автоматизации бытового общения.

- **"Серый" путь: *веб-автоматизация***

Когда официального API нет или он не подходит для задачи, в дело вступает автоматизация веб-интерфейса. Специальные библиотеки вроде whatsapp-web.js имитируют действия пользователя: открывают браузер, загружают веб-версию мессенджера, находят нужные элементы на странице и "кликают" по ним программно.

Этот подход работает там, где белого пути просто не существует. С его помощью можно отправлять личные сообщения в WhatsApp, автоматизировать групповые чаты, даже загружать медиафайлы. Главное преимущество — универсальность. Если человек может сделать что-то в веб-интерфейсе, то и скрипт сможет.

Но цена этой универсальности высока. Веб-автоматизация хрупка, как карточный домик. Любое изменение дизайна мессенджера может сломать скрипт. Платформы активно борются с автоматизацией — алгоритмы отслеживают подозрительное поведение, а аккаунты блокируются без предупреждения. Скорость работы ограничена необходимостью имитировать человеческое поведение.

3.3.1. Пример: оповещение автовладельцев через WhatsApp

Рассмотрим классический кейс для "серого" пути — задачу, которая не решается никакими официальными инструментами, но критически важна для повседневной жизни.



Рис. 3.3. ***Пример двора. Фото из Интернета***

Постановка задачи

Ситуация знакома каждому жителю российского города. Завтра во дворе планируется уборка снега. Нужно оповестить 200 автовладельцев о необходимости переставить машины (рис. 3.3). Объявления в подъезде читает половина жильцов, общий чат WhatsApp заглушен у трети участников из-за постоянных обсуждений ремонта лифтов и выгула собак.

Решение лежит в персональной рассылке. У каждого жильца есть номер телефона (из базы управляющей компании), есть данные об автомобиле (марка, госномер), есть WhatsApp. Остается только автоматизировать отправку персонализированных сообщений.



*Рис. 3.4. **Библиотека** whatsapp-web.js*

Техническая реализация

Основа решения — библиотека **whatsapp-web.js**, которая превращает обычный браузер в программируемого клиента WhatsApp (рис. 3.4). Процесс выглядит так:

1. **Авторизация по QR-коду**. Скрипт запускает браузер, открывает web.whatsapp.com и генерирует QR-код. Сканируем его обычным телефоном — и аккаунт привязан к скрипту.
2. **Загрузка данных**. Excel-файл с тремя колонками (имя, номер телефона, марка авто) превращается в JavaScript-массив. Каждая строка — это будущее персональное сообщение.
3. **Формирование сообщений**. Вместо безликого "Уберите машины" каждый получает: "Уважаемый Иван Иванович, просим убрать Вашу Toyota Camry с парковки завтра с 9:00 до 15:00". Персонализация делает сообщение важнее спама, а получателя — сознательнее.
4. **Отправка с задержками**. Между сообщениями — пауза 4–6 секунд. Это имитирует естественный ритм общения и снижает подозрения антиспам-системы.

```
/**
 * Система автоматических уведомлений через WhatsApp для ТСЖ и УК
 *
 * Данный скрипт загружает список жильцов из Excel-файла и отправляет им
сообщения через WhatsApp.
 * Используется библиотека whatsapp-web.js для взаимодействия с WhatsApp Web.
 *
 * Подготовка окружения *
 * 1. Установите Node.js:
 *    - Скачайте LTS версию с официального сайта nodejs.org
 *    - Проверьте установку командой: 'node -v'
 *
 * 2. Установите необходимые библиотеки:
 *    npm install whatsapp-web.js xlsx qrcode-terminal
 *
 * Подготовка данных *
 * Создайте Excel-файл 'TSZH.xlsx' со следующими колонками:
 * - Name (Имя жильца)
 * - Phone number (Номер телефона)
 * - Car (Марка автомобиля, опционально)
 *
 */

// Импорт необходимых библиотек
const { Client, LocalAuth } = require('whatsapp-web.js');
const qrcode = require('qrcode-terminal');
const xlsx = require('xlsx');
const path = require('path');

// Путь к Excel-файлу с данными
const excelFilePath = path.join(__dirname, 'TSZH.xlsx');

// Читаем данные из Excel-файла
console.log("Чтение данных из Excel-файла...");
const workbook = xlsx.readFile(excelFilePath);
const sheetName = workbook.SheetNames[0];
const data = xlsx.utils.sheet_to_json(workbook.Sheets[sheetName]);
```

```
// Вывод данных в консоль сразу после чтения
console.log("Данные из Excel-файла:", data);

// Настройка клиента WhatsApp
console.log("Инициализация клиента WhatsApp...");
const client = new Client({
    authStrategy: new LocalAuth()
});

// Генерация QR-кода
client.on('qr', qr => {
    console.log("Сканируйте этот QR-код в приложении WhatsApp:");
    qrcode.generate(qr, { small: true });
});

// Когда клиент готов
client.on('ready', async () => {
    console.log("Клиент WhatsApp готов к отправке сообщений.");

    // Функция для отправки сообщения с задержкой
    const sendMessageWithDelay = async (person, delay) => {
        return new Promise(resolve => {
            setTimeout(async () => {
                // Проверка наличия имени и номера телефона, чтобы
избежать ошибок
                if (!person.Name || !person['Phone number']) {
                    console.error('Ошибка: Отсутствует имя или номер
телефона для записи:', person);
                    return resolve(); // Переходим к следующей записи
                }

                // Преобразуем номер телефона в строку, если он число
                let phoneNumber = person['Phone number'];
                if (typeof phoneNumber === 'number') {
                    phoneNumber = String(phoneNumber);
                }

                // Удаляем все нецифровые символы из номера и добавляем
суффикс WhatsApp
```

```
                phoneNumber = phoneNumber.replace(/\D/g, '') + '@c.us';


                const message = 'Уважаемый(ая) ${person.Name},

Завтра будет проводиться уборка снега с 9:00 до 18:00. Просим вас
заранее убрать автомобиль ${person.Car || ' '} с парковки, чтобы работа
прошла быстро и качественно.

Спасибо за понимание!';

                try {
                    await client.sendMessage(phoneNumber, message);
                    console.log('Сообщение успешно отправлено на номер
${person['Phone number']}.');
                } catch (err) {
                    console.error('Ошибка при отправке сообщения на
номер ${person['Phone number']}:', err);
                }
                resolve();
            }, delay);
        });
    };


    // Персонализированная рассылка сообщений с задержкой
    for (let i = 0; i < data.length; i++) {
        // Задержка 4 секунды между сообщениями (15 сообщений в минуту)
        await sendMessageWithDelay(data[i], i * 4000);
    }

});

// Обработка ошибок
client.on('auth_failure', () => {
    console.error("Ошибка авторизации. Проверьте свои учетные данные.");
});
```

```
client.on('disconnected', () => {
    console.log("Клиент WhatsApp был отключен.");
});

// Запуск клиента
client.initialize();
```

3.3.2. Думаем как IT-специалист: безопасность и риски

Риск блокировки аккаунта

WhatsApp рассматривает массовые рассылки как потенциальный спам. Алгоритмы анализируют частоту отправки, похожесть текстов, количество новых чатов. Превысили порог — аккаунт заблокирован на 24 часа или навсегда (рис. 3.5).



Рис. 3.5. ***Риск блокировки аккаунта***

Снижение риска:

- Персонализированные тексты вместо копипасты.
- Задержки 4–6 секунд между сообщениями (максимум 15 сообщений в минуту).

- Ограничение 50–100 сообщений за сессию.
- Использование "технического" номера телефона, а не основного.

Работа с персональными данными

База данных с именами и телефонами жильцов — это персональные данные в понимании 152-ФЗ. Председатель ТСЖ имеет право их обработки для целей управления домом, но автоматическая рассылка требует дополнительных мер защиты:

- Файл Excel должен храниться локально, без синхронизации в облако.
- Желательно получить письменные согласия жильцов на SMS/мессенджер-уведомления.

Техническая надежность

Веб-автоматизация может сломаться в любой момент. WhatsApp регулярно обновляет интерфейс, меняя идентификаторы элементов и логику работы. Библиотека **whatsapp-web.js** обычно адаптируется за несколько дней, но в критический момент скрипт может просто не запуститься.

Подстраховка: всегда держите под рукой план Б — список телефонов для ручной рассылки через обычные SMS или общедомовой чат.

3.3.3. Пример 2: автоматизация личной переписки в Telegram

Этот пример демонстрирует возможности "белого" пути и одновременно поднимает вопросы этики автоматизации.

Постановка задачи

Современный ритм жизни не всегда оставляет время для поддержания личных связей. Работа затягивает, проекты требуют концентрации, а близкие люди нуждаются во внимании. Компромиссное решение — автоматизация инициации общения, когда скрипт отправляет первое сообщение, а дальше продолжается живой диалог.

Техническая реализация

Основа — библиотека **Telethon** для Python, которая использует официальный клиентский API Telegram. В отличие от веб-автоматизации, здесь нет имитации кликов и поиска элементов на странице. Скрипт работает напрямую с серверами Telegram, как обычное приложение (рис. 3.6).

Настройка:

- Регистрация приложения на *my.telegram.org* (получение *api_id* и *api_hash*).
- Авторизация через номер телефона и код подтверждения.
- Сохранение сессии для повторных запусков.

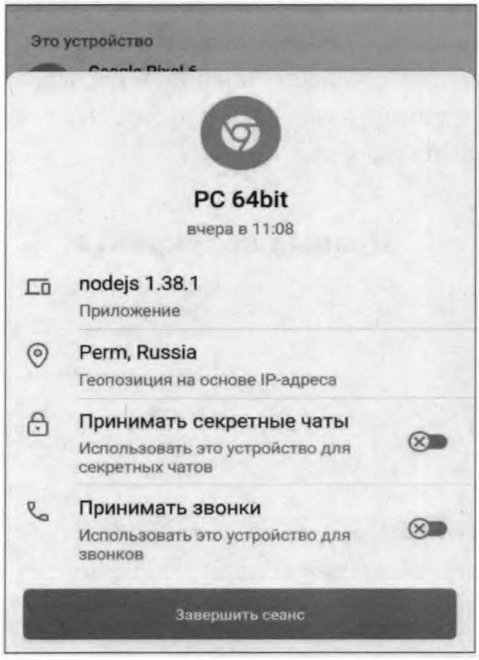


Рис. 3.6. ***Скрипт на Telethon отображается в списке устройств Telegram***

Логика работы:

```
# Пример планирования сообщений
greetings = [
    "Доброе утро! Как настроение?",
    "Привет! Надеюсь, день складывается удачно",
```

```
    "Как дела? Что нового происходит?"
]

# Отложенная отправка на завтра в случайное время
schedule_time = datetime.now() + timedelta(days=1)
random_time = schedule_time.replace(
    hour=random.randint(8, 10),
    minute=random.randint(0, 59)
)

message = random.choice(greetings)
await client.send_message(user, message, schedule=random_time)
```

Ключевая особенность — использование встроенной функции отложенной отправки Telegram. Сообщения планируются заранее, но отправляются официальными серверами в назначенное время. Это выглядит естественно и не нарушает правила платформы.

Полный код скрипта

```
# pip install telethon python-dotenv pytz

import sys
sys.stdout.reconfigure(encoding='utf-8')

import asyncio
from telethon import TelegramClient, utils
from datetime import datetime, timedelta, time
import random
from dotenv import load_dotenv
import os
import pytz

# Загрузка переменных окружения из .env
load_dotenv()
api_id = os.getenv("api_id")  # Ваш API ID
api_hash = os.getenv("api_hash")  # Ваш API Hash
session_name = 'nodejs_session'  # Название сессии
```

```
# Сообщения
greetings = [
    "Доброе утро! Как ты себя чувствуешь сегодня?",
    "Привет, как настроение? Готовность к новому дню?",
    "Доброе утро! Что нового на сегодня?",
    "Утро доброе! Как у тебя все происходит с утра?",
    "Доброе утро, как ты себя сегодня чувствуешь?",
    "Привет! Как ты, как сны?",
    "Как твои дела, готовность к этому дню?",
    "Доброе утро! Как настроение, что планируешь на день?",
    "Здравствуй! Как ты себя чувствуешь после сна?"
]
supportive_messages = [
    "Как ты там? Все в порядке?",
    "Как проходит твой день?",
    "Надеюсь, день не слишком утомительный для тебя!",
    "Как настроение? Все ли хорошо?",
    "Не забывай делать перерывы, если устала!",
    "Как ты? Чем занимаешься сейчас?",
    "Если что-то нужно, не стесняйся мне писать.",
    "Надеюсь, день приносит тебе только положительные моменты!",
    "Как ты себя чувствуешь? Хочешь поговорить?"
]

# Таймзона Екатеринбурга
ekb_timezone = pytz.timezone('Asia/Yekaterinburg')

async def schedule_messages(client, user, messages, start_hour, end_
hour, days, description):
    """Планирует сообщения для заданного пользователя на несколько дней."""
    for day in range(days):
        # Вычисляем дату и время в Екатеринбурге
        now_ekb = datetime.now(ekb_timezone)
        schedule_date_ekb = now_ekb.date() + timedelta(days=day + 1)
        schedule_time_ekb = ekb_timezone.localize(datetime.combine
(schedule_date_ekb, time(hour=random.randint(start_hour, end_hour),
minute=random.randint(0, 30))))
```

```
        # Конвертируем время в UTC для Telethon
        schedule_time_utc = schedule_time_ekb.astimezone(pytz.UTC)

        message_text = random.choice(messages)

        try:
            await client.send_message(user, message_text, schedule=
schedule_time_utc)
            print(f"✅ Отложенное сообщение отправлено для
{description} на {schedule_time_ekb.strftime('%Y-%m-%d %H:%M:%S')}")
        except Exception as e:
            print(f"❌ Ошибка при отправке сообщения для {description}: {e}")


async def main():
    client = TelegramClient(session_name, api_id, api_hash)
    await client.start()

    print("🚀 Запуск программы...")
    try:
        tasks = []

        user_Name1 = await client.get_entity("ВАШ_USERNAME_ЗДЕСЬ") #
Имя пользователя
        tasks.append(schedule_messages(client, user_Name1, greetings,
7, 7, 3, "пользователя userName"))  # Утренние сообщения
        tasks.append(schedule_messages(client, user_Name1, supportive_
messages, 12, 12, 3, "пользователя userName"))  # Дневные сообщения

        user_Name2 = await client.get_input_entity("+7912XXXXX")  #
Непосредственно номер телефона
        tasks.append(schedule_messages(client, user_Name2, greetings,
7, 7, 3, "пользователя +7912XXXXX"))  # Утренние сообщения

        await asyncio.gather(*tasks)
        print("🎉 Все сообщения успешно настроены.")
    except Exception as e:
```

```
        print(f"✗ Ошибка: {e}")
    finally:
        await client.disconnect()
        print("🔌 Соединение с Telegram завершено.")


if __name__ == '__main__':
    asyncio.run(main())
```

Этика автоматизации

Автоматизация личного общения — территория этических споров. Я писал статью на эту тему и получил следующие комментарии к ней. Немного философского диспута:

Позиция "за":

"Это не обман, а современная напоминалка. Человек сам пишет тексты, он просто делегирует рутину отправки. Это способ проявить заботу, когда ты очень занят".

Сторонники подхода видят в автоматизации продолжение естественной эволюции коммуникационных инструментов. Мы уже используем отложенные сообщения в Telegram, напоминания в календаре, автоответчики в мессенджерах. Скрипт просто объединяет эти возможности в единую систему.

Аналогия с подарками показательна: никто не осуждает мужчину, который ставит напоминание о дне рождения жены в календаре. Важен сам факт внимания, а не способ, которым оно проявляется. Более того, создание такого скрипта требует времени и размышлений о том, что важно для близкого человека — это тоже форма заботы.

Позиция "против":

"Это имитация внимания и неискренность. Отношения строятся на живом участии, а не на скриптах. Что будет, если адресат узнает?"

В комментариях оппоненты указывают на подмену понятий. Истинное внимание возникает из эмпатии — способности чувствовать состояние другого человека и откликаться на него.

Автоматические сообщения по расписанию этой связи лишены. Они создают иллюзию заботы, за которой может скрываться равнодушие.

Критики предлагают простой тест честности: расскажите адресату об автоматизации и посмотрите на реакцию. Если человек отнесется к этому негативно, значит, автоматизация нарушает его ожидания от отношений. А скрывать такие вещи — прямая дорога к разрушению доверия.

Возможно, истина лежит в понимании контекста и целей. Автоматизация может быть этичной, если:

- она используется как напоминание для себя, а не замена живого общения;
- адресат знает об автоматизации и не возражает против нее;
- скрипт инициирует диалог, но не ведет его;
- целью является забота, а не создание видимости внимания.

Один из комментаторов к той статье привел показательный пример: его девушка сама предложила такую автоматизацию, потому что хотела получать больше внимания, а парень часто забывал написать из-за загруженности работой. В этом случае автоматизация решила реальную проблему с согласия всех сторон.

Важно понимать, где автоматизация уместна, а где становится токсичной.

Уместно:

- Напоминания о важных датах (дни рождения, годовщины).
- Утренние приветствия для поддержания регулярного контакта.
- Пожелания удачи перед важными событиями.

Токсично:

- Имитация эмоциональной поддержки в сложных ситуациях.
- Автоматические ответы на серьезные вопросы.
- Скрытая автоматизация без согласия адресата.
- Замена живого общения на важные темы.

Мессенджеры открывают новые возможности для автоматизации, но требуют более тонкого подхода, чем электронная почта. Технический инструмент остается нейтральным — его этичность определяется целями и способом применения.

Следующий раздел покажет, как перейти от индивидуальных сообщений к управлению целыми сообществами через автоматизацию социальных сетей.

3.4. Управление репутацией (соцсети)

Если автоматизация личной переписки в мессенджерах — это прогулка по минному полю этики, то автоматизация публичной активности в социальных сетях — это та же прогулка, но на моноколесе и с завязанными глазами. Ставки выше, риски заметнее, а цена ошибки — ваша публичная репутация. Вы переходите от диалогов тет-а-тет к общению с целой аудиторией. Здесь каждый "лайк", каждый комментарий виден десяткам, а то и сотням людей.

Логика этого шага проста: вы уже научились отправлять информацию, теперь научимся на нее публично реагировать. API для этого предоставляют многие платформы, и ВКонтакте, с его богатой документацией и гибкими возможностями, — идеальный полигон для экспериментов.

Поддержание социальной активности для "лентяев" в ВКонтакте

Сценарий: у вас в друзьях ВКонтакте больше трехсот человек: старые одноклассники, коллеги с пяти разных работ, дальние родственники и люди, которых вы добавили после случайной встречи на конференции. Следить за их жизнью нет ни времени, ни желания — лента превратилась в бесконечный поток чужих отпусков, политических дебатов и фотографий детей. Но полное игнорирование ослабляет социальные связи, а в профессиональной среде может выглядеть как признак незаинтересованности.

Проблема: социальные сети спроектированы так, чтобы поглощать ваше время. Лайк — это минимальная единица социального взаимодействия, цифровое "я тебя вижу", которое поддерживает контакт, не требуя реального вовлечения. Но даже на это простое действие уходят часы, если делать это вручную. Как поддерживать видимость социальной жизни, не тратя на нее… ну, саму жизнь?

Решение: написать скрипт, который будет вашим цифровым аватаром. Раз в день он будет заходить в ВК, пробегаться по списку друзей, находить их последний пост и ставить лайк. Выглядит как идеальное решение: время сэкономлено, социальные связи подкреплены, все довольны.



Рис. 3.7. ***JavaScript и библиотека*** *vk-io*

Техническая реализация:

Основа нашего решения – JavaScript (Node.js) и библиотека **vk-io**, которая значительно упрощает общение с API ВКонтакте (рис. 3.7). Вместо того чтобы вручную формировать сложные HTTP-запросы, вы будем вызывать простые и понятные функции.

Процесс работы скрипта можно разбить на четыре шага:

1. **Авторизация.** Чтобы скрипт мог действовать от вашего имени, ему нужен специальный ключ доступа — токен. Представьте его как временный паспорт, который вы выдаете своему приложению. Получить его можно в настройках для разработчиков ВКонтакте. Этот токен — сверхсекретная информация, его нельзя публиковать или передавать кому-либо.
2. **Получение списка друзей.** Скрипт обращается к API ВКонтакте с запросом: "Привет, это я. Дай мне, пожалуйста, полный список ID всех моих друзей". За это отвечает метод vk.api.friends.get().
3. **Поиск и оценка поста.** Далее скрипт запускает цикл: для каждого ID из полученного списка он делает новый запрос: "А теперь покажи мне самый свежий пост на стене этого пользователя". Это работа метода vk.api.wall.get({ owner_id: friendId, count: 1 }).
4. **Простановка лайка (и пауза!)**. Если пост найден, скрипт проверяет, не ставили ли вы уже на него лайк. Если нет — он отправляет команду "поставить лайк" через метод vk.api.likes.add(). А после этого наступает самый важный момент: скрипт засыпает на 1–2 секунды (await delay(1200)). Эта пауза имитирует поведение живого человека и защищает ваш аккаунт от блокировки за подозрительную активность.

Вот как выглядит код:

```
const { VK } = require('vk-io');
const fs = require('fs');

const credentialsFile = './secret/credentials.json';
const credentialsData = JSON.parse(fs.readFileSync(credentialsFile,
'utf-8'));

const vk = new VK({
    token: credentialsData.accessToken  // Ваш токен доступа из файла
```

```
});

async function likeLastPost() {
    const { default: delay } = await import('delay');

    try {
        // Получаем список всех друзей
        const friends = await vk.api.friends.get({
            fields: 'nickname', // Необязательно, но может быть
полезно для вывода
            order: 'hints' //  hints — рекомендуемые друзья, name —
по алфавиту. Можно убрать, тогда будет в порядке по умолчанию.
        });
        const friendIds = friends.items.map(friend => friend.id)

        console.log('Найдено ${friendIds.length} друзей.');


        const [me] = await vk.api.users.get({});
        console.log('Мой ID: https://vk.com/id${me.id}');

        let actionCounter = 1; // Инициализируем счетчик действий
        let likeCounter = 1;

        for (const friendId of friendIds) {
            try {

                // Получаем последний пост друга
                const wall = await vk.api.wall.get({
                    owner_id: friendId,
                    count: 1
                });

                if (wall.items.length > 0) {
                    const post = wall.items[0];

                    // Проверяем, лайкали ли мы уже этот пост
                    const likes = await vk.api.likes.getList({
```

```
                        type: 'post',
                        owner_id: post.owner_id,
                        item_id: post.id,
                        filter: 'likes'
                    });

                    if (!likes.items.includes(me.id)) { // Проверяем
наличие текущего userId в списке лайкнувших
                        // Ставим лайк
                        await vk.api.likes.add({
                            type: 'post',
                            owner_id: post.owner_id,
                            item_id: post.id
                        });
                        console.log('${actionCounter} из ${friendIds.
length}: поставил лайк на пост https://vk.com/id${friendId}?w=wall${f
riendId}_${post.id} 👍'); // друга https://vk.com/id${friendId}
                        actionCounter++;
                        likeCounter++;
                    } else {
                        console.log('${actionCounter} из ${friendIds.
length}: пост https://vk.com/id${friendId}?w=wall${friendId}_${post.
id} уже лайкнут');
                        actionCounter++;
                    }
                } else {
                    console.log('${actionCounter} из ${friendIds.
length}: у друга https://vk.com/id${friendId} нет постов.');
                    actionCounter++;
                }

                // Задержка для предотвращения бана
                await delay(1200); // 1200мс = 2 секунды,  60000мс /
50 запросов = 1200мс на запрос

            } catch (error) {
                console.error('Ошибка обработки друга ${friendId}:',
error);
```

```
            }
        }
        console.log('\nПроставлено лайков: ${likeCounter}, а всего в
списке было ${friendIds.length} друзей.');
    } catch (error) {
        console.error('Глобальная ошибка:', error);
    }
}


likeLastPost();
```

Скрипт работает. Лайки расставляются. Вы чувствуете себя гением автоматизации. Но, прежде чем запускать его на постоянной основе, давайте подумаем.

Обратите внимание, что токен мы загружаем из отдельного файла *credentials.json*. Никогда не вставляйте токены и пароли прямо в код! Если вы опубликуете такой код, например, на GitHub, ваши учетные данные могут украсть. Хранение их в отдельном файле, который вы не публикуете, — базовое правило цифровой гигиены.

Думаем о последствиях: репутационные риски слепой автоматизации

Технически задача решена безупречно. Но с точки зрения здравого смысла мы создали цифрового идиота, который с энтузиазмом жмет "Мне нравится" под всем, что видит. А теперь представьте реальные ситуации.

Что если ваш скрипт лайкнет пост о трагедии? Ваш знакомый пишет о смерти близкого человека, делится своей болью, а через час под постом появляется ваш радостный синий палец. Выглядит чудовищно.

Что если он одобрит спорное политическое высказывание? Друг детства репостнул контент, который противоречит вашим убеждениям или, что хуже, находится на грани законности. Ваш лайк — это публичное согласие. Готовы ли вы объяснять коллегам или начальству, почему вы "поддержали" именно эту точку зрения?

Что если это просто глупая шутка или безвкусный мем? Ваш лайк навсегда останется в ленте у других людей как свидетельство вашего "отличного" чувства юмора.

> **Вывод прост: слепая автоматизация опасна.** Ваш лайк — это не просто клик, это публичное высказывание, акт солидарности. Доверяя его бездушному алгоритму, вы рискуете своей репутацией. Этот пример — отличная демонстрация технической мощи API, но одновременно и суровое предостережение.

Значит ли это, что от идеи нужно отказаться? Не совсем. Инструмент не плох сам по себе, важен способ его применения. Идеальное использование такого скрипта — не для всех 300+ друзей, а для узких, контролируемых задач:

- **Создание "белого списка"** — запускать скрипт только для 10–15 самых близких друзей, в чьем контенте вы уверены.
- **Поддержка тематических сообществ** — автоматически лайкать все посты в группе вашего хобби (например, клуба любителей астрофотографии), где контент предсказуем и всегда позитивен.
- **Рабочие задачи** — если ваша команда использует группу в ВК для публикации отчетов, скрипт может автоматически "одобрять" их, показывая, что вы видели обновление.

Вы прошли путь от сортировки почты до имитации социальной жизни. Вы увидели, как простые скрипты могут не только экономить десятки часов, но и создавать серьезные этические и репутационные риски. Ключевой вывод этой главы прост.

Технические принципы:

- Выбирайте "белый" путь, когда он доступен — официальные API стабильнее и безопаснее.
- При использовании "серых" методов всегда готовьте план Б на случай поломки.
- Соблюдайте лимиты, имитируйте поведение человека и используйте технические аккаунты для рискованных операций.

Этические принципы:

- Честность превыше эффективности. Автоматизация должна помогать, а не обманывать.
- Автоматизация — это дополнение к живому общению, а не его замена.
- Всегда учитывайте контекст и возможную реакцию получателя.

Помните: молоток можно использовать для строительства дома или для разбивания окон. Инструменты автоматизации работают по тому же принципу. Используйте их для созидания, а не для имитации.

Заключение: ваш цифровой секретарь готов к работе

Эта глава была настоящим погружением в искусство коммуникации. Мы начали с малого — научились наводить идеальный порядок в почтовом ящике с помощью фильтров, а затем создали своего первого почтового робота на Google Apps Script, который вежливо и по делу отвечает на однотипные запросы. Мы превратили почтовый ящик из склада писем в умного ассистента.

Дальше — больше. Мы шагнули в мир мгновенных сообщений, где освоили два фундаментально разных подхода. Мы прошли и "белый" путь официальных API Telegram, создав этичного и полезного помощника для поддержания личных связей, и рискованный, но порой незаменимый "серый" путь веб-автоматизации, чтобы решить бытовую задачу с оповещением жильцов в WhatsApp. Наконец, мы вышли на публичную арену социальных сетей и на примере сервиса ВКонтакте научились не только делегировать рутинные действия, но и глубоко понимать репутационные риски, которые несет за собой слепая автоматизация. Наш арсенал пополнился мощными инструментами, но, что важнее, мы научились думать о последствиях.

Главная мысль, которую вы должны вынести из этой главы, заключается в смене вашей роли. Вы больше не просто исполнитель, который вручную пересылает файлы и пишет однотипные сообщения. Вы — архитектор коммуникационных систем. Вы прошли путь от курьера, который лично разносит каждую посылку, до диспетчера, который один раз настраивает маршруты, а

затем лишь контролирует слаженную работу огромной логистической сети. Этот навык освободил для вас еще один огромный пласт времени и умственной энергии — тот, что раньше уходил на рутину доставки информации. Вы теперь не просто нажимаете кнопку **Отправить**. Вы проектируете системы, которые общаются за вас: вовремя, к месту и с нужным содержанием.



Глава 4.

Умный дом, который работает на вас, а не на корпорации

Представьте, что вы заходите домой, а свет уже зажегся, шторы открылись и любимая музыка тихо играет в гостиной. Вы говорите "включи чайник" — и система выполняет вашу команду мгновенно, без пауз на "думание" и без риска, что ваши слова где-то сохранятся и будут проанализированы. Весь этот комфорт работает, даже если Интернет отключен на сутки. Это не сцена из фантастического фильма о далеком будущем. Это реальность, которую вы можете построить уже сегодня.

А теперь другая картина. Вы пытаетесь выключить свет в спальне через приложение, а в ответ — молчание и ошибка "Нет соединения с сервером". Ваша умная розетка, купленная пару лет назад, внезапно превратилась в обычную, потому что производитель решил закрыть сервис. Ваши привычки, распорядок дня и даже моменты, когда вас нет дома, становятся товаром для крупных корпораций.

Какой из этих сценариев вам ближе? Если второй, то эта глава — для вас. Мы не будем здесь говорить о том, как подключить очередную "умную" лампочку к облаку Алисы или Google. Напротив, мы объявим бунт против этой модели. Наша цель — цифровой суверенитет: умный дом, который принадлежит именно вам, работает локально, защищает вашу приватность и не теряет функциональности из-за решений менеджеров в другой стране.

С помощью открытой платформы Home Assistant и небольшой инженерной смекалки мы построим систему, которая будет подчиняться только вам. Мы начнем с основ, развернув "мозг" управления за 30 минут, и дойдем до

продвинутых сценариев, где Python и компьютерное зрение помогают анализировать происходящее вокруг. Мы вернем к жизни "мертвые" устройства, заблокированные производителями, и создадим собственного голосового ассистента, который не шпионит за вами.

Готовы перестать быть пользователем и стать хозяином своего умного дома? Тогда начинаем.

4.1. Бунт против облака: почему ваш умный дом должен принадлежать вам

Представьте: вы поднимаетесь ночью, чтобы выключить свет в детской кликом на привычной кнопке — и ничего не происходит. Приложение показывает "Ошибка сервера". Или другой сценарий: производитель меняет политику или закрывает поддержку — и через год ваши "умные" розетки перестают принимать команды. Такие случаи уже не гипотеза — это реальность, вот реальные случаи, которые иллюстрируют проблему облачной зависимости:

- **Philips Hue** — интеграция (навык) с Домом/Алисой у части пользователей переставала работать (пример ограничения, связанного с санкциями).
- **Home Connect (Bosch/Siemens)** — сервис перестал работать в России: приложение выводит "В вашем регионе приложение не поддерживается".
- **Устройства Сбер на базе Tuya** — умные лампы и розетки под маркой Сбера были сняты с продажи, и подключать новые устройства стало невозможно.
- **Legrand** — фирменное приложение недоступно в России с декабря 2023 года: новые пользователи не могут добавить или изменить устройства (у старых приложение пока продолжает работать).
- **Redmond (платформа Sky)** — в ноябре 2023 у ряда приборов Redmond возникли проблемы с подключением к платформе Sky; по тексту проблема была частично решена в 2024 году, но это хороший пример того, как лицензии и договоры влияют на работоспособность.

- **Insteon** — разработчик отключил свои серверы, и оборудование потеряло смарт-функции.
- **"Умный дом Яндекса" / Алиса** — короткий, но показательный сбой в декабре 2023 года, когда система не работала около дня из-за аварии на серверах.

Эти примеры показывают одни и те же риски в разных обличьях: санкции, изменение политики, менеджмент лицензий, технические сбои — и результат один: ваш "умный" гаджет может внезапно стать "очень глупым".

Риски облачной зависимости можно свести в несколько ключевых тезисов:

- **Отсутствие Интернета = отсутствие дома.** Если логика проходит через удаленные серверы, локальный обрыв связи делает устройство бесполезным.
- **Компания ушла — функции исчезли.** Производитель может закрыть поддержку старых устройств или уйти с рынка, и вы останетесь с железом, которое не умеет ничего, кроме базовой "физической" функции.
- **Ваши данные — товар.** Расписания, шаблоны поведения, время прихода/ухода — все это ценно для корпораций. Чем больше облачных сервисов, тем больше данных у третьих лиц.
- **Задержка и надежность.** Команда, уходящая в Облако и возвращающаяся обратно, неизбежно медленнее и имеет лишние точки отказа.
- **Безопасность и доступ.** Компрометация облачного аккаунта или провайдера может дать злоумышленнику контроль над девайсами — или хотя бы доступ к подробной картине вашей жизни.

Эти угрозы — не гипотеза для параноиков, а практический вывод из описанных кейсов.

4.1.1. Решение — локальный умный дом: почему это работает

Противоядие простое: перенести управление и данные домой. Локальный умный дом — это когда команды, автоматизации и журналы событий хранятся и исполняются внутри вашей локальной сети, на выделенном сервере в квартире, на NAS или мини-ПК.

Что вы получаете:

- **Независимость от внешних серверов.** Большинство сценариев будут работать вне Интернета. Критические функции (перекрытие воды, сигнализация, резервный свет) — в локальной логике.
- **Приватность.** Данные остаются у вас; вы решаете, что и когда отправлять в Облако (если вообще отправлять).
- **Надежность.** Меньше сторонних точек отказа. Резервное питание (UPS) и физические ручки/рубильники дают дополнительные гарантии.
- **Гибкость и долговечность.** При уходе производителя с рынка устройства можно перепрошить, заменить интеграцию или продолжить работу через локальные протоколы.

Это не означает полного "отречения" от Облака: полезны удаленный доступ, пуш-уведомления и бэкапы, но контроль и "жизненно важные" сценарии должны быть локальными.

4.1.2. Знакомство с "мозгом" — Home Assistant как практический выбор

Home Assistant — бесплатная, открытая платформа, которую легко поставить на Raspberry Pi, NUC или NAS. Он выступает "дирижером": объединяет устройства разных брендов (Xiaomi, Tuya, Aqara, Legrand и др.), протоколы (Zigbee, Z-Wave, MQTT) и локальные скрипты в единое управление.

Почему Home Assistant подходит именно для локального подхода:

- **Множество интеграций** — сотни готовых интеграций, включая работу с локальными протоколами и камерными серверами.
- **Автоматизации работают локально** — многие сценарии исполняются без отправки команд в Облако.
- **История и логирование** — храните траекторию температуры, энергопотребления и сработавших датчиков у себя.

- **Сообщество и плагины** — большое количество готовых рецептов и решений, в том числе обходов для устройств, "приклеенных" к чужому Облаку (или примеров перепрошивок).

4.1.3. Практический чек-лист для перехода

1. Выделите сервер (Raspberry Pi 3/4/5, другой мини-ПК, NAS).
2. Подумайте о Zigbee-стике или, что дороже, о Z-Wave-контроллере.
3. Организуйте резервное питание для сервера и важной периферии.
4. Сохраните критичные вручную переключаемые цепи (рубильники, механические выключатели).
5. Проверьте список устройств: если кто-то жестко привязан к Облаку (как в приведенных примерах — Legrand, некоторые устройства Sber/Tuya, Insteon), изучите варианты перепрошивки или замены.
6. Делайте регулярные бэкапы конфигурации Home Assistant перед апдейтами.

4.2. Практикум: разворачиваем "мозг" за 30 минут

Теория — это прекрасно, но наша цель — практика. Мы выяснили, почему локальный умный дом лучше облачного. Теперь давайте создадим его фундамент — установим центральный сервер, который станет дирижером для всех ваших устройств.

Забудьте о сложных инструкциях и командной строке. Следуя этому руководству, вы запустите собственный сервер Home Assistant быстрее, чем заварится утренний кофе. Наша цель — получить работающий "мозг" умного дома с минимальными усилиями, чтобы вы сразу почувствовали результат.

4.2.1. Железо для "мозга": выбираем платформу

Сердцем нашего умного дома станет небольшой компьютер, который будет работать 24/7.

Самый популярный, доступный и энергоэффективный вариант — это микрокомпьютер **Raspberry Pi**. Для наших задач идеально подойдут модели Raspberry Pi 4 или Raspberry Pi 5, желательно с 4 ГБ оперативной памяти или больше. Этого с запасом хватит для десятков устройств и сотен автоматизаций.

Впрочем, если у вас без дела лежит старый ноутбук или неттоп (мини-ПК вроде Intel NUC), они тоже отлично справятся с ролью сервера. Главное преимущество Raspberry Pi — огромное сообщество и низкое энергопотребление, поэтому в качестве основного примера мы будем использовать именно его.

4.2.2. Пошаговая инструкция: от коробки до рабочего сервера

Шаг 1. Что купить: минимальный набор

Для запуска вам понадобится всего четыре компонента:

1. **Raspberry Pi 4 или 5.** Как мы уже говорили, это оптимальный выбор.
2. **Накопитель для операционной системы.** Здесь есть два варианта:
   - **Карта microSD (от 32 ГБ, класс A2)** — простой и дешевый способ для старта. Однако карты памяти менее надежны при постоянной записи данных, что характерно для Home Assistant.
   - **Накопитель NVMe M.2 с адаптером** — это более продвинутый и настоятельно рекомендуемый вариант для долгосрочного использования. Он в разы быстрее и значительно надежнее microSD-карт.
3. **Качественный блок питания.** Не экономьте на питании. Для Raspberry Pi 4 нужен блок на 5V/3A, для Raspberry Pi 5 — на 5V/5A. Слабый блок питания — частая причина сбоев.
4. **Корпус.** Он защитит плату от пыли и повреждений, а модели с пассивным охлаждением (корпус-радиатор) обеспечат стабильную работу без перегрева.

Шаг 2. Как установить Home Assistant OS: проще не бывает

Мы будем использовать официальный метод установки — Home Assistant OS. Это готовая операционная система, в которой уже есть все необходимое. Никакой командной строки, только мышь и пара кликов.

1. **Скачайте Raspberry Pi Imager.** Зайдите на официальный сайт Raspberry Pi и загрузите утилиту Raspberry Pi Imager для вашей операционной системы (Windows, macOS или Linux).

2. **Запустите утилиту и выберите все необходимое.**

   » Нажмите **Выбрать устройство** и выберите вашу модель Raspberry Pi (например, Raspberry Pi 5).

   » Нажмите **Выбрать ОС**. В открывшемся меню выберите **Other specific-purpose OS => Home assistants and home automation => Home Assistant** (рис. 4.1). Утилита сама подберет нужную версию для вашей платы.

   » Нажмите **Выбрать запом. устройство**. Вставьте вашу microSD-карту или NVMe-накопитель в компьютер и выберите его в списке.

3. **Запишите образ.** Нажмите кнопку **Далее**. Программа сама скачает последнюю версию Home Assistant OS и запишет ее на носитель. Это может занять 5–10 минут.

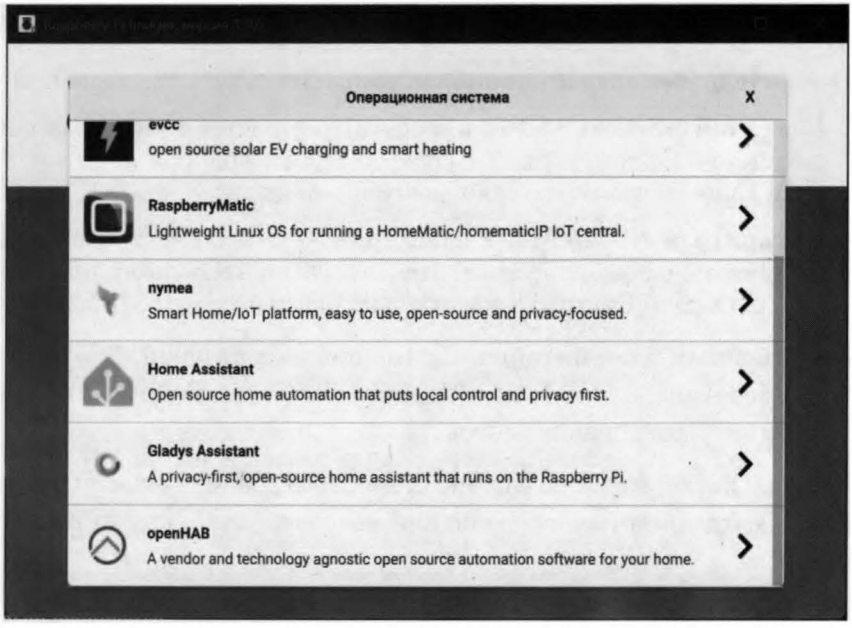


Рис. 4.1. ***Raspberry Pi Imager и выбор операционной системы***

Шаг 3. Первый запуск: магия начинается

Самый волнительный момент. Все готово к первому старту.

1. **Подключите все.** Вставьте карту microSD (или установите NVMe-плату) в ваш Raspberry Pi. Подключите к нему сетевой кабель Ethernet от вашего роутера — на первом этапе это обязательно для надежности.
2. **Включите питание.** Подключите блок питания к Raspberry Pi. Загорятся индикаторы — процесс пошел.
3. **Будьте терпеливы.** Во время первого запуска Home Assistant производит начальную настройку. Это может занять от 5 до 20 минут. Просто дайте ему поработать.
4. **Найдите ваш новый умный дом.** Откройте браузер на компьютере, который находится в той же сети, и перейдите по адресу: http://homeassistant.local:8123 .
5. **Создайте учетную запись.** Вы увидите приветственный экран, который сообщит, что Home Assistant готовится к запуску. Как только он будет готов, вам предложат создать учетную запись администратора. Придумайте имя пользователя и надежный пароль — это будет ваш ключ к управлению всем домом (рис. 4.2).

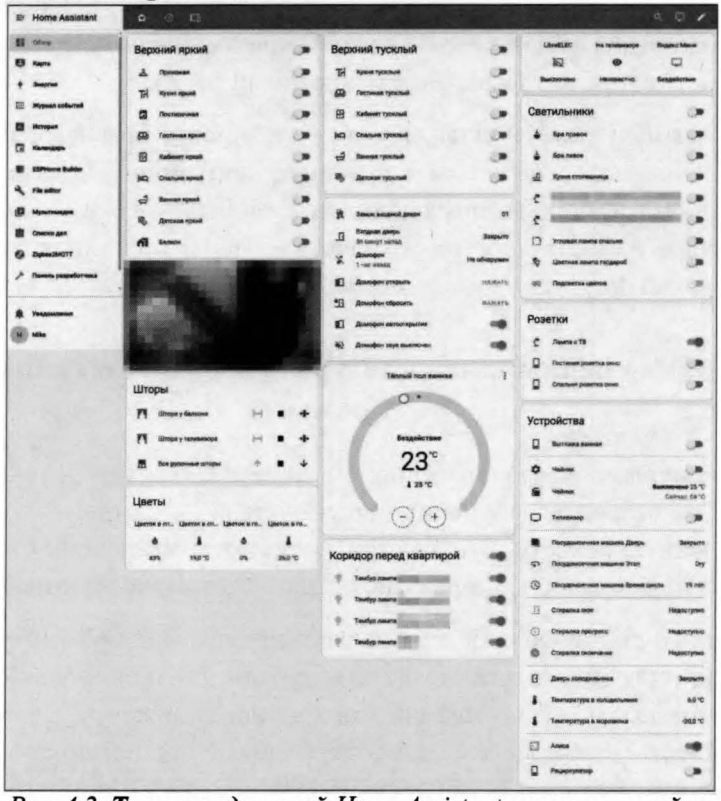


Рис. 4.2. ***Так выглядит мой*** *Home Assistant* ***после настройки***

4.2.3. Результат: ваш цифровой суверенитет обрел форму

Поздравляю! Если вы видите главный экран Home Assistant, значит, все получилось. Буквально за полчаса вы развернули мощный, локальный и полностью ваш сервер умного дома. Это фундамент, на котором мы будем строить систему автоматизации, независимую от прихотей корпораций и сбоев в Интернете.

Теперь, когда "мозг" готов, пора научить его управлять первыми устройствами и создавать сценарии, которые сделают вашу жизнь проще. Об этом — в следующей главе.

4.3. Первые "победы": автоматизация, которую можно потрогать

Теперь, когда у нас есть работающий "мозг" системы, пора научить его чему-то полезному. Давайте оживим наш Home Assistant и создадим две простые автоматизации, которые дадут мгновенный эффект.

Home Assistant уже запущен, но пока это просто красивый интерфейс без реальной пользы. Сейчас мы исправим эту ситуацию и покажем вам две простые автоматизации, которые дадут мгновенный эффект. Никаких сложных настроек, никаких дорогих устройств — только практическая польза, которую можно ощутить уже сегодня вечером.

4.3.1. Пример первый: два контура освещения в квартире без ремонта

Представьте типичную ситуацию: вы живете в квартире, где освещение работает "по старинке". Включили выключатель — лампы в патроне E27 горят на полную яркость. Выключили — темнота. Но почему бы вечером, когда глаза привыкли к полумраку, не сделать свет мягче и уютнее?

В новых домах такую проблему решают прокладкой двух контуров освещения: один для яркого дневного света, другой для приглушенного вечернего. Но что делать, если ваша квартира не новая, а желание иметь умное освещение есть? Ломать стены, штробить каналы под новую проводку, вызывать электриков?

Есть гораздо более изящное решение.

Решение: одна умная лампа вместо новой проводки

Весь секрет — в использовании Zigbee-лампы вместо обычной в патроне E27. Такая лампа получает питание от существующей проводки, но может менять яркость и цветовую температуру по командам из Home Assistant. Выключатель по-прежнему включает и выключает питание, но теперь лампа "помнит", какой режим должен быть активен в зависимости от времени суток.

Что нужно купить:

- Zigbee-лампу с E27 цоколем (около 700–1200 рублей) (рис. 4.3);
- Zigbee-адаптер для Home Assistant (если его еще нет).



Рис. 4.3. ***Пример Zigbee-лампы с цоколем E27. Именно такие устройства позволяют реализовать многоконтурное освещение без смены проводки***

Принцип работы: когда датчик движения подает питание на лампу, она сразу включается в заранее определенном режиме. Home Assistant каждую минуту проверяет время суток и отправляет лампе команду на установку нужной яркости и цветовой температуры. Если сейчас день — включается яркий холодный свет (6500 K, максимальная яркость). Если вечер или ночь — теплый приглушенный свет (2700 K, 30% яркости).

Практическая настройка в Home Assistant

После подключения Zigbee-лампы к системе создаем простую автоматизацию. Вот как она выглядит в текстовом формате:

```
alias: Умное освещение коридора
description: "Два контура освещения без новой проводки"
triggers:
  - trigger: mqtt
    topic: zigbee2mqtt/Лампа/availability
    payload: true
    enabled: false
  - trigger: time_pattern
    seconds: "59"
conditions: []
actions:
  - if:
      - condition: sun
        before: sunset
        after: sunrise
    then:
      - action: light.turn_on
        metadata: {}
        data:
          brightness: 255
          color_temp: 153
        target:
          device_id:
            - 3593e2beeXXXXXX82ddda6e461a23105
            - c5c0efa006XXXXX75b73a28b16057f85
            - 46e690f49bXXXXXXXb77477270ab9587
            - 45235336bXXXXXXXX51a2d9207498c0e
    else:
      - action: light.turn_on
        metadata: {}
        data:
          brightness: 30
          color_temp: 500
```

```
        target:
          device_id:
            - 3593e2beeXXXXXX82ddda6e461a23105
            - c5c0efa006XXXXX75b73a28b16057f85
            - 46e690f49bXXXXXXXb77477270ab9587
            - 45235336bXXXXXXXX51a2d9207498c0e
mode: single
```

Логика автоматизации:

- **Триггер:** каждую 59-ю секунду проверяем настройки.
- **Условие:** если сейчас время между восходом и закатом.
- **Действие днем:** яркость 255 (максимум), цветовая температура 153 майреда (холодный белый ~6500 K).
- **Действие вечером:** яркость 76 (~30%), цветовая температура 500 майред (теплый белый ~2000 K).

Результат: экономия времени, денег и нервов

Что вы получаете:

- **Комфорт:** днем — яркий рабочий свет, вечером — мягкое освещение, которое не режет глаза.
- **Экономия:** ~200 000 рублей, которые потратили бы на демонтаж плитки, новую проводку и восстановительный ремонт.
- **Время:** установка заняла 5 минут вместо недель ремонта.
- **Гибкость:** в любой момент можно изменить настройки через телефон.

4.3.2. Пример второй: автоматические рулонные шторы без проводки 220В

Каждый день одно и то же: утром обходить все комнаты и поднимать шторы, вечером — опускать. Если у вас большая квартира или дом, это превращается в марафон. А если вы в отпуске, закрытые шторы сразу выдают отсутствие хозяев.

Промышленные решения для автоматизации штор требуют подведения питания 220В к каждому окну. Это означает штробление стен, прокладку кабелей, работу электрика. Аналогично дорого, грязно, шумно.

Решение: аккумуляторные приводы

Секрет в том, что современные приводы для рулонных штор работают от встроенного аккумулятора. Энергопотребление у них настолько низкое, что одной зарядки хватает на 2–4 недели при условии открытия/закрытия дважды в день.

Что нужно:

- аккумуляторный привод для рулонных штор (3000–5000 рублей);
- Zigbee-конвертер для интеграции с Home Assistant (1000–1500 рублей) (рис. 4.4).



Рис. 4.4. ***Zigbee-конвертер, установленный в скрытой нише из гипсокартона***

Установка без ремонта:

- Извлекаем цепочку ручного управления из рулонной шторы.
- Вставляем привод внутрь трубы (важно подобрать правильный диаметр: 17, 25 или 36 мм) (рис. 4.5).

- Подключаем Zigbee-конвертер к системе.
- Настраиваем концевые положения через пульт управления.

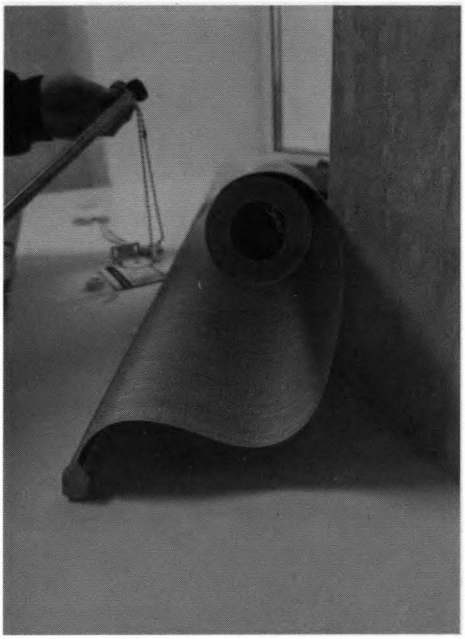


Рис. 4.5. ***Удаление цепочки и установка привода шторы***

Интеграция с Home Assistant

После настройки привода создаем автоматизацию для утреннего и вечернего управления:

```
alias: Автоматические шторы
description: "Открытие на восходе, закрытие на закате"
alias: Шторы на восходе
description: ""
trigger:
  - platform: sun
    event: sunrise
  - platform: time
    at: "07:45:00"
condition:
  - condition: time
```

```
    after: "07:44:00"
  - condition: sun
    after: sunrise
action:
  - device_id: 706fef3dfdf4fe413837f
    domain: cover
    entity_id: 77f2ab59e64dd1e6a4aef62
    type: open
  - device_id: 5b62ef36c019e77ddc4ef5f6
    domain: cover
    entity_id: 5c23bfda6d86d01766f71049
    type: open
  - repeat:
      count: 3
      sequence:
        - device_id: 3236bb67e4c7cd3d9a109c99
          domain: cover
          entity_id: dc5298731da0885abbfb1a
          type: open
        - delay:
            hours: 0
            minutes: 0
            seconds: 30
            milliseconds: 0
        - device_id: 3236bb67e4c77b5109c99
          domain: cover
          entity_id: dc5298731da069685abbfb1a
          type: stop
        - delay:
            hours: 0
            minutes: 0
            seconds: 15
            milliseconds: 0
mode: single
```

Честный разговор о недостатках

Дешевые решения имеют ограничения, и не будем их скрывать:

- Нет точного позиционирования в процентах — только "открыто" или "закрыто".
- Команды не всегда доходят с первого раза, поэтому в автоматизации предусмотрены повторы.
- Разные шторы могут двигаться с разной скоростью.
- Периодически требуется подзарядка аккумулятора.

Но есть и плюсы:

- Установка за 30 минут без грязных работ.
- Работает годами при правильной настройке.
- Легко снимается для обслуживания или замены.
- Стоит в разы дешевле аналогов.

Результат: умный дом, который работает на вас.

4.4. "Разблокируем" бытовую технику: возвращаем умные функции

Home Assistant — это не просто система управления лампочками и датчиками. Это мощный инструмент, который может стать мостом между вами и "отключенной" производителем техникой.

Сегодня мы покажем, как обойти искусственные ограничения и заставить работать даже то оборудование, от которого корпорации решили отказаться.

4.4.1. История одной "мертвой" стиральной машины

Представьте ситуацию: вы купили дорогую стиральную машину Bosch с Wi-Fi и приложением Home Connect (рис. 4.6). Заплатили не только за качество стирки, но и за умные функции — удаленное управление, уведомления, продвинутые режимы. А потом в один момент приложение перестает работать. Не из-за поломки, не из-за устаревания технологий, а из-за геополитических решений.

Именно это произошло с техникой Bosch/Siemens в апреле 2022 года. Приложение Home Connect просто исчезло из российских магазинов приложений, а существующие пользователи получили сообщение "В вашем регионе приложение не поддерживается". Тысячи людей внезапно лишились функций, за которые заплатили реальные деньги.

Но есть способ это исправить — и Home Assistant здесь играет ключевую роль.



Рис. 4.6. ***Колонна из стиральной и сушильной машин***

4.4.2. Практикум: возвращаем Bosch к жизни

Мы не будем углубляться в технические детали, но покажем общую логику решения. Суть в том, что само оборудование исправно и умеет все то же самое, что умело раньше. Проблема только в искусственной блокировке на уровне серверов.

Шаг 1. Обходим географические ограничения

Первое препятствие — установка приложения Home Connect. В российских магазинах приложений его нет, но это легко решается:

- Временно меняем регион в аккаунте Google Play (добавляем аккаунт другой страны).
- Устанавливаем приложение.
- Возвращаем исходные настройки — приложение продолжает работать.

Шаг 2. Подключаем технику к Wi-Fi

Стиральная машина должна быть подключена к домашней сети 2,4 ГГц. Если через приложение не получается ввести пароль (иногда возникают проблемы с кириллицей), используем кнопку WPS на роутере — простое решение без ввода паролей.

Шаг 3. Регистрируемся в системе разработчиков

Чтобы интегрировать технику с Home Assistant, нужно получить ключи API через Home Connect Developer Program:

- Создаем аккаунт через доступ (в нашем случае это означает смену региона в браузере).
- Регистрируем "приложение" и получаем Client ID и Client Secret.
- Эти данные понадобятся для настройки интеграции в Home Assistant.

Шаг 4. Подключаем к Home Assistant

В Home Assistant добавляем интеграцию Home Connect, вводим полученные ключи — и техника появляется в системе со всеми своими датчиками и возможностями управления.

Результат: практическая польза.

После подключения стиральная машина Bosch становится полноценной частью умного дома. Вот простая, но полезная автоматизация:

Уведомление о завершении стирки

```
alias: Стиралка Алиса
description: ""
trigger:
```

```
  - platform: state
    entity_id:
      - binary_sensor.stiralnaia_mashina_door
    to: "off"
condition: []
action:
  - if:
      - condition: time
        after: "08:00:00"
        before: "20:00:00"
        weekday:
          - mon
          - tue
          - wed
          - thu
          - fri
        enabled: true
    then:
      - service: media_player.play_media
        target:
          entity_id: media_player.yandex_station_m00nXXXXXXa23k
        data:
          media_content_id: >-
            {{("давайте постираем", "даешь стирку", "да будет
чистота")|random
            }} в {{ now().hour|round|format(morph='час') }} {{
            now().minute|round|format(morph='минута') }}.
          media_content_type: text
          extra:
            volume_level: 0.6
mode: single
```

Простая логика: когда программа завершается, проговаривается уведомление. Больше никакого забытого мокрого белья в барабане.

Честный разговор об ограничениях

Нужно понимать — это решение не делает технику полностью независимой.

Она по-прежнему зависит от облачных серверов Bosch. Если завтра компания решит заблокировать доступ еще жестче, наш "обход" может перестать работать.

Home Assistant здесь выступает как мост, который возвращает вам управление устройством, но кардинально проблему не решает. Это скорее демонстрация возможностей: при желании можно обойти многие искусственные ограничения и интегрировать "мертвую" технику в единую систему домашней автоматизации.

> **Умный дом** — это не про зависимость от корпораций, а про возвращение контроля над собственными устройствами. Иногда для этого приходится применить немного технической смекалки, но результат того стоит: вы получаете функциональность, за которую заплатили, и можете интегрировать ее в собственную систему автоматизации.

Home Assistant в таких случаях — не просто платформа для умного дома, а инструмент цифровой независимости. Он позволяет создать систему, которая работает на вас, а не на интересы далеких корпораций.

Мы увидели, как Home Assistant помогает вернуть контроль над "закрытой" техникой. Но его возможности гораздо шире. Он позволяет не только управлять существующими устройствами, но и создавать собственные интерфейсы и даже наделять систему интеллектом для анализа происходящего.

4.5. Продвинутый уровень: ваш личный швейцар и охранник

Мы уже убедились, что Home Assistant — это мощный фундамент для независимого умного дома. Но истинная магия начинается там, где стандартных интеграций становится недостаточно. В этом разделе мы выйдем за рамки простого управления устройствами и покажем, как связка Home Assistant и Python позволяет создавать проекты, которые раньше казались фантастикой. Именно здесь мы выполним обещание из аннотации — "узнаем лица всех соседей" — и докажем, что ваш умный дом может быть не только комфортным, но и по-настоящему интеллектуальным.

4.5.1. Создаем пульт для всей семьи: настенная панель управления

Управлять умным домом с телефона удобно, но не всегда. Иногда хочется иметь стационарный пульт — на стене у входа или рядом с кроватью, который доступен всем членам семьи и не требует поиска смартфона. Готовые решения от производителей умных домов стоят дорого и часто привязаны к их экосистемам. Но мы, как всегда, пойдем другим путем.

Телефон может быть разряжен, находиться в другой комнате или быть занят звонком. Гости или дети не всегда имеют доступ к вашему приложению. Стационарная панель решает эти проблемы, предоставляя единую точку входа для управления ключевыми функциями: светом, климатом, шторами или сценарием "выключить все".

Решение: "освобождаем" заводское устройство

Вместо того чтобы собирать панель с нуля, мы возьмем готовое, недорогое и популярное устройство — Sonoff NSPanel (рис. 4.7, 4.8). Это сенсорная Wi-Fi-панель со встроенным микроконтроллером ESP32, двумя физическими кнопками и термометром. "Из коробки" она работает с облачными сервисами производителя, но ее истинный потенциал раскрывается после перепрошивки.

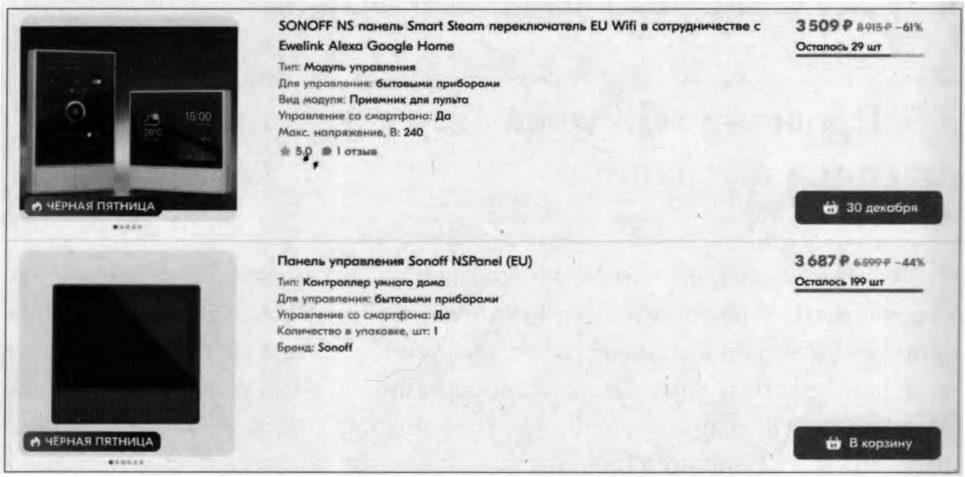


Рис. 4.7. ***Цены на Sonoff NSPanel на популярном маркетплейсе***



Рис. 4.8. ***Sonoff NSPanel — Wi-Fi-панель с контроллером ESP32 и дисплеем Nextion, перепрошитая и отвязанная от китайских сервисов, установленная не в стене, а на подставке***

Используя платформу ESPHome, мы можем полностью отвязать NSPanel от Облака и превратить ее в стопроцентно локальный и гибко настраиваемый пульт для Home Assistant. Это идеальный проект для DIY-энтузиастов: минимум "железной" работы и максимум программной настройки.

Процесс "освобождения" панели выглядит так:

1. **Разборка и подключение.** Панель аккуратно разбирается, чтобы получить доступ к плате. Для прошивки понадобится простой USB-TTL-адаптер, который подключается к специальным контактам на плате.
2. **Прошивка ESPHome.** С помощью веб-инструментов ESPHome или командной строки на микроконтроллер ESP32 заливается кастомная прошивка. Она заменяет заводскую логику и учит панель "общаться" напрямую с вашим сервером Home Assistant.
3. **Обновление интерфейса.** Экран панели (Nextion) имеет собственную прошивку. После установки ESPHome мы загружаем на него новый, кастомный интерфейс — это можно сделать прямо по Wi-Fi через Home Assistant. Существуют готовые проекты, например NSPanel Lovelace UI, которые предлагают красивые и функциональные шаблоны.

4. **Настройка в Home Assistant.** После прошивки панель появляется в Home Assistant как новое устройство. С помощью специальных "чертежей" (Blueprints) вы можете легко настроить, какие устройства и сценарии будут отображаться на экране и какие действия будут выполнять физические кнопки (рис. 4.9).



Рис. 4.9. ***Процесс загрузки шаблона в Nextion начался***

В результате за пару часов вы получаете элегантную настенную панель, которая полностью под вашим контролем. Вы сами решаете, что будет на каждом из экранов: управление светом, климат-контроль, информация о погоде или кнопки запуска сложных автоматизаций. И все это — без единой строчки, отправленной на серверы корпораций.

4.6. Анализ лиц с домофонной камеры

Теперь перейдем к задаче, которая наглядно демонстрирует, зачем в этой книге мы так много внимания уделяем Python. Представьте, что у вас есть архив видеозаписей с камеры домофона за последний месяц. Вам стало интересно, кто чаще всего проходит мимо вашей двери: курьеры, знакомые соседи или незнакомцы? Вручную отсматривать гигабайты видео — титанический труд. А вот поручить эту задачу компьютеру — идеальный проект для связки Home Assistant и Python.

Сразу оговоримся: мы не строим систему постоянной слежки в реальном времени. Наша цель — разово проанализировать существующий архив, чтобы получить статистику и протестировать возможности современных алгоритмов компьютерного зрения. Это скорее "домашняя аналитика данных", чем охранная система.

4.6.1. Почему Python?

Встроенные средства Home Assistant великолепны для автоматизаций, но они не предназначены для таких ресурсоемких задач, как анализ видео и распознавание лиц. Здесь нам нужна "тяжелая артиллерия". Python, с его мощнейшими библиотеками для компьютерного зрения, такими как **dlib** и **face_recognition**, подходит для этого идеально. Они позволяют находить лица на изображениях, преобразовывать их в уникальные "цифровые отпечатки" и сравнивать между собой.

Для быстрой обработки видео нам понадобится вся мощь вашей видеокарты (и она обязательно должна быть). Технология NVIDIA CUDA позволяет перенести вычисления с центрального процессора (CPU) на графический (GPU), ускоряя процесс в десятки, а то и сотни раз. То, что CPU обрабатывал бы часами, GPU сделает за минуты.

4.6.2. Практикум: от видеоархива к отчету

Подготовка окружения: обходим "подводные камни"

Настройка рабочего окружения для компьютерного зрения с поддержкой GPU — задача нетривиальная. Она часто превращается в многодневную борьбу с несовместимыми версиями драйверов, компиляторов и самих библиотек. Чтобы избавить вас от этого, я написал скрипт *setup_env.sh*.

```
#!/bin/bash

set -e

VENV_DIR=".venv"
PYTHON_VERSION_TARGET="3.11.9"

echo "=== Установка окружения и зависимостей ==="
```

```
# --- Проверка наличия pyenv ---
if ! command -v pyenv &> /dev/null; then
    echo -e "\n\033[1;31m[ERROR] pyenv не найден. Установим pyenv перед
запуском.\033[0m"
    exit 1
fi

echo -e "\n[INFO] Выбор версии Python $PYTHON_VERSION_TARGET через
pyenv..."
pyenv local $PYTHON_VERSION_TARGET
echo "[INFO] Текущая версия Python: $(python --version)"

# --- Проверка системных библиотек ---
echo -e "\n[INFO] Проверка и установка системных библиотек для dlib..."
sudo apt update
sudo apt install -y build-essential cmake libopenblas-dev liblapack-dev
libjpeg-dev git

# --- Очистка и создание виртуального окружения ---
if [ -d "$VENV_DIR" ]; then
    echo "[INFO] Удаление старого виртуального окружения '$VENV_DIR'..."
    rm -rf "$VENV_DIR"
fi

echo "[INFO] Создание виртуального окружения '$VENV_DIR'..."
python -m venv "$VENV_DIR"

echo "[INFO] Активация окружения..."
source "$VENV_DIR/bin/activate"

echo "[INFO] Установка Python-зависимостей..."
pip install --upgrade pip
pip install -r requirements.txt

echo "[INFO] Установка dlib с поддержкой CUDA..."
pip install dlib \
    --no-binary :all: \
```

```
    --verbose \
    --config-settings="cmake.args=-DDLIB_USE_CUDA=1"

echo "[INFO] Установка face_recognition..."
pip install face_recognition

echo -e "\n\033[1;32m[OK] Окружение и зависимости успешно
установлены.\033[0m"
```

Вам не нужно вникать в детали компиляции. Просто запустите его, и он сделает все за вас:

- Установит правильную, проверенную версию Python (через pyenv), чтобы избежать конфликтов.
- Загрузит необходимые системные библиотеки для работы с изображениями.
- Создаст изолированное виртуальное окружение, чтобы не засорять систему.
- Самое главное: скачает исходный код библиотеки **dlib** и скомпилирует ее с поддержкой CUDA, чтобы задействовать вашу видеокарту NVIDIA.

Анализ видео: разбираем скрипт *face_report.py*

Основную работу выполняет Python-скрипт. Давайте разберем его ключевые шаги.

Поиск лиц на кадре

Скрипт покадрово читает видеофайлы. Для каждого кадра он использует модель на основе сверточных нейронных сетей (CNN) для поиска лиц. Этот метод точнее, чем классические алгоритмы (например, HOG), и лучше справляется с лицами, повернутыми под углом или в плохом освещении. Благодаря CUDA-ускорению работает это очень быстро.

```
# face_report.py
import argparse
import base64
import cv2
import face_recognition
import numpy as np
```

```
from pathlib import Path
from tqdm import tqdm
import sys
import os

# --- КОНФИГУРАЦИЯ (значения по умолчанию) ---
FACE_COMPARISON_TOLERANCE = 0.6
THUMBNAIL_SIZE = (150, 150)

def create_thumbnail(frame, location):
    """
    Создает иконку лица с увеличенными отступами.
    """
    top, right, bottom, left = location

    # Увеличиваем вертикальный и горизонтальный отступы с 25% до 50% от
размера лица.
    # Это позволяет захватить больше пространства вокруг лица (волосы, лоб,
подбородок).
    pad_y = (bottom - top) // 2
    pad_x = (right - left) // 2

    # Применяем отступы, но следим, чтобы не выйти за границы кадра
    top = max(0, top - pad_y)
    bottom = min(frame.shape[0], bottom + pad_y)
    left = max(0, left - pad_x)
    right = min(frame.shape[1], right + pad_x)

    face_image = frame[top:bottom, left:right]

    # Проверка, не является ли вырезанное изображение пустым
    if face_image.size == 0:
        return None

    resized_face = cv2.resize(face_image, THUMBNAIL_SIZE)
    is_success, buffer = cv2.imencode(".jpg", resized_face)

    if not is_success:
```

```
        return None

    return base64.b64encode(buffer).decode("utf-8")

def generate_html_report(face_metadata, output_path):
    """
    Генерирует HTML-отчет со списком найденных лиц.
    """
    if not face_metadata:
        print("Не найдено ни одного лица для создания отчета.")
        return

    # Сортировка личностей по количеству появлений (по убыванию)
    sorted_metadata = sorted(face_metadata, key=lambda x: x['count'],
reverse=True)

    # Сначала определяем шаблон с именованным плейсхолдером {total_count}
    # и {person_cards}. CSS-стили теперь не будут конфликтовать с
форматированием.
    html_template = """
    <!DOCTYPE html>
    <html lang="ru">
    <head>
        <meta charset="UTF-8">
        <title>Отчет по распознаванию лиц</title>
        <style>
            body {{ font-family: Arial, sans-serif; margin: 20px;
background-color: #f4f4f9; }}
            h1 {{ text-align: center; color: #333; }}
            h2 {{ text-align: center; color: #666; font-weight: normal;
margin-top: -10px; }}
            .container {{ display: flex; flex-wrap: wrap; gap: 20px; justify-
content: center; }}
            .person-card {{
                background-color: white;
                border: 1px solid #ddd;
                border-radius: 8px;
                padding: 15px;
                text-align: center;
```

```
                box-shadow: 0 2px 4px rgba(0,0,0,0.1);
                transition: transform 0.2s;
                width: 200px;
            }}
            .person-card:hover {{ transform: translateY(-5px); }}
            .person-card img {{ border-radius: 8px; width: 150px; height:
150px; object-fit: cover; border: 3px solid #eee; }}
            .person-card h3 {{ margin: 10px 0 5px 0; color: #555; }}
            .person-card p {{ margin: 0; color: #777; }}
            .person-card .count {{ font-size: 1.2em; font-weight: bold;
color: #333; }}
        </style>
    </head>
    <body>
        <h1>Отчет по распознаванию лиц</h1>
        <h2>Всего уникальных личностей: {total_count}</h2>
        <div class="container">
            {person_cards}
        </div>
    </body>
    </html>
    """

    # Генерируем HTML-код для каждой карточки личности
    cards_html = []
    for i, data in enumerate(sorted_metadata):
        person_id = f"Личность #{i + 1}"
        count_text = f"Обнаружен в {data['count']} файлах"
        thumbnail_b64 = data['thumbnail']
        cards_html.append(f"""
            <div class="person-card">
                <img src="{thumbnail_b64}"
alt="{person_id}">
                <h3>{person_id}</h3>
                <p class="count">{count_text}</p>
            </div>
        """)
```

```
    # Собираем итоговый HTML, подставляя данные в шаблон
    final_html = html_template.format(
        total_count=len(sorted_metadata),
        person_cards="\n".join(cards_html)
    )

    with open(output_path, "w", encoding="utf-8") as f:
        f.write(final_html)
    print(f"\nОтчет успешно сохранен в: {output_path}")


def main():
    parser = argparse.ArgumentParser(description="Анализ видеозаписей для
подсчета появлений уникальных лиц.")
    parser.add_argument("--input-dir", required=True, help="Корневой каталог
с папками камер.")
    parser.add_argument("--output-html", default="face_report.html",
help="Путь для сохранения HTML-отчета.")
    parser.add_argument("--model", default="hog", choices=["hog", "cnn"],
help="Модель для детекции лиц: 'hog' (быстрая, CPU) или 'cnn' (точная, GPU).")
    parser.add_argument("--scale", type=float, default=0.5, help="Коэффициент
масштабирования кадра перед детекцией (например, 0.5 для уменьшения в 2
раза, 1.0 для исходного размера).")
    parser.add_argument("--skip-frames", type=int, default=25,
help="Количество пропускаемых кадров между анализами.")
    parser.add_argument("--debug", action="store_true", help="Включить режим
отладки: сохраняет кадры, где должны быть лица.")

    args = parser.parse_args()

    input_path = Path(args.input_dir)
    if not input_path.is_dir():
        print(f"Ошибка: Директория не найдена: {args.input_dir}")
        sys.exit(1)

    print("Поиск видеофайлов .mp4...")
    video_files = sorted(list(input_path.rglob("*.mp4"))) # Сортируем для
более предсказуемого порядка
    if not video_files:
        print(f"В директории {input_path} не найдено файлов .mp4")
```

```
        sys.exit(0)

    print(f"Найдено {len(video_files)} видеофайлов для анализа.")
    print(f"Используемые параметры: модель={args.model}, масштаб={args.scale},
пропуск кадров={args.skip_frames}")

    if args.debug:
        debug_dir = Path("debug_frames")
        debug_dir.mkdir(exist_ok=True)
        print(f"Режим отладки включен. Кадры будут сохраняться в папку:
{debug_dir}")

    known_face_encodings = []
    known_face_metadata = []

    for video_path in tqdm(video_files, desc="Общий прогресс", unit="видео"):
        cap = cv2.VideoCapture(str(video_path))
        if not cap.isOpened():
            tqdm.write(f"Не удалось открыть видео: {video_path.name}")
            continue

        # Этот set будет хранить индексы личностей, уже посчитанных для
ТЕКУЩЕГО видеофайла.
        # Он сбрасывается для каждого нового видео.
        persons_found_in_this_file = set()

        frame_count = 0
        processed_one_frame_for_debug = False
        while cap.isOpened():
            ret, frame = cap.read()
            if not ret:
                break

            frame_count += 1
            if frame_count % args.skip_frames != 0:
                continue

            if args.scale != 1.0:
```

```
                small_frame = cv2.resize(frame, (0, 0), fx=args.scale,
fy=args.scale)
            else:
                small_frame = frame

            rgb_small_frame = cv2.cvtColor(small_frame, cv2.COLOR_BGR2RGB)
            face_locations = face_recognition.face_locations(rgb_small_
frame, model=args.model)

            if args.debug and not processed_one_frame_for_debug and face_
locations:
                debug_frame = frame.copy()
                tqdm.write(f"[DEBUG] Найдено {len(face_locations)} лиц в
{video_path.name}")
                for (top, right, bottom, left) in face_locations:
                    top, right, bottom, left = int(top/args.scale),
int(right/args.scale), int(bottom/args.scale), int(left/args.scale)
                    cv2.rectangle(debug_frame, (left, top), (right, bottom),
(0, 255, 0), 2)
                debug_filename = debug_dir / f"found_{video_path.stem}_
{frame_count}.jpg"
                cv2.imwrite(str(debug_filename), debug_frame)
                processed_one_frame_for_debug = True

            if face_locations:
                current_face_encodings = face_recognition.face_encodings
(rgb_small_frame, face_locations, num_jitters=1)

                scale_inv = 1.0 / args.scale
                for i, face_encoding in enumerate(current_face_encodings):
                    matches = face_recognition.compare_faces(known_face_
encodings, face_encoding, tolerance=FACE_COMPARISON_TOLERANCE)

                    match_index = -1
                    if True in matches:
                        match_index = matches.index(True)

                    # Если личность уже известна
```

```
                    if match_index != -1:
                        # --- ИЗМЕНЕНИЕ: Логика подсчета по файлам ---
                        # Увеличиваем счетчик, только если мы видим эту
личность В ЭТОМ ФАЙЛЕ впервые.
                        if match_index not in persons_found_in_this_file:
                            known_face_metadata[match_index]['count'] += 1
                            persons_found_in_this_file.add(match_index)
                            tqdm.write(f"Личность #{match_index + 1}
повторно обнаружена в новом файле: {video_path.name}")

                    # Если это новая личность
                    else:
                        tqdm.write(f"Найдена новая личность в файле {video_
path.name}")
                        known_face_encodings.append(face_encoding)

                        (top, right, bottom, left) = face_locations[i]
                        original_location = (int(top*scale_inv),
int(right*scale_inv), int(bottom*scale_inv), int(left*scale_inv))

                        thumbnail = create_thumbnail(frame, original_
location)
                        if thumbnail:
                            known_face_metadata.append({
                                "count": 1,
                                "thumbnail": thumbnail
                            })
                            # --- ИЗМЕНЕНИЕ: Логика подсчета по файлам ---
                            # Добавляем новую личность в set для текущего
файла, чтобы не посчитать ее дважды.
                            new_person_index = len(known_face_metadata) - 1
                            persons_found_in_this_file.add(new_person_index)
        cap.release()

    generate_html_report(known_face_metadata, args.output_html)

if __name__ == "__main__":
    main()
```

Превращение лица в "цифровой отпечаток"

Каждое найденное лицо преобразуется в **face_encoding** — вектор из 128 чисел. Это уникальная математическая характеристика лица, не зависящая от ракурса или освещения.

Сравнение и уникализация

Когда скрипт находит новое лицо, он вычисляет его "отпечаток" и сравнивает с уже известными (рис. 4.10). Если вектор достаточно близок к одному из сохраненных (с определенным допуском tolerance), скрипт считает, что это один и тот же человек. Если похожих "отпечатков" нет, он сохраняет новый как уникальную личность. Чтобы избежать многократного подсчета одного и того же человека в одном видео, действует правило: "один файл — один голос".

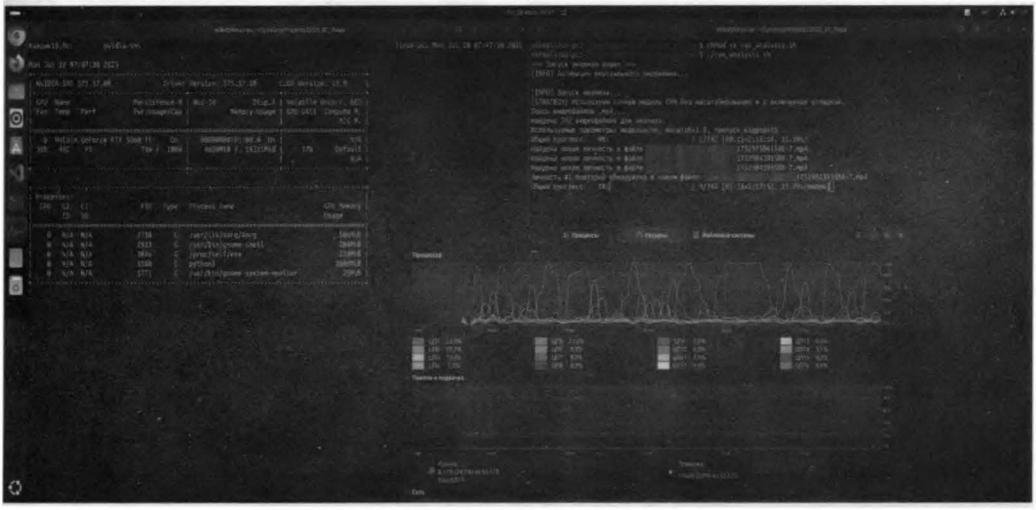


Рис. 4.10. ***Скрипт за работой***

Генерация HTML-отчета

После обработки всех видеофайлов скрипт собирает статистику и генерирует простой, но наглядный HTML-отчет. Для каждой уникальной личности он сохраняет лучшую фотографию-иконку и указывает, сколько раз этот человек появлялся в архиве.

Результат: кто ходит у вашей двери?

На выходе вы получаете веб-страницу, на которой в виде галереи представлены все уникальные посетители, отсортированные по частоте появлений.

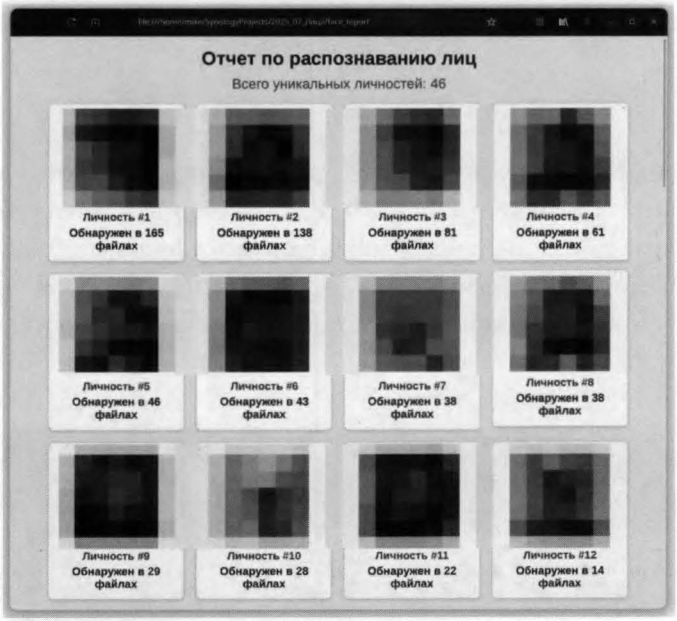


Рис. 4.11. ***Пример итогового HTML-отчета: иконки уникальных лиц и статистика их появлений***

Такой отчет может дать интересные инсайты. Например, вы можете обнаружить, что один и тот же курьер доставляет вам заказы из разных сервисов, или оценить, как часто к вам заходят гости. Конечно, у технологии есть пределы: человек в шапке и без нее может быть определен как две разные личности. Но как инструмент для разового анализа и демонстрации мощи Python — это впечатляющий проект.

Этот пример показывает, как навыки работы с Python, которые вы получили в предыдущих главах, открывают дверь в мир сложных и захватывающих проектов. Там, где возможностей Home Assistant уже не хватает, на помощь приходит универсальный и мощный язык программирования. Вы не просто автоматизируете рутину, а создаете собственные инструменты анализа данных, ограниченные лишь вашей фантазией. Связка "локальный умный дом + Python" дает вам полный контроль не только над устройствами, но и над информацией, которую они генерируют.

4.7. Финальный штрих: избавляемся от Яндекс-Алисы и создаем приватного ассистента

Мы успешно построили локальный умный дом, который работает независимо от капризов корпораций. Устройства подчиняются вам, данные остаются дома, автоматизации исполняются локально. Но есть одно слабое звено — голосовой помощник. Пока вы говорите: "Алиса, включи свет на кухне", ваша команда летит на серверы Яндекса, обрабатывается там, и только потом возвращается обратно для выполнения. Это не только медленно, но и нарушает всю концепцию приватности, ради которой мы строили локальную систему.

4.7.1. Почему пора расстаться с облачными ассистентами

Представьте: каждая ваша команда — это кусочек информации о вашей жизни. "Включи свет в спальне" в 23:00 говорит о том, когда вы ложитесь спать. "Выключи все" — когда уходите из дома. "Поставь температуру 25 градусов" — раскрывает ваши предпочтения и даже состояние здоровья. Умножьте это на месяцы и годы использования — и получится детальный портрет вашего образа жизни, хранящийся на чужих серверах.

Но дело не только в приватности. Облачные ассистенты — это точка отказа в вашей системе. Нет Интернета — нет управления. Сервер перегружен — команды не выполняются. Компания меняет политику — функции могут исчезнуть. Мы уже видели примеры с техникой Bosch, устройствами Сбера, Insteon. Голосовые помощники не исключение.

Пора замкнуть последнюю петлю и создать голосового ассистента, который работает полностью на вашей территории, понимает именно ваши команды и никогда не "сливает" данные третьим лицам.

4.7.2. Анатомия локального ассистента: как это работает изнутри

Чтобы заменить Алису, нужно понять, из чего состоит голосовой помощник.

Это не одна программа, а цепочка специализированных модулей, каждый из которых решает свою задачу:

[Пользователь]
↓ произносит команду
[Микрофон] — захватывает звук
↓
[Wake Word Detection] — слушает ключевую фразу ("Привет, дом!")
↓ активируется
[Speech-to-Text] — превращает речь в текст
↓
[Home Assistant Assist] — понимает намерение и выбирает действие
↓
[Выполнение команды] — включает свет, меняет температуру и т.д.
↓ (опционально)
[Text-to-Speech] — синтезирует голосовой ответ
↓
[Динамик] — воспроизводит ответ

Ключевое отличие от облачных решений: все эти этапы происходят локально, в вашей сети. Никаких запросов в Интернет, никаких задержек на обработку в дата-центрах, никаких рисков утечки данных.

- **Wake Word Detection** постоянно слушает микрофон, но не записывает и не передает звук. Он просто ждет ключевую фразу — например, "Привет, пирожок!" или "Окей, дом!". Можете выбрать любую.
- **Speech-to-Text (STT)** включается только после активации и преобразует вашу команду в текст. Мы будем использовать Whisper — один из лучших современных движков, который отлично работает с русским языком и не требует Интернета.
- **Распознавание намерений** ложится на встроенную в Home Assistant систему Assist. Она анализирует текст команды, понимает, какое устройство имеется в виду и какое действие нужно выполнить.
- **Text-to-Speech (TTS)** превращает ответ системы обратно в речь. Для этого используем Piper — быстрый и качественный синтезатор с хорошими русскими голосами.

Все модули общаются через **Wyoming Protocol** — простой, но мощный протокол, который разработали специально для таких задач.

4.7.3. Практикум: два пути к голосовой независимости

Теперь от теории к практике. Предлагаю два варианта реализации — от простого к продвинутому.

Вариант 1. "Быстрый старт" — все на Raspberry Pi

Если вы хотите попробовать локальное голосовое управление с минимальными вложениями, этот путь для вас. Предупреждаю честно: будет работать медленнее облачных решений, но зато вы почувствуете вкус цифровой независимости.

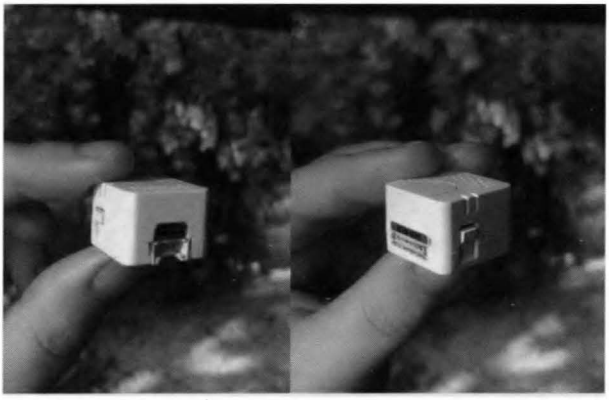


Рис. 4.12. ***M5Stack ATOM Echo. Микроразмер. Фото из Интернета***

Что понадобится:

- ваш существующий Home Assistant на Raspberry Pi 4 (желательно с 4+ ГБ памяти);
- M5Stack ATOM Echo (около 1400 рублей) — крошечное устройство с микрофоном и динамиком (рис. 4.12).

Архитектура:

[M5Stack ATOM Echo] <= Wi-Fi => [Raspberry Pi 4: HA + STT + TTS]

Настройка:

1. **Прошивка ATOM Echo через ESPHome**. В Home Assistant добавляете новое ESPHome-устройство и используете готовый YAML-конфиг для голосового ассистента (рис. 4.13). Примерно такой:

```
substitutions:
  name: voice-assistant-echo
  micro_wake_word_model: okay_nabu

esphome:
  name: ${name}
  friendly_name: Voice Assistant
  min_version: 2024.6.0
  project:
    name: m5stack.atom-echo-voice-assistant
    version: "1.0"

esp32:
  board: m5stack-atom
  framework:
    type: esp-idf

# Конфигурация I2S для микрофона и динамика
i2s_audio:
  i2s_lrclk_pin: GPIO33
  i2s_bclk_pin: GPIO19

microphone:
  - platform: i2s_audio
    id: echo_microphone
    i2s_din_pin: GPIO23
    adc_type: external
    pdm: true

speaker:
  - platform: i2s_audio
    id: echo_speaker
```

```
    i2s_dout_pin: GPIO22
    dac_type: external
    mode: mono

micro_wake_word:
  model: ${micro_wake_word_model}
  on_wake_word_detected:
    - voice_assistant.start:

voice_assistant:
  microphone: echo_microphone
  speaker: echo_speaker
  use_wake_word: false
  on_listening:
    - light.turn_on:
        id: led_ring
        blue: 100%
        red: 0%
        green: 0%
  on_stt_end:
    - light.turn_on:
        id: led_ring
        blue: 0%
        red: 100%
        green: 0%
  on_end:
    - light.turn_off: led_ring
```

2. **Настройка pipeline в Home Assistant**. Добавляете три аддона:
   - **faster-whisper** с моделью *tiny* (для экономии ресурсов);
   - **Piper** с русским голосом;
   - **OpenWakeWord** для активации по ключевой фразе.
3. **Создание голосового конвейера**. В настройках Home Assistant создаете новый Assist pipeline, выбираете установленные сервисы и задаете русский язык.

Плюсы:

- Минимальные затраты времени и денег.
- Все работает на одном устройстве.
- Полная приватность и независимость.
- Можно реализовать за один вечер.

Минусы:

- Заметная задержка (3–10 секунд на команду).
- STT может грузить Home Assistant.
- Качество распознавания ниже облачных решений.
- Сложно масштабировать на несколько комнат.

Этот вариант идеален для знакомства с технологией и понимания ее возможностей.

Вариант 2. "Правильная архитектура" — выносим обработку на отдельный сервер

Это мой рекомендуемый путь для серьезного использования. Мощные задачи обработки речи выносятся на отдельный сервер, а Raspberry Pi остается заниматься только управлением устройствами.

Что понадобится:

- ваш Home Assistant на Raspberry Pi 4;
- Mini PC (Intel N100/N150 или AMD Ryzen 5, 14–17 тыс. рублей);
- ESP32-S3-BOX для гостиной (около 6000 рублей) — с экраном и лучшим звуком (рис. 4.13);
- M5Stack ATOM Echo для дополнительных комнат (по 1400 рублей).



Рис. 4.13. ***ESP32-S3-BOX-3. Фото из Интернета***

Архитектура:

[Пользователь] → [ESP32-устройства] → [Mini PC: STT + TTS + Wake Word]
↓
[Raspberry Pi: Home Assistant]

Настройка сервера обработки речи

Устанавливаете на Mini PC легкую Linux-систему (Ubuntu Server) и разворачиваете Docker Compose с тремя контейнерами:

```
# docker-compose.yml
version: '3.8'

services:
  whisper:
    image: rhasspy/wyoming-whisper
    restart: unless-stopped
    ports:
      - "10300:10300"
    environment:
      - TZ=Europe/Moscow
    command: --model medium --language ru --uri tcp://0.0.0.0:10300
```

```
    volumes:
      - whisper_data:/data

  piper:
    image: rhasspy/wyoming-piper
    restart: unless-stopped
    ports:
      - "10200:10200"
    environment:
      - TZ=Europe/Moscow
    command: --voice ru_RU-ruslan-medium --uri tcp://0.0.0.0:10200
    volumes:
      - piper_data:/data

  openwakeword:
    image: rhasspy/wyoming-openwakeword
    restart: unless-stopped
    ports:
      - "10400:10400"
    environment:
      - TZ=Europe/Moscow
    command: --preload-model alexa --uri tcp://0.0.0.0:10400
    volumes:
      - wake_data:/data

volumes:
  whisper_data:
  piper_data:
  wake_data:
```

Интеграция с Home Assistant

В настройках **Home Assistant** добавляете **Wyoming Protocol** три раза — для каждого сервиса:

1. Whisper STT: IP_MINI_PC:10300;
2. Piper TTS: IP_MINI_PC:10200;
3. OpenWakeWord: IP_MINI_PC:10400.

Затем создаете Voice Pipeline, объединяющий все компоненты, и настраиваете устройства ESP32 на работу с этим *pipeline*.

Настройка "спутников"

ESP32-S3-BOX прошивается аналогично ATOM Echo, но с дополнительными возможностями:

```
# Дополнительно для ESP32-S3-BOX
display:
  - platform: ili9xxx
    model: ili9342
    id: box_display
    # ... конфигурация экрана

# Отображение статуса на экране
on_listening:
  - display_text: "Слушаю..."
on_stt_end:
  - display_text: "Думаю..."
on_tts_start:
  - display_text: "Отвечаю..."
```

Плюсы:

- Быстрая реакция (1–3 секунды).
- Высокое качество распознавания.
- Home Assistant не перегружается.
- Легко добавить новые "спутники".
- Продвинутые возможности (экран, лучший звук).

Минусы:

- Более высокая стоимость.
- Требует базовых навыков Linux и Docker.
- Дополнительное энергопотребление.

Практические советы по настройке

- **Выбор ключевых фраз.** Избегайте слишком коротких ("Дом") или созвучных с обычными словами фраз. "Привет, пирожок!" или "Окей, умный дом" работают хорошо.
- **Именование устройств.** Система лучше понимает простые, описательные названия: "Основной свет спальня" вместо "light.bedroom_ceiling_01". Используйте синонимы через aliases.
- **Тестирование команд.** Начните с простых команд: "Включи свет в кухне", "Выключи все", "Установи температуру 23 градуса". Постепенно добавляйте более сложные.
- **Мониторинг производительности.** Следите за нагрузкой CPU во время распознавания. Если процесс занимает более 5–7 секунд, рассмотрите более мощное железо или облегченную модель Whisper.

Результат: ваш цифровой суверенитет обретает голос.

Конечно, на начальном этапе система может казаться менее "умной", чем Алиса с доступом к поисковой системе и внешним сервисам. Но помните: ваша цель не заменить универсального помощника, а создать идеально настроенный инструмент управления именно вашим домом.

Со временем вы сможете интегрировать локальные LLM-модели для более естественного общения, добавить распознавание эмоций в голосе, настроить персонализированные ответы для разных членов семьи. Но это уже продвинутые возможности — основа локального голосового управления готова.

Теперь ваш умный дом действительно работает на вас, а не на интересы далеких корпораций. От железа до программного обеспечения, от данных до голосовых команд — все находится под вашим полным контролем. Это и есть настоящий цифровой суверенитет в собственном доме.

Заключение

Мы прошли большой путь от осознания рисков облачной зависимости до построения полноценной, автономной экосистемы умного дома.

Давайте оглянемся на главные вехи этого путешествия.

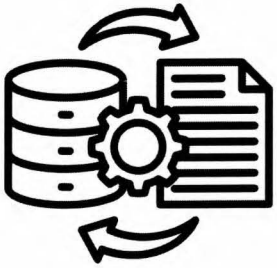
Home Assistant стал тем универсальным "мозгом" и "дирижером", который связал воедино разрозненные устройства из разных экосистем, вернув вам полное управление над ними. Мы не просто заменили один облачный хаб на другой — мы перенесли всю логику, все команды и все данные к себе домой, за брандмауэр своей локальной сети. Мы увидели, что это не только возможно, но и практично: от развертывания системы на Raspberry Pi до создания сложных автоматизаций, решающих реальные бытовые проблемы без дорогостоящего ремонта.

Но на этом мы не остановились. Мы вышли за рамки готовых решений, применив Python для задач, которые раньше казались фантастикой: от анализа видеоархивов до создания собственных голосовых интерфейсов. Это доказало главное: связка "локальная платформа + язык программирования" снимает все ограничения. Ваша система может быть настолько умной и сложной, насколько хватит вашей фантазии.

В итоге мы замкнули последнюю петлю зависимости, создав приватного голосового ассистента, который работает полностью офлайн. Теперь ваши команды никуда не уходят, а ваш распорядок дня остается вашей личной тайной.

Вы построили не просто набор умных гаджетов. Вы построили архитектуру надежности и приватности. Архитектуру, которая будет работать на вас годами, не завися от санкций, бизнес-решений и сбоев в облаках. Ваш дом наконец-то стал по-настоящему вашим — и в физическом, и в цифровом смысле.

Теперь, когда ваш дом обрел цифровой суверенитет и работает как швейцарские часы, самое время посмотреть на другие сферы жизни. Как эти же принципы — автоматизация, локальность и контроль — могут помочь нам решать более личные задачи: управлять семейным расписанием, здоровьем, финансами и личными проектами? Об этом мы поговорим в следующей главе.



Глава 5.

Цифровые лайфхаки для семьи и хобби

Мы уже научили дом включать свет по расписанию. Теперь научим нашу цифровую жизнь работать так же слаженно. Эта глава — о маленьких скриптах-помощниках, которые будут помнить важные даты, помогать в быту и даже снижать напряжение в семейных вопросах. Мы создадим инструменты, экономящие не просто минуты, а ваше внимание и душевное спокойствие для самого главного — близких и любимых дел.

5.1. Простой старт: решаем бытовые мелочи за 5 минут

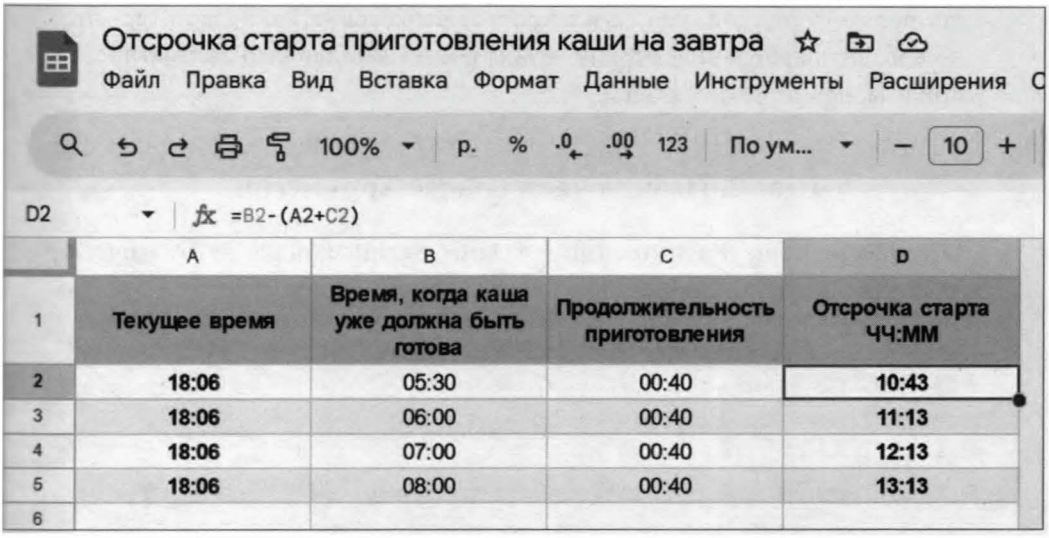
Начнем с чего-то до боли знакомого. Вечер, вы стоите над мультиваркой и пытаетесь прикинуть, на сколько часов поставить таймер, чтобы каша была готова ровно к завтраку. Кажется, задача на уровне второго класса, но если вы встаете в 6:00, готовка занимает 40 минут, а время сейчас — 21:47, то мозг уже не хочет решать эту задачу, он хочет спать. В итоге ставим "на глаз", а утром либо ждем кашу, либо она готова сильно заранее.

Мы решили эту задачу раз и навсегда — с помощью Google Таблиц и нескольких строк кода на Google Apps Script. И получилось удивительно изящно.

Шаг 1. Создаем основу в Google Таблицах

Первым делом открываем новую Google Таблицу и задаем четыре столбца:

1. текущее время;
2. время, когда каша должна быть готова;
3. продолжительность приготовления;
4. и наконец — рассчитанную отсрочку старта (в формате ЧЧ:ММ).



	A	B	C	D
1	**Текущее время**	**Время, когда каша уже должна быть готова**	**Продолжительность приготовления**	**Отсрочка старта ЧЧ:ММ**
2	**18:06**	05:30	00:40	**10:43**
3	**18:06**	06:00	00:40	**11:13**
4	**18:06**	07:00	00:40	**12:13**
5	**18:06**	08:00	00:40	**13:13**
6				

Рис. 5.1. ***Google Таблица с решением***

На **рис. 5.1** показан пример таблицы, которую мы будем использовать.

Текущее время	Время, когда каша уже должна быть готова	Продолжительность приготовления	Отсрочка старта ЧЧ:ММ
18:06	05:30	00:40	10:43
19:06	06:00	00:40	10:13
20:06	07:00	00:40	10:13
21:06	08:00	00:40	10:13

Мы используем встроенную формулу `=TIME(HOUR(NOW()); MINUTE(NOW()); SECOND(NOW()))`, чтобы автоматически подставлять текущее время. Это простая, но важная деталь — таблица "живет" в реальном времени, а значит, всегда показывает актуальные расчеты.

Далее в колонке "Время, когда каша уже должна быть готова" мы вручную указываем, к какому часу хотим получить кашу. Для разных сценариев можно добавить несколько строк — например, если в семье кто-то завтракает в 6:00, а кто-то в 8:00 или позже. В третьей колонке задаем длительность готовки — обычно 40 минут, но можно и больше для других блюд. Теперь главное — посчитать отсрочку старта. Можно сделать это прямо формулой, используя простое выражение, но куда интереснее — добавить небольшой скрипт, чтобы расчет выполнялся автоматически при изменении любой ячейки.

Шаг 2. Подключаем Google Apps Script

Открываем меню **Расширения => Apps Script**. Google автоматически создает для нас файл `Code.gs`, куда вставляем следующий код:

```
function CALCULATE_DELAY(desiredCompletionTime, cookingTime) {
  desiredCompletionTime = desiredCompletionTime.toString();
  cookingTime = cookingTime.toString();
  var currentDateTime = new Date();
  var desiredParts = desiredCompletionTime.split(':');
  var desiredDateTime = new Date(currentDateTime);
  desiredDateTime.setHours(parseInt(desiredParts[0], 10));
  desiredDateTime.setMinutes(parseInt(desiredParts[1], 10));
  desiredDateTime.setSeconds(0);
  if (desiredDateTime <= currentDateTime) {
    desiredDateTime.setDate(desiredDateTime.getDate() + 1);
  }
  var cookingParts = cookingTime.split(':');
  var cookingTimeMinutes = parseInt(cookingParts[0], 10) * 60 +
parseInt(cookingParts[1], 10);
  var startCookingTime = new Date(desiredDateTime.getTime() -
cookingTimeMinutes * 60000);
  var delayMinutes = (startCookingTime - currentDateTime) / 60000;
  var delayHours = Math.floor(delayMinutes / 60);
  var delayRemainingMinutes = Math.round(delayMinutes % 60);
  return ('0' + delayHours).slice(-2) + ':' + ('0' +
delayRemainingMinutes).slice(-2);
}
```

Скрипт делает именно то, что мы делали раньше в уме: берет текущее время, вычитает из целевого времени приготовления длительность готовки, а затем вычисляет, через сколько часов и минут нужно запустить процесс. Все это — без лишних арифметических ошибок и утренних "ой, не так посчитал".

Шаг 3. Проверяем и используем

Проверить работу скрипта можно функцией:

```
function test() {
  CALCULATE_DELAY("05:20", "00:40")
}
```

После выполнения в логах увидим результат, например:

```
Результат: 19:40
```

Это значит, что если сейчас 21:40, то мультиварку стоит запустить через 19 часов 40 минут — чтобы каша была готова ровно к 05:20 утра.

Мы можем использовать эту функцию прямо в ячейке таблицы, как обычную формулу:

```
=CALCULATE_DELAY(B2; C2)
```

Google Таблицы автоматически передадут параметры в скрипт и вернут результат в формате ЧЧ:ММ.

Шаг 4. Почему это важно

На первый взгляд — ерунда, просто каша. Но это идеальный пример того, как автоматизация меняет мышление. Алгоритм понятен? Значит, его можно поручить машине. Сегодня мы автоматизировали таймер для мультиварки. Завтра по этому же принципу создадим напоминание о поливе цветов или авторасчет корма для кота. А если добавить в наш скрипт одну строчку с вызовом API, таблица сможет сама присылать уведомление в Telegram: "Пора закладывать ингредиенты". В этом и есть философия бытовой автоматизации: маленькие шаги дают ощутимый результат уже сегодня.

5.2. Скрипт для напоминания о днях рождения с подсчетом возраста

Продолжим нашу борьбу с мелкими бытовыми раздражителями. Вот еще один: дни рождения в Google Календаре. Функция, безусловно, полезная: Google автоматически подтягивает даты из ваших контактов и создает отдельный календарь "Дни рождения". Но у него есть фундаментальный недостаток — он сообщает лишь факт, но не контекст. Мы видим "День рождения: Иван Петров", но не знаем, какой именно. Ему исполняется 28? Или 38?

В итоге каждое такое напоминание превращается в мини-квест: нужно открыть контакт, найти год рождения, мысленно вычесть его из текущего года и только потом решать, насколько торжественным должно быть поздравление. Это отнимает время и, что важнее, превращает подготовку к теплому, личному событию в рутинную арифметическую задачку. Поздравление становится менее персональным, а риск забыть о важной дате (например, о 30-летии лучшего друга) возрастает.

Решение: "умные" события в календаре с помощью Google Apps Script

Мы исправим это раз и навсегда. Напишем скрипт, который будет работать как наш личный ассистент: он будет регулярно, раз в месяц, просматривать наши Google Контакты, находить все предстоящие дни рождения и создавать для них красивые, информативные события в основном календаре. Главное отличие — в названии события будет сразу указан возраст именинника. Вместо сухого "День рождения: Антон" мы получим яркое и понятное "Антон — день рождения, 34 года".

Инструменты:

- **Google Контакты** как источник данных;
- **Google Календарь** как площадка для отображения результата;
- **Google Apps Script** с подключенным **People API** — наш "мозг" операции, который свяжет все воедино.

Шаг 1. Подготовка данных в Google Контактах.

Прежде чем писать код, убедимся, что у него будут данные для работы. Наш скрипт сможет рассчитать возраст только для тех контактов, у которых в карточке указана полная дата рождения, включая год.

Откройте Google Контакты и проверьте несколько ключевых записей. Убедитесь, что у важных для вас людей (семья, близкие друзья, коллеги) поле "Дата рождения" заполнено корректно в формате ДД.ММ.ГГГГ. Это простое действие — залог успеха всей нашей автоматизации.

Шаг 2. Создаем и настраиваем скрипт.

Теперь переходим к самому интересному. Как и в примере с мультиваркой, открываем Google Таблицы (или просто переходим на сайт script.google.com) и создаем новый проект Apps Script. Удаляем стандартный код-заглушку и вставляем наш уже готовый скрипт.

```
/**
 * @fileoverview Скрипт для Google Apps Script, который создает в Google
Календаре события
 * о днях рождения контактов с указанием их возраста.
 *
 * @version 2.0
*/

// --- НАСТРОЙКИ СКРИПТА ---
const CONFIG = {
  // ID календаря, в который будут добавляться события.
  // Чтобы использовать календарь по умолчанию, оставьте 'default'.
  // Чтобы найти ID другого календаря: зайдите в его настройки, раздел
"Интеграция календаря".
  CALENDAR_ID: 'default',

  // Часовой пояс для корректного определения дат.
  // Список часовых поясов: https://en.wikipedia.org/wiki/List_of_tz_
database_time_zones
  TIME_ZONE: 'Europe/Yekaterinburg',

  // За сколько дней вперед создавать события о днях рождения.
```

```
  // Например, 31 день - события будут созданы на ближайший месяц.
  DAYS_AHEAD_TO_CREATE_EVENTS: 31,

  // Местоположение для событий в календаре (необязательно).
  EVENT_LOCATION: 'Пермь',

  // Настройки уведомлений (в минутах до начала события).
  // 0 = в момент начала (в 00:00), 900 = за 15 часов (в 9:00 предыдущего дня).
  REMINDER_MINUTES: [0, 900],

  // Настройки для файла логов на Google Диске.
  LOG_FILE_SETTINGS: {
    ENABLED: true, // Включить или выключить сохранение логов в файл
    FILE_NAME_SUFFIX: '_BirthdayLogs.txt' // Суффикс для имени файла лога
  }
};

// --- ГЛАВНАЯ ФУНКЦИЯ ---

/**
 * Основная функция. Получает контакты и создает события в календаре для
предстоящих дней рождения.
 */
function createBirthdayEvents() {
  const calendar = getCalendar();
  if (!calendar) return;

  const {
    startDate,
    endDate
  } = getDateRange();
  logScriptRunDates(startDate, endDate);

  const contacts = getAllContactsWithBirthdays();
  if (contacts.length === 0) {
    Logger.log('Не найдено контактов с указанной датой рождения.');
    return;
  }
```

```
  Logger.log('Найдено ${contacts.length} контактов с датой рождения.
Начинаем обработку...');

  let eventsCreated = 0;
  contacts.forEach(contact => {
    const birthdayInfo = getBirthdayInfo(contact, startDate.getFullYear());

    if (birthdayInfo.nextBirthday >= startDate && birthdayInfo.nextBirthday
<= endDate) {
      Logger.log('-> День рождения "${birthdayInfo.name}" (${birthdayInfo.
dateString}) попадает в диапазон.');
      deleteExistingEvents(calendar, birthdayInfo.name, birthdayInfo.
nextBirthday);
      createCalendarEvent(calendar, birthdayInfo);
      eventsCreated++;
    }
  });

  Logger.log('Обработка завершена. Создано/обновлено событий:
${eventsCreated}.');
  if (CONFIG.LOG_FILE_SETTINGS.ENABLED) {
    saveLogToDrive();
  }
}

// --- ВСПОМОГАТЕЛЬНЫЕ ФУНКЦИИ ---

/**
 * Получает объект календаря по ID из настроек.
 * @returns {CalendarApp.Calendar|null} Объект календаря или null в случае
ошибки.
 */
function getCalendar() {
  try {
    const calId = CONFIG.CALENDAR_ID;
    const calendar = calId === 'default' ?
      CalendarApp.getDefaultCalendar() :
      CalendarApp.getCalendarById(calId);
```

```
    if (!calendar) {
      throw new Error('Календарь с ID "${calId}" не найден.');
    }
    Logger.log('Используется календарь: "${calendar.getName()}"');
    return calendar;
  } catch (e) {
    Logger.log('Ошибка при получении календаря: ${e.message}');
    return null;
  }
}

/**
 * Определяет диапазон дат для поиска дней рождения.
 * @returns {{startDate: Date, endDate: Date}} Объект с начальной и конечной
датами.
 */
function getDateRange() {
  const startDate = new Date();
  const endDate = new Date(startDate.getTime() + CONFIG.DAYS_AHEAD_TO_
CREATE_EVENTS * 24 * 60 * 60 * 1000);
  return {
    startDate,
    endDate
  };
}

/**
 * Логирует диапазон дат выполнения скрипта.
 * @param {Date} startDate Начальная дата.
 * @param {Date} endDate Конечная дата.
 */
function logScriptRunDates(startDate, endDate) {
  Logger.log('Скрипт запущен. Поиск дней рождения в диапазоне:');
  Logger.log('С: ${Utilities.formatDate(startDate, CONFIG.TIME_ZONE, 'dd.
MM.yyyy HH:mm')}');
  Logger.log('По: ${Utilities.formatDate(endDate, CONFIG.TIME_ZONE, 'dd.
MM.yyyy HH:mm')}');
}
```

```
/**
 * Получает все контакты пользователя, у которых указана дата рождения,
используя People API.
 * @returns {Array<Object>} Массив объектов контактов.
 */
function getAllContactsWithBirthdays() {
  const allContacts = [];
  let pageToken = null;

  try {
    do {
      consteresponse = People.People.Connections.list('people/me', {
        personFields: 'names,birthdays,phoneNumbers',
        pageSize: 1000,
        pageToken: pageToken
      });

      if (response.connections && response.connections.length > 0) {
        const contactsWithBirthdays = response.connections.filter(person =>
          person.birthdays && person.birthdays.some(b => b.date)
        );
        allContacts.push(...contactsWithBirthdays);
      }
      pageToken = response.nextPageToken;
    } while (pageToken);

  } catch (e) {
    Logger.log('Не удалось получить контакты через People API: ${e.message}');
    // Можно добавить отправку уведомления по email в случае критической
ошибки
    // MailApp.sendEmail('your-email@example.com', 'Ошибка в скрипте дней
рождения', e.message);
  }
  return allContacts;
}

/**
```

```
 * Извлекает и форматирует информацию о дне рождения контакта.
 * @param {Object} person Объект контакта из People API.
 * @param {number} currentYear Текущий год.
 * @returns {Object} Объект с информацией о дне рождения.
 */
function getBirthdayInfo(person, currentYear) {
  const name = person.names && person.names.length > 0 ? person.names[0].
displayName : '[Имя не указано]';
  const birthdayData = person.birthdays[0].date;
  const {
    day,
    month,
    year
  } = birthdayData;

  const nextBirthday = new Date(currentYear, month - 1, day);
  // Если день рождения в этом году уже прошел, берем следующий год
  if (nextBirthday < new Date(new Date().setHours(0, 0, 0, 0))) {
    nextBirthday.setFullYear(currentYear + 1);
  }

  const age = year ? nextBirthday.getFullYear() - year : null;
  const ageText = age ? '${age} ${getAgePostfix(age)}' : '';
  const mobilePhone = person.phoneNumbers ? person.phoneNumbers.find(p =>
p.type === 'mobile') : null;

  return {
    name,
    year,
    age,
    ageText,
    nextBirthday,
    mobilePhone: mobilePhone ? mobilePhone.value : null,
    dateString: '${day}.${month}${year ? '.' + year : ''}'
  };
}

/**
```

```
 * Удаляет старые события для этого же контакта на ту же дату.
 * @param {CalendarApp.Calendar} calendar Объект календаря.
 * @param {string} contactName Имя контакта.
 * @param {Date} eventDate Дата события.
 */
function deleteExistingEvents(calendar, contactName, eventDate) {
  try {
    const existingEvents = calendar.getEvents(eventDate, new Date(eventDate.
getTime() + 1), {
      search: contactName
    });
    if (existingEvents.length > 0) {
      Logger.log('   Удаление ${existingEvents.length} старых событий для
"${contactName}"...');
      existingEvents.forEach(event => event.deleteEvent());
    }
  } catch (e) {
    Logger.log('   Ошибка при удалении старых событий: ${e.message}');
  }
}

/**
 * Создает событие в календаре.
 * @param {CalendarApp.Calendar} calendar Объект календаря.
 * @param {Object} birthdayInfo Информация о дне рождения.
 */
function createCalendarEvent(calendar, birthdayInfo) {
  const eventTitle = '${birthdayInfo.name} - день рождения${birthdayInfo.
ageText ? ', ' + birthdayInfo.ageText : ''}';
  let description = 'Сегодня ${birthdayInfo.name} празднует день
рождения${birthdayInfo.ageText ? ' - ${birthdayInfo.ageText}' : ''}!\n\nC
Днем Рождения! 🍰🎁😊🎈🕺💃';
  if (birthdayInfo.mobilePhone) {
    description += '\n\n☎ ${birthdayInfo.mobilePhone}';
  }

  try {
    const event = calendar.createAllDayEvent(eventTitle, birthdayInfo.
nextBirthday, {
```

```
      description: description,
      location: CONFIG.EVENT_LOCATION
    });
    CONFIG.REMINDER_MINUTES.forEach(minutes => event.
addPopupReminder(minutes));
    Logger.log('  ✓ Событие "${eventTitle}" успешно создано.');
  } catch (e) {
    Logger.log('  ✗ Не удалось создать событие для "${birthdayInfo.name}":
${e.message}');
  }
}

/**
 * Возвращает правильное окончание для возраста ("год", "года", "лет").
 * @param {number} age Возраст.
 * @returns {string} Слово с правильным окончанием.
 */
function getAgePostfix(age) {
  const lastDigit = age % 10;
  const lastTwoDigits = age % 100;

  if (lastTwoDigits >= 11 && lastTwoDigits <= 19) {
    return 'лет';
  }
  if (lastDigit === 1) {
    return 'год';
  }
  if (lastDigit >= 2 && lastDigit <= 4) {
    return 'года';
  }
  return 'лет';
}


// --- ФУНКЦИИ УПРАВЛЕНИЯ ТРИГГЕРАМИ И ЛОГАМИ ---

/**
```

```
 * Создает или обновляет триггер для ежемесячного автоматического запуска
скрипта.
 * Запускается 1-го числа каждого месяца. Рекомендуется запустить эту
функцию один раз вручную.
 */
function setupMonthlyTrigger() {
  const functionName = 'createBirthdayEvents';
  // Удаляем все предыдущие триггеры для этого проекта, чтобы избежать
дублирования
  ScriptApp.getProjectTriggers().forEach(trigger => ScriptApp.
deleteTrigger(trigger));

  // Создаем новый триггер, который будет запускать скрипт 1-го числа
каждого месяца в 1-2 часа ночи
  ScriptApp.newTrigger(functionName)
    .timeBased()
    .onMonthDay(1) // Запускать в 1-й день месяца
    .atHour(1)     // Указываем час запуска
    .create();

  Logger.log('Триггер для функции "${functionName}" успешно создан. Он будет
запускаться ежемесячно, 1-го числа.');
}

/**
 * Сохраняет журнал выполнения (логи) в текстовый файл на Google Диске.
 */
function saveLogToDrive() {
  try {
    const scriptFile = DriveApp.getFileById(ScriptApp.getScriptId());
    const scriptName = scriptFile.getName();
    const parentFolder = scriptFile.getParents().next() || DriveApp.
getRootFolder();
    const logFileName = '${scriptName}${CONFIG.LOG_FILE_SETTINGS.FILE_NAME_
SUFFIX}';

    // Удаляем старый файл лога, если он существует
    const oldLogs = parentFolder.getFilesByName(logFileName);
    if (oldLogs.hasNext()) {
      oldLogs.next().setTrashed(true);
```

```
    }

    // Создаем новый файл с текущими логами
    parentFolder.createFile(logFileName, Logger.getLog());
    Logger.log('Логи сохранены в файл: "${logFileName}" в папке
"${parentFolder.getName()}"');
  } catch (e) {
    Logger.log('Ошибка при сохранении лога на Диск: ${e.message}');
  }
}


// --- ТЕСТОВАЯ ФУНКЦИЯ ---

/**
 * Тестовая функция. Находит и выводит в лог все контакты, у которых есть
дата рождения.
 * Ничего не создает и не изменяет.
 */
function test_listAllBirthdays() {
  Logger.log("--- Начало теста: Поиск всех контактов с днями рождения ---");
  const contacts = getAllContactsWithBirthdays();

  if (contacts.length > 0) {
    contacts.forEach(person => {
      const name = person.names && person.names.length > 0 ? person.
names[0].displayName : '[Имя не указано]';
      const bday = person.birthdays[0].date;
      Logger.log('Найден контакт: ${name} (ДР: ${bday.day || '?'}.${bday.
month || '?'}.${bday.year || '????'})');
    });
    Logger.log('--- Тест завершен. Всего найдено контактов с днем рождения:
${contacts.length} ---');
  } else {
    Logger.log("--- Тест завершен. Контакты с днями рождения не найдены. ---
");
  }
}
```

Перед запуском обратим внимание на блок CONFIG в самом верху. Здесь мы можем легко настроить скрипт под себя, не вникая в логику кода:

- CALENDAR_ID — оставьте 'default', если хотите добавлять события в ваш основной календарь.
- TIME_ZONE — укажите ваш часовой пояс, чтобы избежать ошибок с датами. Например, 'Europe/Moscow'.
- DAYS_AHEAD_TO_CREATE_EVENTS — по умолчанию скрипт ищет дни рождения на 31 день вперед. Можете изменить это значение.
- REMINDER_MINUTES — здесь настраиваются уведомления. Значения [0, 540] означают, что вы получите два напоминания: одно в день события в 00:00, а второе — за 9 часов до полуночи предыдущего дня (то есть в 15:00).

Шаг 3. Разбираем код — что он делает?

Наш скрипт — это хорошо организованный механизм. Давайте кратко разберем его ключевые части:

- **getAllContactsWithBirthdays()** — эта функция обращается к вашим контактам через современный People API. Она запрашивает только нужную информацию (имена и дни рождения), что делает процесс быстрым и эффективным.
- **getBirthdayInfo()** — "сердце" скрипта. Для каждого найденного контакта эта функция вычисляет, когда будет следующий день рождения, и рассчитывает возраст. Именно здесь происходит магия.
- **getAgePostfix()** — маленькая, но важная деталь. Эта функция отвечает за правильные окончания в русском языке: "1 год", "2 года", "5 лет". Благодаря ей заголовки событий выглядят грамотно и естественно.
- **createCalendarEvent()** — функция создает само событие в календаре. Она формирует красивый заголовок (Антон — день рождения, 34 года), добавляет в описание поздравление с эмодзи и номер телефона контакта для удобства.
- **deleteExistingEvents()** — чтобы не создавать дубликаты при каждом запуске, скрипт сначала ищет и удаляет старые события, созданные им же для этого контакта на эту дату.

Шаг 4. Первый запуск и выдача разрешений.

Теперь, когда код на месте, пора его проверить.

1. В верхнем меню редактора выберите функцию **createBirthdayEvents** и нажмите кнопку ▶ **Выполнить**.
2. При первом запуске Google попросит вас предоставить разрешения. Появится окно **Требуется авторизация**. Нажимайте **Просмотреть разрешения**.
3. Вы увидите список того, к чему скрипт запрашивает доступ: к вашим контактам, календарю и Google Диску (для сохранения логов). Это стандартная процедура. Нажмите **Разрешить**.
4. После этого скрипт выполнится. Откройте ваш Google Календарь. Если у кого-то из ваших контактов день рождения в ближайший месяц, вы увидите новое, красивое событие!

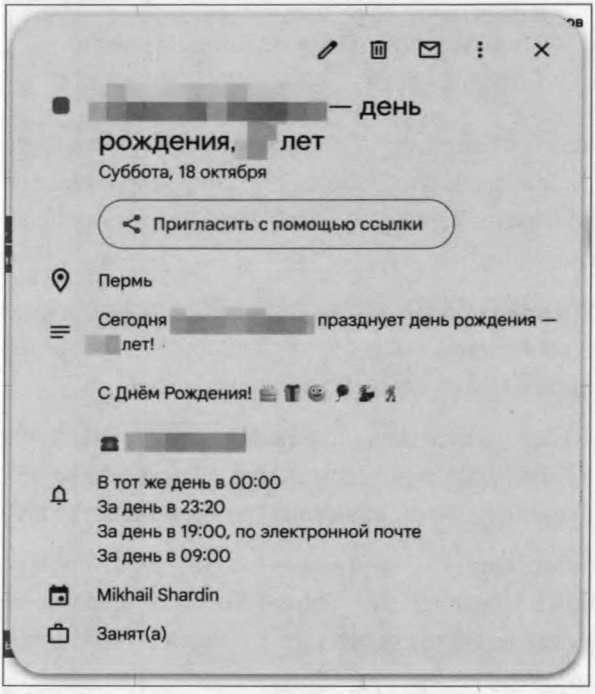


Рис. 5.2. ***Пример события, созданного нашим скриптом. Сразу виден возраст, и есть быстрый доступ к номеру телефона***

> **Важное замечание**: многих может смутить этический вопрос: стоит ли акцентировать внимание на возрасте? Важно помнить: это напоминание — только для вас.

Скрипт дает вам полную информацию, а как ее использовать — поздравлять ли с 34-летием или просто написать "С днем рождения!" — вы решаете сами.

Шаг 5. Автоматизация. Настраиваем триггер

Чтобы не запускать скрипт вручную каждый месяц, мы настроим его автоматический запуск.

1. В редакторе скриптов перейдите в раздел **Триггеры** (иконка будильника слева).
2. Нажмите кнопку **+ Добавить триггер** в правом нижнем углу.
3. Настройте его так:
   - » Выберите функцию для запуска: createBirthdayEvents.
   - » Выберите источник события: на основе времени.
   - » Выберите тип триггера на основе времени: счетчик месяцев.
   - » Выберите день месяца: 1-й день месяца.
   - » Выберите время дня: с 1 до 2.
4. Нажмите **Сохранить**.

Готово! Теперь каждое первое число месяца, пока вы спите, Google будет сам запускать наш скрипт. Он просканирует контакты и заблаговременно создаст в вашем календаре напоминания о днях рождения на весь предстоящий месяц. Мы только что делегировали машине еще одну рутинную задачу, освободив свой мозг для более важных дел.

5.3. Продвинутый уровень: автоматизация для сложных жизненных сценариев

Мы уже научились автоматизировать мультиварку и дни рождения — задачи простые, почти игрушечные.

Но жизнь, как известно, не ограничивается кашей и поздравлениями. Иногда автоматизация нужна не для того, чтобы сэкономить пять минут, а чтобы сохранить душевное равновесие в ситуациях, где каждая мелочь может стать источником конфликта.

Развод — одна из таких ситуаций. Особенно когда есть дети. Бывшие супруги пытаются выстроить новую жизнь, но при этом остаются родителями, которым нужно координировать школу, врачей, кружки и десятки других дел. И здесь начинается самое сложное: как организовать это взаимодействие так, чтобы не нарушать границы новой семьи, не перегружать партнеров чужой информацией и при этом не забыть ни одной важной детали, касающейся ребенка?

Сегодня мы разберем реальный кейс, который показывает: автоматизация — это не только про продуктивность, но и про человечность, про уважение к личному пространству каждого и про то, как технологии могут снизить эмоциональную нагрузку в самых непростых жизненных обстоятельствах.

5.4. Скрипт для координации жизни ребенка после развода

Представим ситуацию: вы разведены, но остаетесь активным родителем. Ребенок живет поочередно — неделю у вас, неделю у бывшего супруга.

Вы, возможно, создали новую семью. И теперь перед вами стоит задача-головоломка: как синхронизировать три совершенно разных мира?

- **Первый мир** — это ваше общение с бывшим супругом. Здесь только самое важное: когда и где передаете ребенка, приемы у врачей, родительские собрания, поездки к бабушке. Никаких лишних подробностей, только координация.
- **Второй мир** — это ваша новая семья. Ваш партнер имеет право знать, когда ребенок будет дома, чтобы планировать совместные дела, ужины, выходные. Но при этом нет никакого смысла загружать его информацией о том, что происходит в "чужую" неделю, когда ребенок живет у бывшего супруга.

- **Третий мир** — это семейный календарь, где отражается все, что касается ребенка в ваши недели: тренировки, занятия с репетитором, дни рождения одноклассников.

Ручное управление всем этим превращается в работу диспетчера: постоянно нужно помнить, что и куда записать, кого предупредить, что скопировать. Один забытый визит к стоматологу — и начинаются звонки, упреки, испорченные планы. Но самое неприятное — это ощущение, что ты стал "курьером" между двумя семьями, тратишь силы не на близких, а на бесконечную координацию.

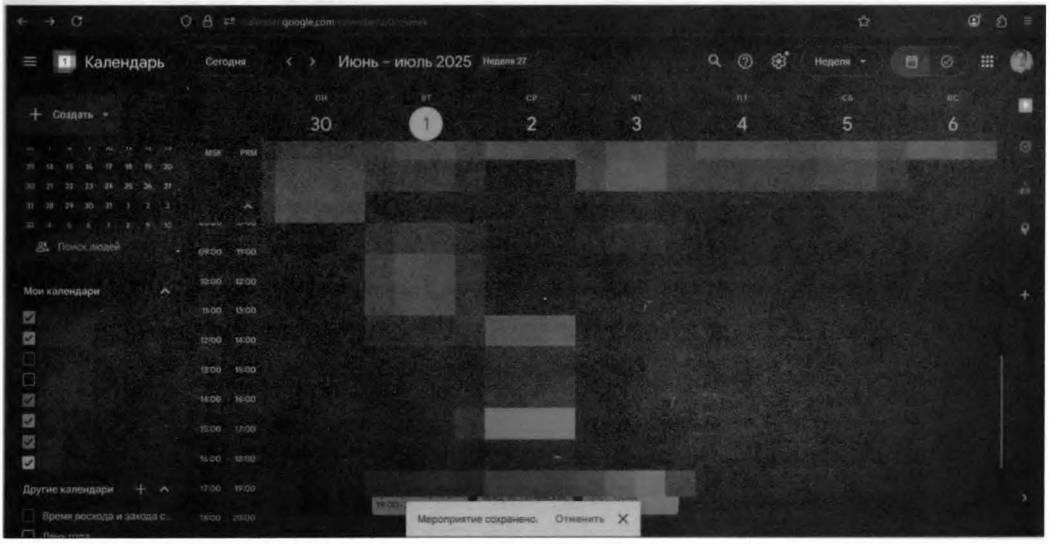


Рис. 5.3. ***Пример множества календарей***

Решение: умная фильтрация событий с помощью Google Apps Script

Мы создадим систему из трех Google Календарей, связанных автоматическим скриптом (рис. 5.3). Логика простая и элегантная:

- **Календарь с бывшим супругом** содержит все ключевые события, связанные с ребенком.
- **Маркерное событие** — это специальная запись, которая появляется в понедельник каждой "вашей" недели. Например, событие с названием "Максим на Березовой аллее" (адрес вашего дома). Это сигнал для скрипта: эта неделя — ваша.

- **Скрипт анализирует** календарь каждое воскресенье вечером. Если в понедельник есть маркерное событие — значит, предстоящая неделя ваша, и все события из календаря с бывшим супругом автоматически копируются в семейный календарь. Если маркера нет — ребенок у бывшего супруга, и ничего не копируется.

Результат: ваш партнер видит в семейном календаре только то, что имеет к нему отношение. Никакой лишней информации, никакого дискомфорта. При этом вы не забываете ни одного важного дела.

Шаг 1. Создаем структуру календарей

Откройте Google Календарь и создайте (или используйте существующие):

- **Календарь для координации с бывшим супругом.** Назовем его "Дела сына". Сюда добавляются все события, связанные с ребенком: врачи, школа, переезды между домами.
- **Семейный календарь.** Например, "Наша семья". Сюда скрипт будет автоматически копировать события в ваши недели.

Чтобы получить ID каждого календаря, откройте его настройки (три точки рядом с названием => **Настройки и доступ**) и пролистайте до раздела **Интеграция календаря**. Скопируйте строку вида `example123@group.calendar.google.com` — это и есть ID календаря.

Шаг 2. Создаем маркерное событие

В календаре "Дела сына" создайте повторяющееся событие, которое будет появляться каждый понедельник вашей недели. Например:

- **Название:** "Максим на Березовой аллее" (имя ребенка + ваш адрес или любая другая узнаваемая фраза).
- **Повторение:** каждые две недели, начиная с понедельника вашей первой недели.
- **Время:** например, с 00:00 до 23:59 (событие на весь день).

Это событие — ключ ко всей системе. Скрипт будет искать его, чтобы понять, ваша ли это неделя.

Шаг 3. Пишем и настраиваем скрипт

Откройте Google Apps Script (перейдите на `script.google.com` или через любую Google Таблицу: Расширения => Apps Script). Вставьте следующий код:

```
// Получаем список идентификаторов всех доступных календарей
function getCalendars() {
    var calendars = CalendarApp.getAllCalendars();
    Logger.log('Этот пользователь подписан на %s календарей:', calendars.
length);
    for (var i = 0; i < calendars.length; i++) {
        var calendar = calendars[i];
        Logger.log((i + 1) + 'й календарь: "' + calendar.getName() + '",\n
ID: "' + calendar.getId() + '"\n');
    }
}

// Синхронизация из календаря сына в семейный
function syncChildEventsToFamily() {
  const CHILD_CALENDAR_ID = 'example123@group.calendar.google.com';
  const FAMILY_CALENDAR_ID = 'family456@group.calendar.google.com';

  const today = new Date();
  const nextMonday = new Date(today);
  nextMonday.setDate(today.getDate() + ((1 - today.getDay() + 7) % 7));
  nextMonday.setHours(0, 0, 0, 0);

  const mondayEnd = new Date(nextMonday);
  mondayEnd.setHours(23, 59, 59, 999);

  const weekEnd = new Date(nextMonday);
  weekEnd.setDate(weekEnd.getDate() + 6);
  weekEnd.setHours(23, 59, 59, 999);
```

```
const childCalendar = CalendarApp.getCalendarById(CHILD_CALENDAR_ID);
const mondayEvents = childCalendar.getEvents(nextMonday, mondayEnd);

// Ищем события с определенным местоположением
const specialLocationEvents = mondayEvents.filter(event =>
  event.getTitle().toLowerCase().includes("на березовой аллее")
);

if (specialLocationEvents.length === 0) {
  Logger.log("Неделя не ваша. Событие с 'на Березовой аллее' не найдено.");
  return;
}

const familyCalendar = CalendarApp.getCalendarById(FAMILY_CALENDAR_ID);
const existingFamilyEvents = familyCalendar.getEvents(nextMonday, weekEnd);
let copiedCount = 0;

// 🔁 Функция для проверки дубликатов
function isDuplicate(event, existingEvents) {
  return existingEvents.some(famEvent =>
    famEvent.getTitle() === event.getTitle() &&
    famEvent.getStartTime().getTime() === event.getStartTime().getTime()
  );
}

// ✅ Копируем событие "на Березовой аллее"
specialLocationEvents.forEach(event => {
  if (!isDuplicate(event, existingFamilyEvents)) {
    familyCalendar.createEvent(
      event.getTitle(),
      event.getStartTime(),
      event.getEndTime(),
      {
        description: event.getDescription(),
        location: event.getLocation(),
        guests: event.getGuestList().map(g => g.getEmail()).join(",")
      }
```

```
      );
      Logger.log('Добавлено событие (Березовая аллея): ${event.getTitle()}
(${event.getStartTime()} - ${event.getEndTime()})');
      copiedCount++;
    }
  });

  const weekEvents = childCalendar.getEvents(nextMonday, weekEnd);

  // 🔁 Копируем остальные события, кроме уже добавленных специальных
  weekEvents.forEach(event => {
    const isSpecialLocation = specialLocationEvents.some(specialEvent =>
      specialEvent.getTitle() === event.getTitle() &&
      specialEvent.getStartTime().getTime() === event.getStartTime().
getTime()
    );

    if (isSpecialLocation) return; // ⚠ Пропускаем — уже скопировано

    if (!isDuplicate(event, existingFamilyEvents)) {
      familyCalendar.createEvent(
        event.getTitle(),
        event.getStartTime(),
        event.getEndTime(),
        {
          description: event.getDescription(),
          location: event.getLocation(),
          guests: event.getGuestList().map(g => g.getEmail()).join(",")
        }
      );
      Logger.log('Добавлено событие: ${event.getTitle()} (${event.
getStartTime()} - ${event.getEndTime()})');
      copiedCount++;
    }
  });

  Logger.log('Всего добавлено событий: ${copiedCount}');
}
```

Обратите внимание: замените `CHILD_CALENDAR_ID` и `FAMILY_CALENDAR_ID` на реальные ID ваших календарей, а также укажите вашу фразу-маркер в переменной `MARKER_PHRASE`.

Шаг 4. Настраиваем автоматический запуск

Чтобы скрипт работал без вашего участия:

1. В редакторе Apps Script перейдите в раздел **Триггеры** (иконка будильника слева).
2. Нажмите + **Добавить триггер**.
3. Настройте:
   - **Функция**: `syncChildEventsToFamily`.
   - Источник события: на основе времени.
   - Тип триггера: еженедельный таймер.
   - День недели: воскресенье.
   - Время: с 20:00 до 21:00.
4. Сохраните.

Теперь каждое воскресенье вечером скрипт будет автоматически проверять, ваша ли предстоящая неделя, и при необходимости копировать все события в семейный календарь.

Почему это работает и зачем это нужно

Эта система решает сразу несколько задач.

- Во-первых, она снимает с вас роль "человека-моста", который вручную переносит информацию из одного мира в другой.
- Во-вторых, она защищает личное пространство вашего партнера: он видит только то, что касается вашей совместной жизни с ребенком, и не перегружается чужими делами.

- В-третьих — и это, пожалуй, главное — она убирает эмоциональное напряжение, которое неизбежно возникает, когда границы между "старой" и "новой" семьей размыты.

Автоматизация здесь — это не просто удобство. Это акт уважения ко всем участникам ситуации: к бывшему супругу, к новому партнеру и, конечно, к ребенку, который заслуживает того, чтобы его родители действовали слаженно, без лишних конфликтов.

5.5. Автоматизируем хобби и укрощаем цифровой быт

Мы научились освобождать себя от семейных координационных задач — теперь пора взяться за то, что приносит нам радость. Хобби и увлечения часто становятся заложниками информационного хаоса: данные разбросаны по десяткам сайтов, форматы неудобны, поиск работает через раз. В итоге мы тратим больше времени на организацию досуга, чем на сам досуг. Давайте исправим это.

Возьмем конкретный пример — любительский спорт. Допустим, мы увлекаемся бегом, велосипедом или плаванием и хотим найти ближайшие соревнования. Казалось бы, простая задача. Но реальность такова: информация размазана по множеству сайтов (RussiaRunning, MyRace, IRONSTAR, X-WATERS и другим), у каждого своя логика поиска, и нигде нет общей карты или удобного фильтра по расстоянию от вашего города.

Классическая ситуация: я живу в Перми, вбиваю в поиск "Пермь" — ничего не нахожу. А соревнование есть, просто оно проходит в поселке Юг, который находится в 40 минутах езды, но формально это другой населенный пункт. Результат? Упускаю возможность, хотя она была рядом.

Решение: единая таблица всех соревнований с автоматическим обновлением

Мы напишем скрипт на Google Apps Script, который раз в неделю обойдет несколько популярных сайтов с анонсами соревнований, соберет данные о названии, дате и месте проведения и сложит все это в одну Google Таблицу.

Получится персональный агрегатор — всегда актуальный, всегда под рукой, без рекламы и лишних кликов.

Инструменты:

- Google Таблицы как база данных.
- Google Apps Script для автоматизации сбора.
- Библиотека Cheerio для парсинга HTML-страниц.

Шаг 1. Создаем структуру таблицы

Открываем новую Google Таблицу (или создаем копию https://docs.google.com/spreadsheets/d/1mjlngbQ8rcv1jJR3YbC8ilsMWOCJK2TRlw8z9V6CyNo/edit?gid=947050992#gid=947050992) и создаем четыре вкладки: одну общую (назовем ее "Все соревнования") и три технических — по одной на каждый источник данных (например, "russiarunning.com", "myrace.info", "iron-star.com"). Технические вкладки будут заполняться скриптом автоматически, а в общую мы выведем консолидированный список через формулу QUERY (рис. 5.4).

A1 | =QUERY(

	A	B	C	D
1	Ссылка	Дата	Название	Место
2	https://russiarunning.com/event/MoushnMa	06.10.2025	Моушн Марафон похудения с призовым фондом октябрь	online
3	https://russiarunning.com/event/10ETAPFIN	08.10.2025	10 ЭТАП ФИНАЛ_Уфа-Здоровая среда_Триатлон	Мелькомбинат
4	https://russiarunning.com/event/kids10/	09.10.2025	Детский тренировочный тур в Новгородскую область	д.Ершино Новгородская область
5	https://russiarunning.com/event/3mountains	10.10.2025	Мангазея ТРИ ГОРЫ Осень	Курорт Газпром
6	https://russiarunning.com/event/DilijanRun2	10.10.2025	DILIJAN RUN 2025	Армения, г. Дилижан, ул. Гетапня, 7,
7	https://iron-star.com/event/anapa-vino-trail-i	11.10.2025	ANAPA VINO TRAIL интерактивный квест	Анапа
8	https://iron-star.com/event/anapa-vinograd-	11.10.2025	ANAPA VINOGRAD детский забег	Анапа
9	https://iron-star.com/event/anapa-vino-trail-i	11.10.2025	ANAPA VINO TRAIL APERITIF 2ML	Анапа
10	https://russiarunning.com/event/IRONSTAR	11.10.2025	IRONSTAR ANAPA VINO TRAIL интерактивный квест	Анапа
11	https://russiarunning.com/event/cbu9/	11.10.2025	CBU 9 Сухум	Город Сухум
12	https://russiarunning.com/event/motogryzz3	11.10.2025	"Энергия спорта"	Территория спортивной мотоциклетн
13	https://russiarunning.com/event/IRONSTAR	11.10.2025	IRONSTAR ANAPA VINO TRAIL карнавальный забег APEF	Анапа
14	https://russiarunning.com/event/kolpinocros	11.10.2025	Колпинский кросс 2025	Колпинский парк на территории о. Чу
15	https://russiarunning.com/event/Yablochnyy	11.10.2025	Яблочный Трейл 2025	Республика Крым, с.Грушевка
16	https://russiarunning.com/event/66/	11.10.2025	Гонка на лыжероллерах в гору - Царь горы	Сочи
17	https://russiarunning.com/event/beermileirk	11.10.2025	Первая Пивная Миля в Иркутске	Динамо, Карпинская улица, 119
18	https://russiarunning.com/event/KhVOSTAT	11.10.2025	ХВОСТАТЫЙ ДОБРЫЙ КРОСС 2025	Липецк
19	https://russiarunning.com/event/Kraskiosen	11.10.2025	ТРЕЙЛ КРАСКИ ОСЕНИ	г. Москва, Салтыковский лесопарк
20	https://russiarunning.com/event/Zavidovotre	11.10.2025	"Завидово" трейл	Комплекс отдыха Завидово
21	https://russiarunning.com/event/FestivalVys	11.10.2025	II Фестиваль трейлраннинга "Вядь. Сурский трейл"	Пенза
22	https://russiarunning.com/event/lyzherollery	11.10.2025	Соревнования по лыжероллерам Old School на призы ков	г.Кинешма, ул Завокзальная, 27
23	https://russiarunning.com/event/IRONSTAR	11.10.2025	IRONSTAR ANAPA VINOGRAD детский забег	Анапа
24	https://russiarunning.com/event/AquathlonS	11.10.2025	Детский акватлон "Шестигранник"	Москва
25	https://russiarunning.com/event/Osen/	11.10.2025	Первоуральск Опен 2025 (Осень)	Первоуральск, Свердловская област
26	https://russiarunning.com/event/ESENINBE	11.10.2025	ЕСЕНИНСКИЙ ПРОБЕГ	Рыбное

+ ≡ Сводная myrace.info iron-star.com russiarunning.com

Рис. 5.4. ***Таблица поиска любительских соревнований***

В каждой технической вкладке создаем заголовки столбцов:

Ссылка | Дата | Название | Город

Это стандартная структура, которую вернет наш парсер.

Шаг 2. Подключаем библиотеку Cheerio

Cheerio — это легкая библиотека для парсинга HTML, аналог jQuery, но для серверной среды. Она позволяет выбирать элементы страницы по CSS-селекторам, как мы делаем это в браузере.

Открываем редактор Apps Script (Расширения => Apps Script), переходим в настройки проекта (шестеренка слева) и добавляем библиотеку с ID: `1ReeQ6WO8kKNxoaA_O0XEQ589cIrRvEBA9qcWpNqdOP17i47u6N9M5Xh0`. Выбираем последнюю версию и сохраняем.

Шаг 3. Пишем функции парсинга

Каждый сайт требует своего подхода. Например, для сайта iron-star.com мы загружаем HTML-страницу, находим блоки с событиями по классу `.event-item`, извлекаем из них дату, название, город и ссылку, затем записываем данные в соответствующую вкладку таблицы. Код выглядит примерно так:

```
function scrapeIronStar() {
  const url = 'https://iron-star.com/event/';
  const html = UrlFetchApp.fetch(url).getContentText();
  const $ = Cheerio.load(html);
  const events = $('.event-item-wrap a.event-item');

  const data = [];
  events.each((i, el) => {
    const link = 'https://iron-star.com' + $(el).attr('href');
    const date = $(el).find('.date').text().trim();
    const name = $(el).find('.title').text().trim();
    const city = $(el).find('.place').text().trim();
    data.push([link, date, name, city]);
```

```
  });

  const sheet = SpreadsheetApp.getActiveSpreadsheet().
getSheetByName('iron-star.com');
  sheet.getRange(2, 1, sheet.getMaxRows(), 4).clearContent();
  if (data.length) sheet.getRange(2, 1, data.length, 4).setValues(data);
}
```

Для других сайтов логика аналогична, меняются только селекторы и структура данных. Некоторые сайты, например russiarunning.com, предоставляют JSON API — там вместо парсинга HTML мы просто отправляем POST-запрос и получаем готовые данные для личных нужд.

Именно этот код написал не я, а Александр Иванов https://contributor.pw/ — эксперт по облачным продуктам и автоматизации.

Полностью код выглядит так:

```
/* Поиск ближайших любительских соревнований по бегу, плаванию,
велосипедному и другим видам спорта

Код, представленный здесь, представляет собой приложение, написанное
на Google Apps Script, которое предназначено для сбора информации о
любительских спортивных соревнованиях с различных веб-сайтов:

    https://russiarunning.com
    https://myrace.info
    https://iron-star.com/event/

Приложение автоматически обходит списки соревнований на каждом из сайтов,
извлекает данные о названии соревнования, дате проведения и городе, где
оно проходит. Затем эти данные используются для создания сводной таблицы
предстоящих соревнований.

function run() {
  userActionScrapeMyraceInfo();
  userActionScrapeIronStarCom();
  userActionScrapeRussiarunningCom();
}
```

```
function userActionScrapeIronStarCom() {
  console.log('userActionScrapeIronStarCom', 'start');
  // Получаем новые ссылки
  const url = 'https://iron-star.com/event/';
  const contentList = UrlFetchApp.fetch(url).getContentText();
  const $ = Cheerio.load(contentList);
  const itemList = $('.event-item-wrap a.event-item');

  const values = [];

  itemList.map((_, item) => {
    const $item = Cheerio.load(item);
    const date = $item('.date').text().trim();
    const name = $item('.title').text().trim();
    const city = $item('.place').text().trim();
    const url = 'https://iron-star.com${$(item).attr('href')}';
    values.push([url, date, name, city]);
  });

  const book = SpreadsheetApp.getActiveSpreadsheet();
  const sheet = book.getSheetByName('iron-star.com');

  sheet.getRange(3, 1, sheet.getMaxRows() - 1, sheet.getMaxColumns()).
clearContent();

  if (values.length) {
    sheet.getRange(2, 1, values.length, values[0].length).setValues(values);
  } else {
    sheet.getRange('2:2').clearContent();
  }
  console.log('userActionScrapeIronStarCom', 'finish');
}


function userActionScrapeMyraceInfo() {
  console.log('userActionScrapeMyraceInfo', 'start');
```

```
  // Получаем новые ссылки
  const url = 'https://myrace.info/take/';
  const contentList = UrlFetchApp.fetch(url).getContentText();
  const $ = Cheerio.load(contentList);
  const itemList = $('.container-content.centered > a.events-list__item.row');

  const values = [];

  itemList.map((_, item) => {
    const $item = Cheerio.load(item);

    const date = $item('.date')
      .text()
      .replace(/(\d+)-(\d)+/, '$1');
    const name = $item('h2').clone().children().remove().end().text().trim();
    const city = $item('.flag').text().trim();
    const url = 'https://myrace.info${$(item).attr('href')}';
    values.push([url, date, name, city]);
  });

  const book = SpreadsheetApp.getActive();
  const sheet = book.getSheetByName('myrace.info');

  sheet.getRange(3, 1, sheet.getMaxRows() - 1, sheet.getMaxColumns()).
clearContent();

  if (values.length) {
    sheet.getRange(2, 1, values.length, values[0].length).setValues(values);
  } else {
    sheet.getRange('2:2').clearContent();
  }
  console.log('userActionScrapeMyraceInfo', 'finish');
}


/* exported userActionScrapeRussiarunningCom */
function userActionScrapeRussiarunningCom() {
```

```
  console.log('userActionScrapeRussiarunningCom', 'start');
  const url = 'https://russiarunning.com/api/events/list/ru';
  const payload = {
    Take: 500,
    DateFrom: new Date().toISOString().split('T')[0],
  };
  const options = {
    method: 'post',
    contentType: 'application/json',
    payload: JSON.stringify(payload),
    muteHttpExceptions: true,
  };
  const data = UrlFetchApp.fetch(url, options);

  const items = JSON.parse(data.getContentText()).Items;

  const values = items.map((item) => {
    const { c, d, t, p } = item;
    const link = 'https://russiarunning.com/event/${c}/';
    const date = d.split('T')[0];
    const row = [link, date, t, p];
    return row;
  });

  const book = SpreadsheetApp.getActive();
  const sheet = book.getSheetByName('russiarunning.com');

  sheet.getRange(3, 1, sheet.getMaxRows() - 1, sheet.getMaxColumns()).
clearContent();

  if (values.length) {
    sheet.getRange(2, 1, values.length, values[0].length).setValues(values);
  } else {
    sheet.getRange('2:2').clearContent();
  }
  console.log('userActionScrapeRussiarunningCom', 'finish');
}
```

Шаг 4. Собираем все воедино

Создаем главную функцию `run()`, которая последовательно вызывает все функции парсинга. Затем в общей вкладке **Все соревнования** используем формулу QUERY, чтобы объединить данные из всех источников и отфильтровать только будущие события:

```
=QUERY({russiarunning.com!A:D; myrace.info!A:D; 'iron-star.com'!A:D},
"SELECT * WHERE Col2 >= date '"&TEXT(TODAY();"yyyy-MM-dd")&"' ORDER
BY Col2 ASC")
```

Эта формула берет данные из трех вкладок, выбирает только те события, чья дата больше или равна сегодняшней, и сортирует их по возрастанию.

Шаг 5. Настраиваем автоматический запуск

Чтобы не обновлять таблицу вручную, создаем триггер: в редакторе Apps Script переходим в раздел **Триггеры**, добавляем новый, выбираем функцию `run()`, тип триггера — еженедельный, день — воскресенье, время — например, с 8 до 9 утра. Готово. Теперь каждое воскресенье скрипт сам обновит данные.

Почему это важно

Мы создали инструмент, который экономит не просто время на поиск — он меняет сам подход к планированию досуга. Вместо хаотичного брожения по сайтам мы открываем одну таблицу и сразу видим полную картину: что, где, когда. Можем отфильтровать по городу, отсортировать по дате, сохранить интересные события в отдельный список.

Этот же подход работает для любого хобби, где информация разбросана: концерты, выставки, мастер-классы, распродажи. Меняются только источники данных и селекторы — логика остается той же. Мы перестаем быть заложниками чужих интерфейсов и создаем свой, идеально заточенный под наши потребности.

5.6. Когда менять шины? Python и API погоды для автовладельца

До сих пор мы работали в экосистеме Google. А теперь давайте выйдем за ее пределы и напишем небольшой, но полезный автономный скрипт на Python.

Каждую весну и осень перед автовладельцами встает один и тот же вопрос: пора ли менять шины? Днем — плюс десять, асфальт сухой, солнце светит. Кажется, что зима закончилась и можно смело записываться в шиномонтаж. Но ночью температура все еще опускается к нулю, а иногда и ниже. И вот мы стоим перед дилеммой: переобуться сейчас или подождать еще неделю?

Проблема не только в износе резины. Летние шины на холодной дороге теряют эластичность и становятся "дубовыми" — тормозной путь увеличивается, управляемость падает. Один неожиданный заморозок ранним утром — и безобидная поездка превращается в лотерею. С другой стороны, зимние шины в теплую погоду быстрее изнашиваются и тоже теряют сцепление из-за чрезмерной мягкости.

> Есть **простое правило**: если ночная температура стабильно держится выше +5–7 °C, можно переходить на летнюю резину. Но как понять, стабильно ли? Открыть погоду на неделю вперед, пролистать каждый день, запомнить минимумы, сравнить их с порогом... Да, можно. Но мы же автоматизируем жизнь, верно?

Мы напишем короткий Python-скрипт, который обратится к открытому погодному API, получит прогноз минимальных ночных температур на неделю вперед и даст однозначный ответ: можно менять или стоит подождать. Никаких догадок, никаких "кажется, что уже тепло" — только данные.

Инструменты простые: Python (достаточно базовой установки), библиотека `requests` для HTTP-запросов и бесплатный API Open-Meteo, который не требует регистрации и выдачи ключей доступа. Весь скрипт занимает около 30 строк кода и работает за пару секунд.

Шаг 1. Находим координаты своего города

API Open-Meteo работает по координатам: широта и долгота. Узнать их просто — заходим на сайт *open-meteo.com*, нажимаем кнопку **Search** в документации и вводим название своего города. Система сама подскажет координаты.

Например, для Перми это:

» Широта: `58.00`

» Долгота: `56.22`

Для Москвы:

» Широта: `55.76`

» Долгота: `37.68`

Эти значения мы подставим в наш скрипт.

Шаг 2. Пишем код

Создаем файл `tire_change.py` и вставляем следующий код:

```
import requests
from datetime import datetime

# Задайте координаты вашего города.

# Москва:
# latitude = 55.76
# longitude = 37.68

# Пермь:
latitude = 58.00
longitude = 56.22

# Порог ночной температуры, ниже которого считаем, что еще рано менять
шины
threshold_temp = 1  # градусов Цельсия
```

```
# Запрашиваем данные погоды на 7 дней вперед у Open-Meteo
url = (
    "https://api.open-meteo.com/v1/forecast"
    f"?latitude={latitude}&longitude={longitude}"
    "&daily=temperature_2m_min"
    "&timezone=auto"
)

response = requests.get(url)
data = response.json()

# Получаем минимальные температуры по дням
dates = data["daily"]["time"]
temps = data["daily"]["temperature_2m_min"]

# Анализируем, будут ли заморозки
print("Погода на ближайшие 7 дней:\n")
frost_expected = False

for date, temp in zip(dates, temps):
    date_formatted = datetime.strptime(date, "%Y-%m-%d").
strftime("%d.%m.%Y")
    print(f"{date_formatted}: минимальная температура ночью {temp}°C")
    if temp < threshold_temp:
        frost_expected = True

print("\nВывод:")
if frost_expected:
    print("❄ Ночью ожидаются заморозки. Смена шин пока не
рекомендуется.")
else:
    print("✅ Заморозков не ожидается. Можно менять зимние шины на
летние.")
```

Запускаем скрипт командой `python tire_change.py` — и через пару секунд получаем ясный ответ (рис. 5.5).

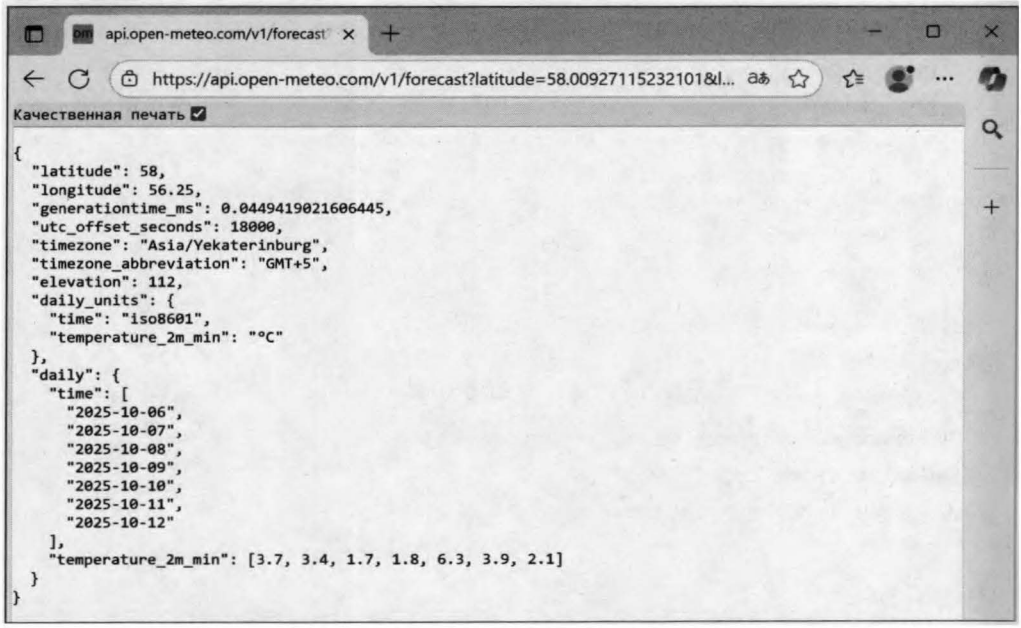


Рис. 5.5. ***Ответ API Open-Meteo:*** *api.open-meteo.com/v1/forecast?latitude=58.00927115232101&longitude=56.2228553210843&daily=temperature_2m_min&timezone=auto*

Шаг 3. Интерпретируем результат

Скрипт выводит минимальную температуру для каждой ночи в ближайшие семь дней. Если хотя бы в одну из ночей температура опускается ниже порога (по умолчанию +5 °C), скрипт сообщает: "Менять шины пока не рекомендуется". Если все ночи теплые — дает зеленый свет.

Мы можем изменить порог в переменной `threshold_temp`. Например, поставить `7` для более консервативного подхода или `3`, если готовы рискнуть ради ранней переобувки.

Почему это работает лучше, чем "просто посмотреть погоду"?

На первый взгляд может показаться: зачем писать скрипт, если можно открыть Яндекс.Погоду и пролистать неделю? Вот несколько причин, почему автоматизация здесь уместна:

1. **Скорость принятия решения.** Скрипт мгновенно анализирует все семь дней и выдает вердикт. Не нужно вручную сравнивать цифры и держать их в голове.

2. **Регулярность.** Мы можем запускать скрипт каждое утро (например, через **Планировщик задач**) и получать актуальную рекомендацию, не открывая браузер.
3. **Объективность.** Мы не игнорируем неудобные данные. Бывает, что человек видит "+10 днем" и думает "да ладно, уже тепло", а ночью -2 °C. Скрипт не дает себя обмануть.
4. **Основа для расширения.** Этот код — заготовка. Мы можем добавить отправку результата в Telegram, интегрировать его в умный дом или связать с напоминанием в календаре: "Через 3 дня можно менять шины — записаться в шиномонтаж".

Шаг 4. Расширяем идею — уведомления в Telegram

Представим, что мы хотим получать уведомление каждое утро: пора ли уже менять шины или стоит подождать. Для этого добавим в скрипт отправку сообщения в Telegram через bot API.

Создаем бота через @BotFather, получаем токен и ID своего чата (можно узнать через @userinfobot). Добавляем в конец скрипта:

```
import requests

def send_telegram_message(message):
    bot_token = "ВАШ_ТОКЕН_БОТА"
    chat_id = "ВАШ_CHAT_ID"
    url = f"https://api.telegram.org/bot{bot_token}/sendMessage"
    payload = {"chat_id": chat_id, "text": message}
    requests.post(url, json=payload)

# После анализа температур формируем сообщение
if frost_expected:
    msg = "❄ Заморозки еще возможны. Шины менять рано."
else:
    msg = "✅ Погода стабильная. Можно менять шины!"

send_telegram_message(msg)
```

Теперь настраиваем автоматический запуск скрипта через `cron` (на Linux/Mac) или Планировщик заданий (на Windows) – например, каждое утро в 8:00. И вот мы уже получаем персонального погодного советника прямо в мессенджер.

5.7. Сканируем в домашнем офисе: превращаем односторонний сканер в двусторонний

Представим ситуацию: перед вами лежит толстая папка рабочих договоров. Все это напечатано с двух сторон, листов — несколько десятков. У вас есть сканер с автоподачей, но он умеет сканировать только одну сторону за раз (рис. 5.6).



Рис. 5.6. ***Инструкция, которую надо сканировать, и она частично распечатана с двух сторон***

Что обычно делают? Кладут документ в автоподатчик, сканируют лицевые стороны, потом вручную переворачивают каждый лист и сканируют оборотные. Затем — самое мучительное — открывают PDF-редактор и начинают вручную перетаскивать страницы, пытаясь восстановить правильный порядок: первая лицевая, первая оборотная, вторая лицевая, вторая оборотная... К двадцатой странице уже путаешься, к пятидесятой — хочется все бросить.

Мы решим эту задачу элегантно и быстро — с помощью простого, но мощного инструмента командной строки **pdftk**. Никаких ручных перестановок страниц. Вся работа займет две команды и пару минут.

Инструменты:

- Сканер с автоподачей (любой).
- **pdftk** — бесплатная консольная утилита для работы с PDF.
- Терминал (командная строка).

5.7.1. Часть 1: магия двустороннего сканирования

Начнем с самого сканирования. Здесь есть один хитрый трюк, который делает всю последующую автоматизацию возможной.

Шаг 1. Сканируем лицевые стороны

Расшиваем документ (убираем все скрепки и скобы), кладем стопку в автоподатчик и сканируем все листы подряд. Сканер идет по порядку: первый лист, второй, третий... последний. Сохраняем результат в файл с понятным названием — например, `Лицевая.pdf`.

Шаг 2. Переворачиваем всю стопку целиком

Вот тут начинается самое интересное. Берем всю пачку отсканированных листов — не по одному, а целиком! — и переворачиваем ее. Теперь то, что было внизу стопки, оказалось сверху. А то, что было лицевой стороной, стало оборотной.

Шаг 3. Сканируем оборотные стороны

Кладем перевернутую стопку обратно в автоподатчик и снова сканируем. Важный момент: теперь сканер идет по *обратному порядку* страниц. То, что было последним листом документа, сканируется первым. Сохраняем в файл `Оборотная.pdf`.

Теперь у нас два PDF-файла:

- `Лицевая.pdf` — страницы идут с 1-й по N-ю;
- `Оборотная.pdf` — страницы идут с N-й по 1-ю (в обратном порядке).

И задача превращается в математически красивую: нужно взять первую страницу из первого файла, последнюю — из второго, вторую — из первого, предпоследнюю — из второго, и так далее.

Знакомимся с *pdftk*

PDFtk (PDF Toolkit) — это "швейцарский нож" для работы с PDF из командной строки. Он умеет объединять файлы, разделять их, поворачивать страницы, шифровать документы и многое другое.

Главное его достоинство — скорость и надежность. Инструмент существует больше 15 лет и проверен миллионами пользователей.

Установить **pdftk** просто:

- **Windows:** скачиваем установщик с сайта *pdflabs.com/tools/pdftk-the-pdf-toolkit* и запускаем его.
- После установки проверяем, что все работает: открываем командную строку (или Терминал) и вводим `pdftk`. Если программа установлена правильно, вы увидите справку по командам.

Шаг 4. Склеиваем два файла в один правильный документ

Теперь самое интересное. Открываем командную строку (в Windows — `cmd` или PowerShell, в macOS и Linux — Terminal), переходим в папку с нашими файлами командой `cd` и вводим волшебную команду:

```
pdftk A="Лицевая.pdf" B="Оборотная.pdf" shuffle A1-end Bend-1
output "Готовый документ.pdf"
```

Разберем, что здесь происходит:

- `A="Лицевая.pdf"` — мы говорим **pdftk**: "Первый файл будем называть A";
- `B="Оборотная.pdf"` — "второй файл – это B";
- `shuffle` — команда "перетасовать", то есть чередовать страницы;
- `A1-end` — берем страницы из **файла A** от первой до последней;

- `Bend-1` — берем страницы из **файла B** от последней к первой (обратный порядок!);
- `output "Готовый документ.pdf"` — результат сохраняем в новый файл.

Программа работает мгновенно. Даже для документа на 200 страниц весь процесс занимает 2–3 секунды. Если все прошло успешно, **pdftk** ничего не выводит — просто создает готовый файл (рис. 5.7). Открываем его — и видим правильный порядок страниц: лицевая, оборотная, лицевая, оборотная...

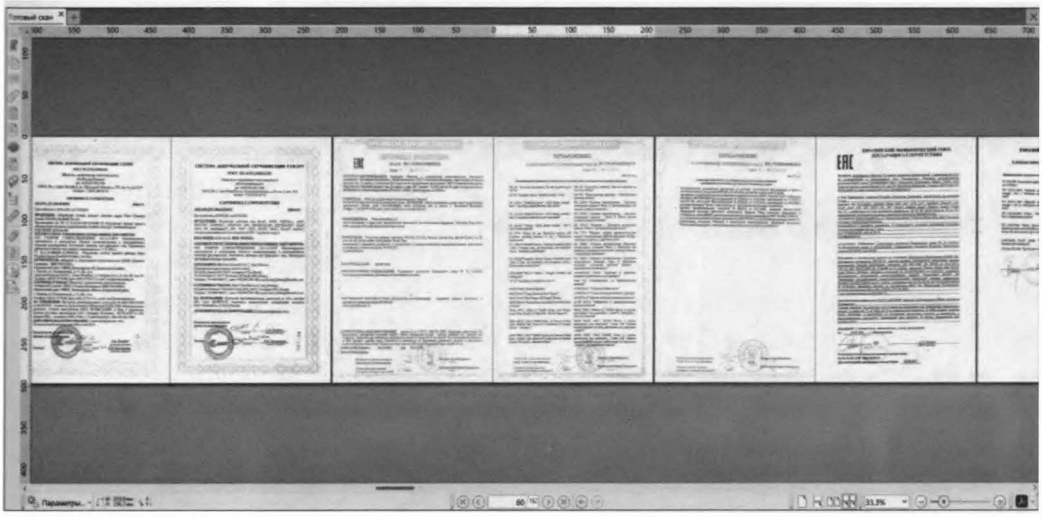


Рис. 5.7. ***Результат работы** pdftk**: страницы автоматически расставлены в правильном порядке. Готовый скан на 162 страницы***

Или в виде запуска двусторонней склейки *.bat* под Windows:

```
@echo off

:: Attempt to get LocalDateTime using WMIC
for /F "usebackq tokens=1,2 delims==" %%i in ('wmic os get
LocalDateTime /VALUE 2^>NUL') do (
    if '.%%i.'=='.LocalDateTime.' set ldt=%%j
)
```

```
:: If WMIC fails, fallback to PowerShell
if not defined ldt (
    echo WMIC failed. Falling back to PowerShell.
    for /F %%i in ('powershell -NoProfile -Command "Get-Date
-Format yyyyMMddHHmmss"') do set ldt=%%i
)

:: Verify that ldt was successfully set
if not defined ldt (
    echo Failed to retrieve datetime. Exiting.
    pause
    exit /b 1
)

:: Format ldt as YYYY-MM-DD_HH-MM
set ldt=%ldt:~0,4%-%ldt:~4,2%-%ldt:~6,2%_%ldt:~8,2%-%ldt:~10,2%
echo %COMPUTERNAME%, %ldt%

:: Use pdftk to combine PDFs
pdftk A="2025-10-06-0001.pdf" B="2025-10-06-0002.pdf" shuffle A1-end
Bend-1 output "%ldt%_scan.pdf"

:: Pause to allow review
pause
```

5.7.2. Часть 2: автоматическая уборка пустых страниц

Мы получили правильно собранный документ, но есть одна проблема. Часто в документах встречаются односторонние листы: титульный лист, разделы, таблицы. При нашем способе сканирования их оборотные стороны тоже попадают в файл — как пустые белые страницы (или, точнее, сканы дырок от дырокола и случайных пятен).

Можно, конечно, открыть PDF в редакторе и вручную удалить эти страницы. Но если документов много, а в каждом десятки таких пустых страниц — это снова превращается в рутину. Автоматизируем и это.

Проблема обнаружения пустоты

Как понять, что страница пустая? Казалось бы, просто: если на ней ничего нет. Но для компьютера это не так очевидно. PDF со сканом — это растровое изображение. Даже "пустая" страница содержит шумы от сканера, тени, возможно — отверстия от дырокола или скрепки. Написать алгоритм, который визуально определит пустоту — сложная задача.

Но мы воспользуемся простым и надежным признаком — размером файла. Страница с текстом или изображениями "весит" значительно больше, чем почти пустая. Если после сжатия PDF-страница занимает меньше 35–40 килобайт — скорее всего, на ней ничего важного нет.

Пишем скрипт для очистки

Создадим bash-скрипт, который автоматически:

1. Разберет PDF на отдельные страницы.
2. Проверит размер каждой.
3. Удалит (точнее, переместит в отдельную папку) "легкие" страницы.
4. Соберет оставшиеся обратно в документ.
5. Создаем файл `remove_blank_pages.sh`:

```
#!/bin/bash

# Удаление пустых страниц из PDF после двустороннего сканирования

datetime=$(date +"%Y-%m-%d_%H-%M-%S")
log_file="removed_pages_$datetime.log"
touch "$log_file"
mkdir -p removed

echo "Начинаем обработку PDF-файлов..." | tee -a "$log_file"

for file in *.pdf; do
  echo "Работаем с файлом: $file" | tee -a "$log_file"
```

```
  # Разбиваем PDF на отдельные страницы
  pdftk "$file" burst output "${file%.*}_page_%04d.pdf"

  # Проверяем размер каждой страницы
  for page in "${file%.*}"_page_*.pdf; do
    size=$(wc -c < "$page")

    # Если страница меньше 35 КВ - считаем ее пустой
    if [[ $size -lt 35000 ]]; then
      echo "  Удаляем: $page (размер: $size байт)" | tee -a "$log_file"
      mv "$page" "removed/"
    fi
  done

  # Склеиваем оставшиеся страницы
  pdftk "${file%.*}"_page_*.pdf cat output "${file%.*}_clean.pdf"
compress

  # Удаляем временные файлы
  rm "${file%.*}"_page_*.pdf doc_data.txt

  echo "Готово: ${file%.*}_clean.pdf" | tee -a "$log_file"
  echo "" | tee -a "$log_file"
done

echo "Обработка завершена. Все удаленные страницы в папке removed/" |
tee -a "$log_file"
```

Как работает скрипт:

- `pdftk "$file" burst` — разбивает PDF на отдельные файлы: `page_0001.pdf`, `page_0002.pdf` и так далее;
- `wc -c < "$page"` — команда, которая возвращает размер файла в байтах;
- Если размер меньше 35000 байт (≈35 КБ), страница перемещается в папку `removed/pdftk ... cat output` — собирает оставшиеся страницы обратно в один файл;
- в конце удаляются все временные файлы.

Запускаем скрипт

В Linux или macOS:

```
chmod +x remove_blank_pages.sh
./remove_blank_pages.sh
```

В Windows (через Git Bash или WSL):

```
./remove_blank_pages.sh
```

Скрипт обработает все PDF-файлы в текущей папке. Для каждого создается новый файл с суффиксом `_clean.pdf`. Все удаленные страницы окажутся в папке `removed/` — мы можем быстро пролистать их, чтобы убедиться, что случайно не удалили что-то важное.

Наш подход:

- **Полностью локальный** — документы не покидают ваш компьютер.
- **Быстрый** — обработка даже больших файлов занимает секунды.
- **Надежный** — основан на измеримом параметре (размер файла), а не на сложном распознавании образов.
- **Прозрачный** — мы видим логи, можем проверить удаленные страницы, настроить порог размера.

И самое главное — это решение масштабируется. Один раз настроили скрипт — и можем обрабатывать хоть тысячи документов. Просто кладем все PDF в одну папку и запускаем.

Мы превратили обычный односторонний сканер в почти полноценный двусторонний. Процесс сканирования 50-страничного документа теперь выглядит так:

1. Сканируем лицевые стороны — 2 минуты.
2. Переворачиваем пачку, сканируем оборотные — 2 минуты.
3. Запускаем одну команду pdftk — 3 секунды.
4. Запускаем скрипт очистки — 5 секунд.
5. Проверяем результат — 30 секунд.

Итого: 5 минут вместо часа ручной работы. И это только для одного документа. А когда таких документов десятки — экономия времени становится огромной.

Заключение

В этой главе мы прошли путь от простейшего расчета времени для мультиварки до сложных сценариев координации семейной жизни после развода. Каждый из этих примеров объединяет одна идея: автоматизация — это не только про экономию времени, но и про освобождение внимания.

Мы научились делегировать компьютеру десятки мелких решений, которые раньше требовали от нас постоянного участия. Google Apps Script взял на себя напоминания о днях рождения с точным указанием возраста. Python и API погоды подсказали оптимальный момент для смены шин. Командная строка и pdftk превратили односторонний сканер в почти профессиональный инструмент оцифровки. А умная фильтрация календарей помогла сохранить личные границы в сложной ситуации совместного воспитания детей.

Все эти решения объединяет простота реализации. Ни один из скриптов не требовал глубоких знаний программирования — только понимание логики задачи и готовность потратить полчаса на настройку. Но результат превосходит затраченные усилия: мы перестаем быть диспетчерами собственной жизни и получаем пространство для того, что действительно важно.

Автоматизация хобби показала, как можно избавиться от информационного хаоса, собрав разрозненные данные в единый удобный источник. Теперь поиск соревнований, концертов или мастер-классов не требует брожения по десяткам сайтов — все собрано в одной таблице, которая обновляется сама.

Главный урок этой главы: технологии могут и должны служить человечности. Автоматизация — это не холодная оптимизация ради оптимизации, а способ создать пространство для тепла, внимания к близким и времени на то, что приносит радость. Мы научились строить цифровых помощников, которые работают незаметно, но меняют качество жизни ощутимо.

В следующей главе мы сделаем еще один шаг вперед: научимся не просто автоматизировать задачи, а анализировать свои собственные данные, превращаясь в персональных аналитиков для самих себя. Мы узнаем, как извлекать смыслы из цифровых следов нашей жизни и принимать более осознанные решения на основе фактов, а не интуиции. Не переключайтесь!



Глава 6.

Персональный аналитик: управляем личными данными

Представьте: вы несколько месяцев складывали все чеки в коробку — неважно, бумажные или электронные. Теперь настал момент высыпать их на стол и понять, куда уходят деньги.

Все то же самое происходит и с цифровыми данными. Мы их копим — в банках, приложениях, таблицах — но редко анализируем. Эта глава посвящена тому, чтобы наконец вытащить данные "из коробки" и научиться извлекать из них пользу.

> Мы сдвигаем фокус: от **автоматизации действий** — к **автоматизации анализа**. Компьютер и таблицы умеют делать гораздо больше, чем просто считать и хранить. При правильной настройке они могут стать вашими **персональными аналитиками** — помогать видеть закономерности, контролировать расходы, оптимизировать решения.

В этой главе вы научитесь:

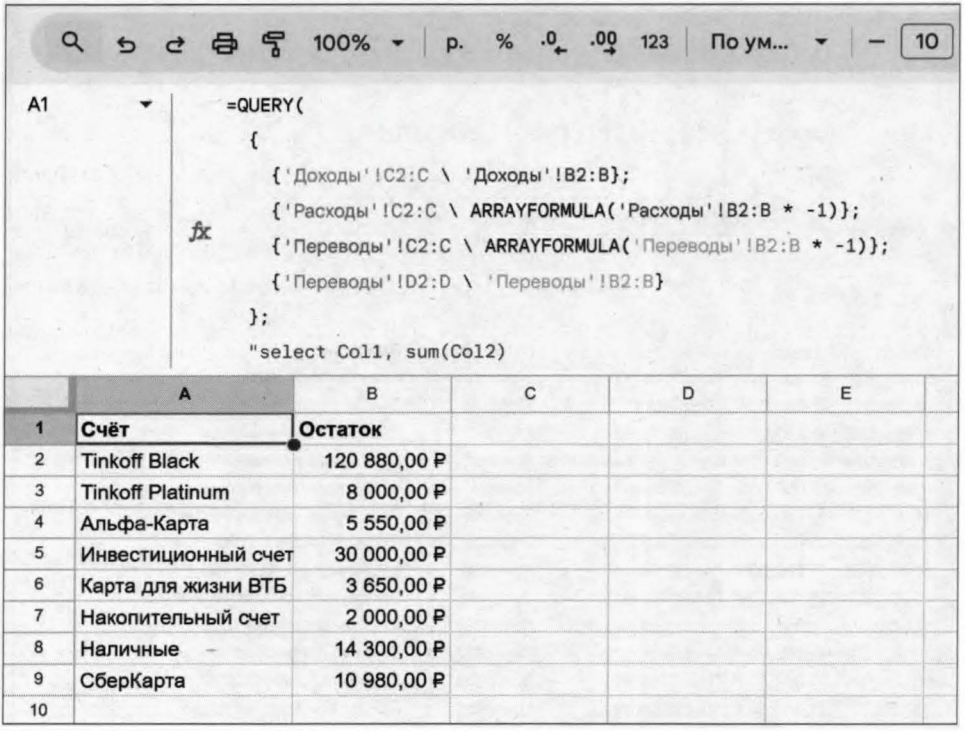
- собирать все финансовые данные в одном месте — в Google Таблицах;
- объединять их и анализировать без сложных макросов, используя всего одну функцию — `QUERY`;
- подключать ИИ-помощников, чтобы получать осмысленные отчеты на человеческом языке;
- расширять подход за пределы финансов: применять аналитику к спорту, хобби, коллекциям и даже отслеживанию цен.

SKLADCHIK.ORG

6.1. Фундамент анализа: ваш личный дата-центр в Google Таблицах

Первый шаг к анализу — прекратить хаос. Когда деньги хранятся на разных картах, счетах и в электронных кошельках, увидеть общую картину невозможно. Вы знаете, сколько потратили в одном банке, но не помните, сколько сняли наличными в другом.

Наша цель — собрать все финансовые операции в одном месте. Google Таблицы — идеальный инструмент для этого. Они бесплатны, доступны с любого устройства и, что главное, дают вам полный контроль над данными. В отличие от банковских приложений, которые показывают готовые, но ограниченные отчеты, таблица становится вашей личной лабораторией для анализа.



	A	B	C	D	E
1	**Счёт**	**Остаток**			
2	Tinkoff Black	120 880,00 ₽			
3	Tinkoff Platinum	8 000,00 ₽			
4	Альфа-Карта	5 550,00 ₽			
5	Инвестиционный счет	30 000,00 ₽			
6	Карта для жизни ВТБ	3 650,00 ₽			
7	Накопительный счет	2 000,00 ₽			
8	Наличные	14 300,00 ₽			
9	СберКарта	10 980,00 ₽			
10					

Рис. 6.1. ***Финансовые источники как отдельные островки: каждая карта — свой кусочек картины***

Мы создадим в них **личный дата-центр**: три вкладки — **Доходы**, **Расходы** и **Переводы**.

Главный принцип — все должно быть в таблице. В отличие от банковских приложений с их готовыми, но ограниченными отчетами, Google Таблица дает полный контроль и становится личной лабораторией для экспериментов с формулами и категориями.

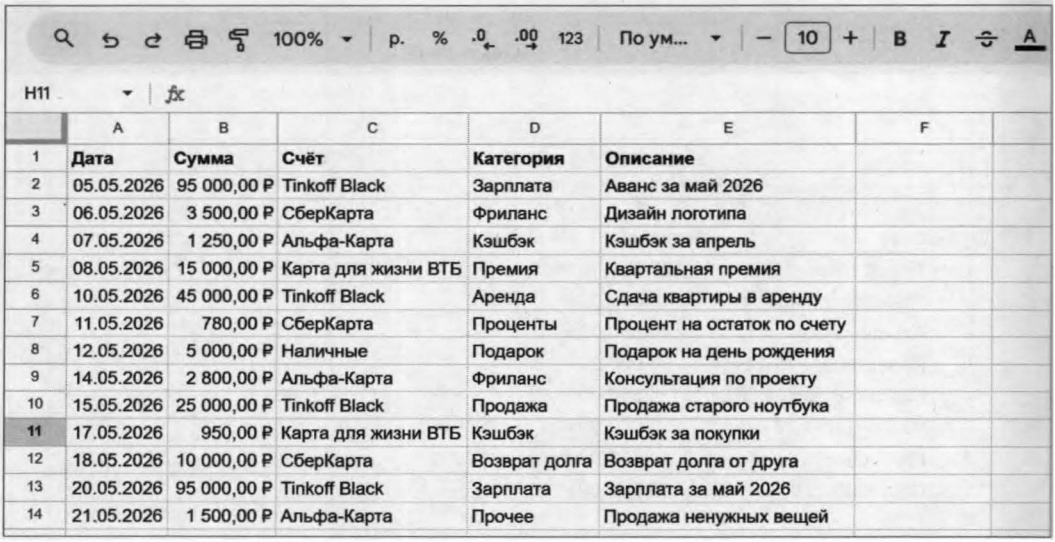
Для начала создаем структуру — открываем новую Google Таблицу и создаем три листа с именами:

- `Доходы`,
- `Расходы`,
- `Переводы`.

В каждом листе — одинаковый набор полей, соответствующий логике движения денег.

Доходы

| Дата | Сумма | Счет | Категория | Описание |



	A	B	C	D	E	F
1	**Дата**	**Сумма**	**Счёт**	**Категория**	**Описание**	
2	05.05.2026	95 000,00 ₽	Tinkoff Black	Зарплата	Аванс за май 2026	
3	06.05.2026	3 500,00 ₽	СберКарта	Фриланс	Дизайн логотипа	
4	07.05.2026	1 250,00 ₽	Альфа-Карта	Кэшбэк	Кэшбэк за апрель	
5	08.05.2026	15 000,00 ₽	Карта для жизни ВТБ	Премия	Квартальная премия	
6	10.05.2026	45 000,00 ₽	Tinkoff Black	Аренда	Сдача квартиры в аренду	
7	11.05.2026	780,00 ₽	СберКарта	Проценты	Процент на остаток по счету	
8	12.05.2026	5 000,00 ₽	Наличные	Подарок	Подарок на день рождения	
9	14.05.2026	2 800,00 ₽	Альфа-Карта	Фриланс	Консультация по проекту	
10	15.05.2026	25 000,00 ₽	Tinkoff Black	Продажа	Продажа старого ноутбука	
11	17.05.2026	950,00 ₽	Карта для жизни ВТБ	Кэшбэк	Кэшбэк за покупки	
12	18.05.2026	10 000,00 ₽	СберКарта	Возврат долга	Возврат долга от друга	
13	20.05.2026	95 000,00 ₽	Tinkoff Black	Зарплата	Зарплата за май 2026	
14	21.05.2026	1 500,00 ₽	Альфа-Карта	Прочее	Продажа ненужных вещей	

Рис. 6.2. ***Пример листа "Доходы" с тестовыми данными***

Расходы

| Дата | Сумма | Счет | Категория | Описание |



	A	B	C	D	E
1	**Дата**	**Сумма**	**Счёт**	**Категория**	**Описание**
2	01.05.2026	40 000,00 ₽	Tinkoff Black	Аренда жилья	Оплата аренды квартиры
3	02.05.2026	7 500,00 ₽	СберКарта	Продукты	Закупка продуктов на неделю
4	03.05.2026	3 500,00 ₽	Альфа-Карта	Рестораны	Ужин с друзьями
5	04.05.2026	1 800,00 ₽	Наличные	Транспорт	Проездной на месяц
6	05.05.2026	15 000,00 ₽	Tinkoff Platinum	Техника	Покупка нового телефона
7	07.05.2026	2 500,00 ₽	Карта для жизни ВТБ	Спорт	Абонемент в спортзал
8	09.05.2026	4 000,00 ₽	Tinkoff Black	Развлечения	Билеты на концерт
9	10.05.2026	6 200,00 ₽	СберКарта	ЖКХ	Оплата коммунальных услуг
10	12.05.2026	8 000,00 ₽	Альфа-Карта	Одежда	Покупка весенней куртки
11	13.05.2026	900,00 ₽	Наличные	Бытовые товары	Хозяйственные товары

Рис. 6.3. ***Пример листа "Расходы"***

Переводы

| Дата | Сумма | Счет (откуда) | Счет (куда) | Описание |



	A	B	C	D	E
1	**Дата**	**Сумма**	**Счёт (откуда)**	**Счёт (куда)**	**Описание**
2	02.05.2026	20 000,00 ₽	Tinkoff Black	Накопительный счет	Ежемесячные накопления
3	04.05.2026	5 000,00 ₽	СберКарта	Наличные	Снятие наличных
4	06.05.2026	10 000,00 ₽	Альфа-Карта	Карта для жизни ВТБ	Перевод жене/мужу
5	08.05.2026	15 000,00 ₽	Tinkoff Black	Tinkoff Platinum	Частичное погашение кредита
6	11.05.2026	3 000,00 ₽	Наличные	СберКарта	Внесение наличных на карту
7	13.05.2026	25 000,00 ₽	Накопительный счет	Tinkoff Black	Перевод на текущие расходы
8	14.05.2026	7 000,00 ₽	Карта для жизни ВТБ	Альфа-Карта	Возврат денег другу
9	19.05.2026	8 000,00 ₽	Tinkoff Black	Наличные	Снятие наличных в банкомате
10	21.05.2026	40 000,00 ₽	Tinkoff Black	Tinkoff Platinum	Полное погашение кредитной карты
11	23.05.2026	5 000,00 ₽	Альфа-Карта	СберКарта	Перевод между своими счетами
12	24.05.2026	10 000,00 ₽	Карта для жизни ВТБ	Накопительный счет	Пополнение "подушки безопасности"
13	26.05.2026	2 500,00 ₽	Наличные	Tinkoff Black	Внесение через банкомат
14	27.05.2026	30 000,00 ₽	Tinkoff Black	Инвестиционный счет	Перевод на брокерский счет
15	29.05.2026	18 000,00 ₽	Накопительный счет	Альфа-Карта	Перевод для крупной покупки

Рис. 6.4. ***Пример листа "Переводы"***

Каждая строка — это событие. Неважно, было ли оно 5 минут назад или 5 месяцев — таблица станет архивом вашей финансовой истории.

Главное — **все записывать одинаково**: дата, сумма, источник и краткое пояснение.

Практикум: магия функции QUERY

Теперь начинается самое интересное. Мы хотим, чтобы таблица *думала вместе с нами*. Чтобы на одном экране мы могли видеть **все операции сразу** — и доходы, и расходы, и переводы.

Обычно в Excel или Google Таблицы для этого используют сводные таблицы. Но есть способ лучше — функция `QUERY`. Она работает как мини-база данных прямо внутри таблицы.

Ее идея проста: мы выбираем нужные диапазоны, объединяем их в один виртуальный лист и делаем SQL-запрос вроде `select`, `group by` или `order by`.

Задача: собрать единую картину

У нас три листа, и на каждом — свой тип данных. Доходы положительные, расходы отрицательные, переводы — нейтральные, потому что они просто перемещают средства внутри системы.

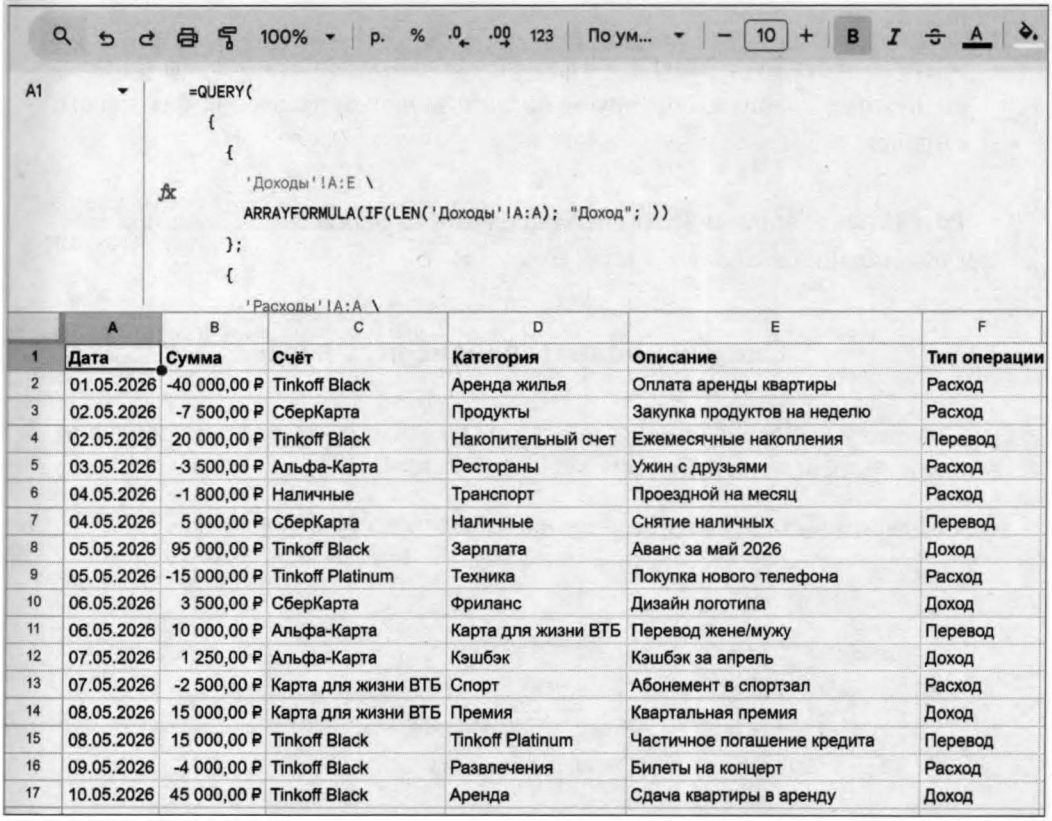
Мы хотим видеть **все движения денег в одном окне**, где у каждой строки будет тип операции: Доход, Расход или Перевод.

Решение: объединяем листы

Создаем новый лист с именем `Все операции` и вставляем формулу:

```
=QUERY(
  {
    {
      'Доходы'!A:E \
      ARRAYFORMULA(IF(LEN('Доходы'!A:A); "Доход"; ))
    };
    {
      'Расходы'!A:A \
```

```
      ARRAYFORMULA(IF(LEN('Расходы'!A:A); 'Расходы'!B:B*-1; )) \
      'Расходы'!C:E \
      ARRAYFORMULA(IF(LEN('Расходы'!A:A); "Расход"; ))
    };
    {
      'Переводы'!A:E \
      ARRAYFORMULA(IF(LEN('Переводы'!A:A); "Перевод"; ))
    }
  };
  "select Col1, Col2, Col3, Col4, Col5, Col6
   where Col1 is not null
   order by Col1 asc
   label Col6 'Тип операции'"
)
```



	A	B	C	D	E	F
1	**Дата**	**Сумма**	**Счёт**	**Категория**	**Описание**	**Тип операции**
2	01.05.2026	-40 000,00 ₽	Tinkoff Black	Аренда жилья	Оплата аренды квартиры	Расход
3	02.05.2026	-7 500,00 ₽	СберКарта	Продукты	Закупка продуктов на неделю	Расход
4	02.05.2026	20 000,00 ₽	Tinkoff Black	Накопительный счет	Ежемесячные накопления	Перевод
5	03.05.2026	-3 500,00 ₽	Альфа-Карта	Рестораны	Ужин с друзьями	Расход
6	04.05.2026	-1 800,00 ₽	Наличные	Транспорт	Проездной на месяц	Расход
7	04.05.2026	5 000,00 ₽	СберКарта	Наличные	Снятие наличных	Перевод
8	05.05.2026	95 000,00 ₽	Tinkoff Black	Зарплата	Аванс за май 2026	Доход
9	05.05.2026	-15 000,00 ₽	Tinkoff Platinum	Техника	Покупка нового телефона	Расход
10	06.05.2026	3 500,00 ₽	СберКарта	Фриланс	Дизайн логотипа	Доход
11	06.05.2026	10 000,00 ₽	Альфа-Карта	Карта для жизни ВТБ	Перевод жене/мужу	Перевод
12	07.05.2026	1 250,00 ₽	Альфа-Карта	Кэшбэк	Кэшбэк за апрель	Доход
13	07.05.2026	-2 500,00 ₽	Карта для жизни ВТБ	Спорт	Абонемент в спортзал	Расход
14	08.05.2026	15 000,00 ₽	Карта для жизни ВТБ	Премия	Квартальная премия	Доход
15	08.05.2026	15 000,00 ₽	Tinkoff Black	Tinkoff Platinum	Частичное погашение кредита	Перевод
16	09.05.2026	-4 000,00 ₽	Tinkoff Black	Развлечения	Билеты на концерт	Расход
17	10.05.2026	45 000,00 ₽	Tinkoff Black	Аренда	Сдача квартиры в аренду	Доход

Рис. 6.5. ***Визуализация объединения трех вкладок в одну таблицу с помощью QUERY***

Разбираем формулу шаг за шагом

- **Кавычки и скобки** { ... } — это способ объединить несколько диапазонов в один массив.
  Мы буквально "склеиваем" три таблицы в одну.
- **ARRAYFORMULA** позволяет формуле обрабатываться сразу для всех строк.
  Мы используем ее, чтобы добавить в конец каждой строки дополнительное поле — тип операции (Доход, Расход, Перевод).
- **IF(LEN(...); ...)** — фильтрует пустые строки, чтобы не тянуть мусор из незаполненных областей.
- **'Расходы'!B:B * -1** — превращаем расходы в отрицательные значения. Это ключевой момент! Теперь при суммировании у нас будет корректный баланс.
- **select ... order by ... label ...** — финальная часть запроса, где мы выбираем нужные колонки, сортируем по дате и задаем заголовок для нового столбца.

Результат — динамическая таблица, которая обновляется сама при каждом изменении в исходных листах (рис. 6.5).

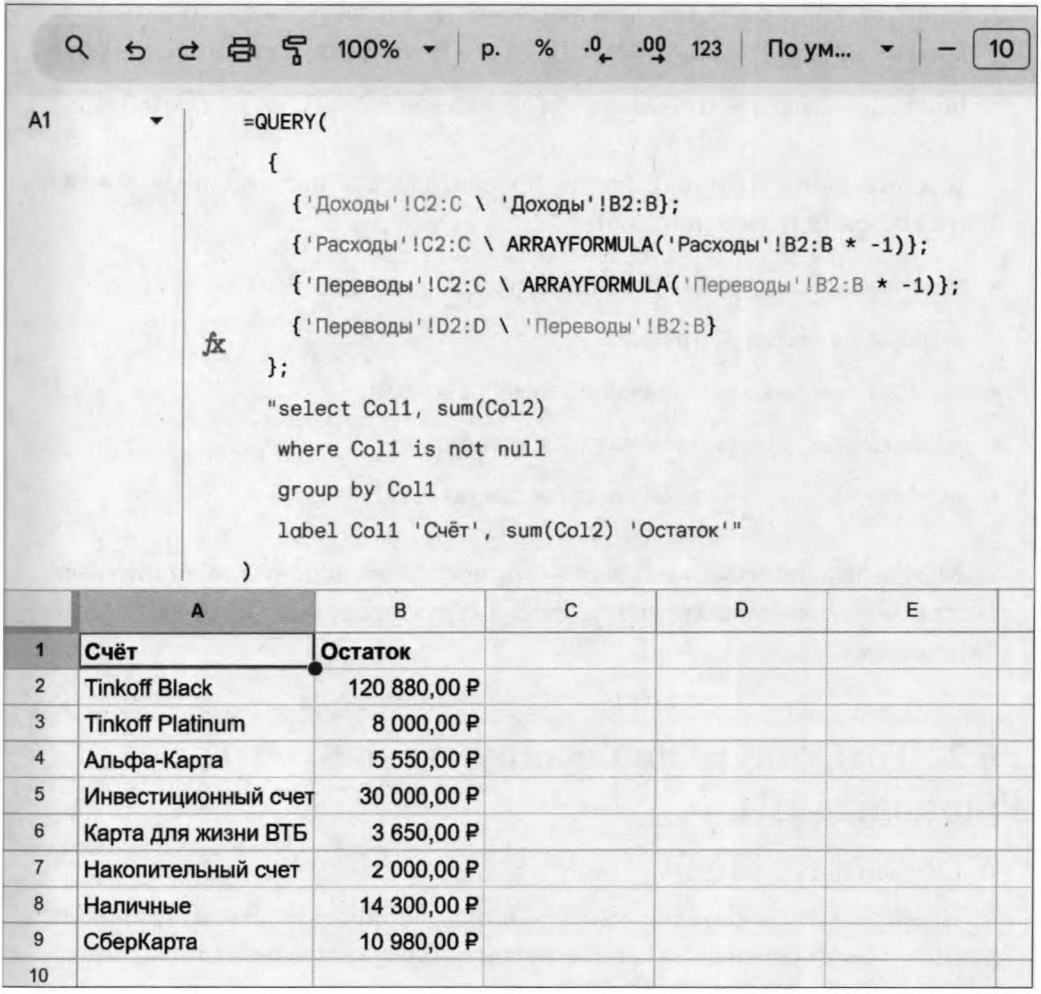
Следующий шаг: баланс всех карт

Сколько у нас сейчас на каждой карте? Сколько денег реально доступно, если учесть переводы и траты?

Создаем новую вкладку `Баланс` и вставляем вторую формулу:

```
=QUERY(
  {
    {'Доходы'!C2:C \ 'Доходы'!B2:B};
    {'Расходы'!C2:C \ ARRAYFORMULA('Расходы'!B2:B * -1)};
    {'Переводы'!C2:C \ ARRAYFORMULA('Переводы'!B2:B * -1)};
    {'Переводы'!D2:D \ 'Переводы'!B2:B}
  };
  "select Col1, sum(Col2)
```

```
    where Col1 is not null
    group by Col1
    label Col1 'Счет', sum(Col2) 'Остаток'"
)
```



	A	B	C	D	E
1	**Счёт**	**Остаток**			
2	Tinkoff Black	120 880,00 ₽			
3	Tinkoff Platinum	8 000,00 ₽			
4	Альфа-Карта	5 550,00 ₽			
5	Инвестиционный счет	30 000,00 ₽			
6	Карта для жизни ВТБ	3 650,00 ₽			
7	Накопительный счет	2 000,00 ₽			
8	Наличные	14 300,00 ₽			
9	СберКарта	10 980,00 ₽			
10					

Рис. 6.6. ***Таблица "Баланс": сводка по всем счетам и картам***

Как это работает:

1. **Мы объединяем все источники.**
   Каждый кусочек формулы { ... } берет данные из одного листа:

» доходы — положительные,

» расходы — отрицательные,

» переводы — дважды (откуда минус, куда плюс).

2. **sum(Col2)** — складывает все движения по каждому счету. То есть формула не просто суммирует строки — она *группирует* их по имени счета.
3. **label ...** — задает читабельные заголовки колонок: "Счет" и "Остаток".

В итоге наша Google Таблица превратилась в полноценный **мини-бухгалтерский центр (рис. 6.6)**:

- вкладка `Доходы` хранит поступления,
- вкладка `Расходы` — траты,
- вкладка `Переводы` — движение между счетами,
- вкладка `Все операции` — единая история,
- вкладка `Баланс` — мгновенная сводка по остаткам.

Мы сделали не просто таблицу — мы построили **основу для аналитики**. Теперь данные не хранятся вразнобой, а структурированы и готовы к автоматическому анализу.

6.2. Практикум: анализируем свои траты с помощью ИИ

В предыдущем разделе мы научились собирать и структурировать данные о своих финансах в единой таблице. Теперь наша база готова — пора добавить в процесс немного магии.

Мы не будем вручную строить десятки графиков и придумывать формулы. Вместо этого мы научимся делегировать часть анализа нейросети — ChatGPT, Gemini или DeepSeek AI. Эти инструменты уже умеют работать с текстовыми таблицами, находить закономерности и давать рекомендации на человеческом языке.

Мы превращаем ИИ в **персонального финансового консультанта**, который знает нашу ситуацию не по рекламе банков, а по реальным цифрам ваших доходов и расходов.

Идея: пусть ИИ анализирует, а мы принимаем решения

Когда мы работаем с личными финансами, нас часто подводит одно: слишком много данных, слишком мало времени. Даже если таблица идеально структурирована, можно легко увязнуть в столбцах и сводках. Но если подать эти данные ИИ в виде простого текста, он сможет:

- увидеть, какие категории трат растут;
- выделить несбалансированные расходы;
- подсказать, где есть потенциал для сбережений;
- помочь выстроить финансовые цели.

Фактически мы используем ту же таблицу, но теперь не как инструмент учета, а как базу знаний для интеллектуального помощника.

Шаг 1. Подготовка данных

Нам не нужно выгружать отчеты, писать скрипты или устанавливать плагины.

Алгоритм такой:

1. В Google Таблице открываем лист **Все операции** или **Расходы**.
2. Выделяем диапазон с заголовками и строками (примерно месяц).
3. Копируем (`Ctrl+C`) и вставляем (`Ctrl+V`) в чат с ChatGPT или Gemini.
4. Добавляем короткое пояснение — кто мы и что хотим получить.

> **Важно**: не используем скриншоты, только **текст**. Тогда ИИ сможет анализировать каждую строку.

Шаг 2. Общий финансовый чекап

Начнем с простого — попросим ИИ сделать обзор наших доходов и расходов.

Мы подаем данные из таблицы (можно за месяц) и используем следующий промпт:

Промпт:

> "Действуй как опытный финансовый консультант. Помоги составить отчет для мужчины 30 лет, ведущего личный бюджет. Вот таблица его доходов и расходов за май 2026 года. Определи структуру доходов, топ-5 категорий расходов, рассчитай чистый денежный поток, норму сбережений и предложи конкретные рекомендации по оптимизации бюджета".

ИИ не просто суммирует числа — он видит логику, распределение и поведение. Пример того, как может выглядеть ответ:

Отчет ИИ

Основные источники дохода:

- Зарплата (190 000 ₽, 63 %).
- Фриланс и проекты (24 000 ₽, 8 %).
- Аренда и дивиденды (49 200 ₽, 16 %).
- Прочие поступления (18 000 ₽, 6 %).

Топ-5 категорий расходов:

1. Аренда жилья — 40 000 ₽.
2. Путешествия — 25 000 ₽.
3. Продукты — 16 000 ₽.
4. Крупные покупки — 10 000 ₽.
5. Техника — 15 000 ₽.

Чистый денежный поток: + 75 000 ₽

Норма сбережений: 27 %

Рекомендации:

- Расходы на рестораны и развлечения составляют 10 % бюджета. Снижение этой доли на 5 % высвободит ≈ 5 000 ₽ в месяц для инвестиций.

- Оптимизируйте платежи по кредиту: часть свободных средств можно направить на досрочное погашение.

Такой отчет можно вставить прямо под таблицу в Google Документ — он превращается в ежемесячную сводку.

Сильная сторона ИИ в том, что он не требует сложных формул. Мы просто говорим человеческим языком: "Посмотри, что здесь происходит" и получаем ответ, понятный даже без диаграмм.

Шаг 3. Расследование аномалий

Теперь предположим, что в январе или мае расходы внезапно выросли, и мы не можем понять почему. Такое часто случается: неожиданные траты, праздники, ремонт. Вместо того чтобы вручную выискивать лишние нули, мы снова обращаемся к ИИ.

Промпт:

> "Действуй как опытный финансовый консультант. Важно разобраться, почему в мае 2025 года произошел существенный перерасход.
>
> Вот таблица расходов и переводов за этот месяц. Найди аномальные категории, сравни их с предыдущими месяцами (если есть данные) и объясни вероятные причины. Приведи примеры конкретных транзакций, которые повлияли на результат".

ИИ может вернуть не просто текст, а целое расследование:

Анализ аномалий:

- Общие расходы выросли на 35 % по сравнению с апрелем.
- Главная причина — категория *Путешествия* (25 000 ₽ — авиабилеты).
- Дополнительный рост связан с покупкой нового телефона (категория *Техника*, 15 000 ₽).
- Эти две операции составляют ≈ 45 % всех майских расходов.
- Без учета этих событий расходы находятся в норме.

Рекомендация:

Можно пометить эти траты как разовые, чтобы не искажали средние значения в будущем анализе.

Преимущество подхода в том, что ИИ умеет работать с контекстом. Он не просто видит числа, а понимает смысл: *ремонт авто ≠ повседневные траты*.

Это особенно удобно, если мы ведем таблицу несколько месяцев подряд и хотим отслеживать тренды, а не отдельные события.

Шаг 4. Глубокое погружение в категорию

Иногда нас интересует не общая картина, а конкретный вопрос: "Я точно не трачу слишком много на транспорт?" или "Почему расходы на еду растут каждый месяц?"

Вместо того чтобы вручную фильтровать строки, мы используем тот же подход: копируем фрагмент таблицы и уточняем вопрос.

Промпт:

> "Действуй как опытный финансовый консультант. Помоги проанализировать категорию **Транспорт**.
>
> Вот таблица расходов за май 2025 года. Разбей траты на подкатегории (такси, общественный транспорт, бензин, парковка), сравни их между собой и оцени, как они влияют на общий бюджет. Подскажи, где можно оптимизировать расходы без потери комфорта".

ИИ возвращает результат в удобной форме, понятной даже без таблицы:

Анализ категории "Транспорт"

- Общие траты: 7 300 ₽ (≈ 6 % от расходов).
- Структура:
  - » Бензин и мойка — 5 500 ₽ (75 %).
  - » Проездной — 1 800 ₽ (25 %).

Расходы стабильны, но зависят от цены топлива.

Возможности оптимизации:

- Попробовать совместные поездки.
- Проверить выгодность годового проездного — снижает ежемесячные затраты.

Такой анализ можно повторять для любой категории — Продукты, Развлечения, Одежда или Путешествия.

Каждый раз ИИ адаптирует рекомендации под контекст и сумму. А если в таблице накоплены данные за несколько месяцев, мы можем попросить:

- "Сравни расходы по категории Продукты за последние 3 месяца и покажи тренд".

ИИ выдаст динамику, комментарии и выводы вроде:

- "Расходы растут на 12 % в месяц из-за увеличения заказов готовой еды. Возможное решение — переход на закупки раз в неделю".

Итого: ИИ как надстройка над таблицей

После нескольких таких сессий мы замечаем, что **анализ перестает быть скучной бухгалтерией**.

ИИ становится мостом между цифрами и действиями. Мы перестаем просто учитывать деньги и начинаем управлять ими.

Схема: Google Таблица => ИИ-отчет => Решение

Главная идея — **не заменять таблицу**, а дополнить ее. Google Таблицы дают прозрачность и контроль, а нейросеть — интерпретацию и инсайты. Вместе они превращают обычный учет в систему принятия решений.

Теперь у нас есть три уровня анализа:

1. **Таблица** — структура и точность.
2. **Формулы** — автоматическое обновление и баланс.
3. **ИИ-анализ** — смысл и выводы.

6.3. За пределами финансов: другие идеи для персональной аналитики

Аналитика — это не только про деньги. Это про осознанность: видеть закономерности в том, что мы делаем, и принимать решения на основе фактов, а не ощущений.

В этом разделе мы шагнем **за пределы финансов** и посмотрим, как тот же подход помогает нам управлять хобби, привычками и даже покупками. Мы создадим несколько мини-проектов, каждый из которых расширит ваши аналитические возможности.

Личный музей: учитываем коллекции и увлечения

Представим, что у нас есть коллекция — виниловых пластинок, настольных игр, инструментов или даже редких сортов кофе. Обычно такие коллекции растут стихийно, а их "инвентаризация" существует только в памяти владельца. Мы же сделаем это системно — в Google Таблицах.

Создадим новую таблицу с листом **"Коллекция"** и добавим следующие столбцы:

Альбом	Исполнитель	Год выпуска	Год покупки	Стоимость	Примечание

Теперь каждая строка — это отдельный предмет коллекции. Через пару десятков строк таблица уже превращается в каталог, который можно фильтровать, сортировать и анализировать.

Чтобы увидеть общие показатели, добавляем две простые формулы:

- **=COUNT(A2:A)** — считает количество позиций;
- **=SUM(E2:E)** — суммирует затраты.

И вот мы уже видим, сколько у нас экземпляров и сколько денег в них вложено.

> **Совет:** если в коллекции есть редкие экземпляры, добавьте поле **Рыночная стоимость** и разницу между ней и ценой покупки. Так можно увидеть потенциальную стоимость коллекции на вторичном рынке.

Следующий шаг — визуализация. Google Таблицы позволяют строить **гистограммы по годам выпуска или по исполнителям**. За пару кликов можно увидеть, какой жанр или период преобладает в вашей коллекции.

Такая аналитика помогает не только поддерживать порядок, но и осознанно развивать хобби: видеть, что мы собираем, чего не хватает и где фокус нашего интереса.

Аналитика тела: визуализируем спорт и привычки

Те же инструменты легко превращаются в **персональный трекер здоровья и активности**.

Создадим таблицу **"Пробежки"** со столбцами:

| Дата | Дистанция (км) | Время (мин) | Средний темп (мин/км) | Настроение |

Теперь после каждой тренировки мы просто вносим одну строку — не нужно приложение, которое запрашивает доступ к GPS и уведомлениям.

Затем добавляем формулу для среднего темпа:

```
=C2/B2
```

и копируем ее вниз через **ARRAYFORMULA**, чтобы вычисление происходило автоматически.

Теперь самое приятное — строим график. Выделяем столбцы **Дата** и **Дистанция => Вставка => Диаграмма => Линейный график.**

> **Готово**! Мы видим, как растут наши дистанции или скорость со временем.

Можно добавить вторую ось и вывести темп, чтобы видеть взаимосвязь между расстоянием и скоростью.

Если дополнительно добавить столбец **Настроение** или **Погода**, можно увидеть, как внешние факторы влияют на результаты. Это уже **анализ поведения** — когда мы находим причинно-следственные связи в собственных данных.

Когда данные работают сами: отслеживаем цены с помощью Python

Теперь усложним задачу. Представьте, что вы хотите купить ноутбук, но ждете скидку. Каждый день заходить на сайт и проверять цену вручную — утомительно. Решение? Мы автоматизируем это, превратив обычный анализ в **автономный мониторинг**.

Создадим **мини-скрипт на Python**, который сам будет следить за ценой и сообщать, когда она снизится.

Для этого нам понадобятся знакомые библиотеки:

```
import requests
from bs4 import BeautifulSoup
```

Логика работы проста — пошагово:

1. **Скрипт заходит на страницу товара.**
   Мы указываем URL в переменной `url`.
2. **Находит HTML-элемент, где указана цена.**
   Для этого в браузере открываем "Инструменты разработчика" (F12) и смотрим, в каком теге отображается цена, например `<span class="price">`.
3. **Сохраняет цену в текстовый файл.**
   Если файл уже есть — считывает старую цену для сравнения.
4. **Сравнивает новую и старую цены.**
   Если новая меньше — значит, скидка!
5. **Отправляет уведомление в Telegram.**
   Используем бота, чтобы получить сообщение вроде:

   "Цена на ноутбук снизилась с 99 990 до 87 990 ₽!"

Таким образом, мы создаем **автономного аналитического бота**, который делает то, что делали бы мы, но автоматически.

Схема работы скрипта отслеживания цен: сайт => скрипт => сравнение => уведомление.

Преимущество такого подхода — его универсальность. Ту же схему можно адаптировать для мониторинга курса валют, авиабилетов или редких товаров на Авито.

Заключение

По результатам этой главы вы научились создавать не просто таблицы, а целую систему управления личными финансами.

А ещё узнали про полезные формулы и получили навыки организации связанных таблиц. Ну и, конечно же, научились использовать ИИ для достижения своих целей.

Все эти навыки будут полезны и для следующей главы, где мы окунёмся в мир инвестиций.



Глава 7.

Автоматический мониторинг инвестиционного портфеля

Многие частные инвесторы, которые не "кладут все яйца в одну корзину", рано или поздно приходят к тому, что активы находятся по разным брокерам, вкладам, недвижимости и даже монетам. А ведь все они выдают отчёты в своём собственном формате и сведение их всех вместе это отдельная работа. Однако и без этой сводной картины нельзя.

Цель главы разобраться как увидеть общую картину. Для этого будем использовать Google Таблицу, которая станет вашим инвестиционным дашбордом. Это будет не простая таблица, она сможет самостоятельно скачивать актуальные котировки, рассчитывать доходность и делать это всё в режиме реального времени для всех активов.

Пройдем путь от простого учета к созданию активной системы поддержки решений, которая освободит ваше время и даст полный контроль над капиталом. Инструментами нам послужат Google Таблицы и их встроенные функции, такие как GOOGLEFINANCE и IMPORTXML. Когда их возможностей перестанет хватать, мы напишем собственные функции на Google Apps Script. Ну, а для самых сложных задач – запустим на своем компьютере небольшой Python-сервер, который будет "посредником" между таблицей (на этот раз в Excel) и закрытыми данными от брокера.

7.1. Создаем универсальную таблицу для учета всех активов

Начнём с базы – таблицы, которая одинаково подходит для акций, валют, облигаций, недвижимости и даже монет.

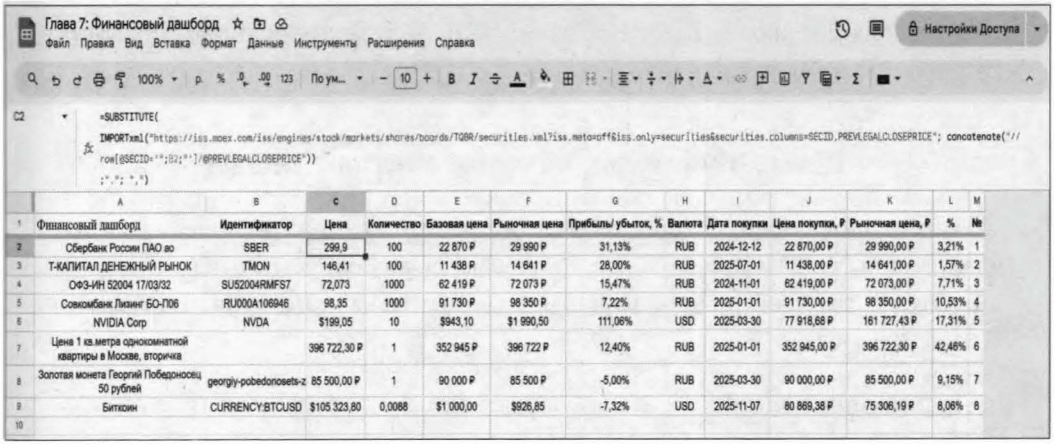
Шаг 1. Структура данных – наш скелет.

Откроем новую Google Таблицу и назовем её, например, **"Финансовый дашборд"**. На этом этапе мы проектируем структуру данных – именно от неё зависит, насколько удобно будет всё автоматизировать позже.

Добавим строки и создадим первые столбцы:

```
| Имя | Идентификатор | Цена | Количество | Базовая цена | Рыночная цена | Прибыль/убыток, % | Валюта | Дата покупки | Цена покупки, ₽ | Рыночная цена, ₽ | % | № |
```

Рисунок ниже показывает базовую структуру таблицы.



Глава 7: Финансовый дашборд

Файл Правка Вид Вставка Формат Данные Инструменты Расширения Справка

Настройки Доступа

C2: =SUBSTITUTE(IMPORTxml("https://iss.moex.com/iss/engines/stock/markets/shares/boards/TQBR/securities.xml?iss.meta=off&iss.only=securities&securities.columns=SECID,PREVLEGALCLOSEPRICE"; concatenate("//row[@SECID='";B2;"']/@PREVLEGALCLOSEPRICE")) ;".";",")

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Финансовый дашборд	Идентификатор	Цена	Количество	Базовая цена	Рыночная цена	Прибыль/ убыток, %	Валюта	Дата покупки	Цена покупки, ₽	Рыночная цена, ₽	%	№
2	Сбербанк России ПАО ао	SBER	299,9	100	22 870 ₽	29 990 ₽	31,13%	RUB	2024-12-12	22 870,00 ₽	29 990,00 ₽	3,21%	1
3	Т-КАПИТАЛ ДЕНЕЖНЫЙ РЫНОК	TMON	146,41	100	11 438 ₽	14 641 ₽	28,00%	RUB	2025-07-01	11 438,00 ₽	14 641,00 ₽	1,57%	2
4	ОФЗ-ИН 52004 17/03/32	SU52004RMFS7	72,073	1000	62 419 ₽	72 073 ₽	15,47%	RUB	2024-11-01	62 419,00 ₽	72 073,00 ₽	7,71%	3
5	Совкомбанк Лизинг БО-П06	RU000A106946	98,35	1000	91 730 ₽	98 350 ₽	7,22%	RUB	2025-01-01	91 730,00 ₽	98 350,00 ₽	10,53%	4
6	NVIDIA Corp	NVDA	$199,05	10	$943,10	$1 990,50	111,06%	USD	2025-03-30	77 918,68 ₽	161 727,43 ₽	17,31%	5
7	Цена 1 кв.метра однокомнатной квартиры в Москве, вторичка		396 722,30 ₽	1	352 945 ₽	396 722 ₽	12,40%	RUB	2025-01-01	352 945,00 ₽	396 722,30 ₽	42,46%	6
8	Золотая монета Георгий Победоносец 50 рублей	georgiy-pobedonosets-z	85 500,00 ₽	1	90 000 ₽	85 500 ₽	-5,00%	RUB	2025-03-30	90 000,00 ₽	85 500,00 ₽	9,15%	7
9	Биткоин	CURRENCY:BTCUSD	$105 323,80	0,0088	$1 000,00	$926,85	-7,32%	USD	2025-11-07	80 869,38 ₽	75 306,19 ₽	8,06%	8
10													

Рис. 7.1. ***Структура универсальной таблицы для учёта активов***

Теперь разберём, зачем нужен каждый столбец.

- **Имя** – простое человеко-читаемое название. Оно помогает не путаться между активами. Может быть любое: "Сбербанк ао", "NVIDIA Corp", "Однушка в Москве" или "Золотая монета".

  Этот столбец не участвует в вычислениях, но делает таблицу понятной.

- **Идентификатор** – сердце автоматизации. Для акций это может быть тикер (SBER, NVDA), для облигаций – ISIN (SU52004RMFS7), для криптовалют – пара в формате CURRENCY:BTCUSD, а для недвижимости или предметов коллекции – просто ключевая часть URL, по которой позже можно будет собрать данные с сайта.

- **Количество** – универсальный параметр. Для акций – количество лотов, для недвижимости – метры, для монет – штуки, для фондов – паи.
- **Базовая цена** – сколько вы заплатили за актив, включая комиссии. Это важно, чтобы корректно посчитать доходность, особенно если покупка была в валюте.
- **Валюта** – обязательное поле. Валюты в таблице могут быть смешанными: рубли, доллары, евро, юани. От этого зависит, будет ли таблица пересчитывать курсы.
- **Дата покупки** – помогает получать исторические курсы валют и рассчитывать реальную доходность в рублях, мы ведь в России, а не просто в долларах или евро.

Эти поля создают основу. Всё остальное – автоматические вычисления, которые мы добавим дальше.

Шаг 2. Формулы, которые считают за нас.

Теперь наша цель – чтобы таблица не просто хранила данные, а сама их анализировала. Мы добавим две ключевые формулы: расчёт цены покупки в рублях и текущей рыночной цены с учётом валютного курса по данным Google Finance.

1. Цена покупки, ₽

Формула, которая пересчитает базовую цену в рубли по курсу на дату покупки. Если актив в рублях, просто подставляем базовую цену. Если актив в валюте – получаем курс с помощью функции `GOOGLEFINANCE`.

```
=IF(H2="RUB"; E2; INDEX(GOOGLEFINANCE("CURRENCY:" & H2 & "RUB";
"price"; I2); 2; 2) * E2)
```

Разберём подробно:

- `IF(H2="RUB"; E2; ...)` – проверяем, рублёвый ли актив. Если да, возвращаем значение без пересчёта.
- Если нет, используем `GOOGLEFINANCE`, чтобы получить исторический курс валюты на дату покупки (`I2` – это столбец с датой).

- `INDEX(...; 2; 2)` – стандартный способ вытащить нужное значение из таблицы, которую возвращает `GOOGLEFINANCE`.

После применения этой формулы у нас появляется корректная стоимость покупки в рублях – даже для иностранных акций или криптовалют.

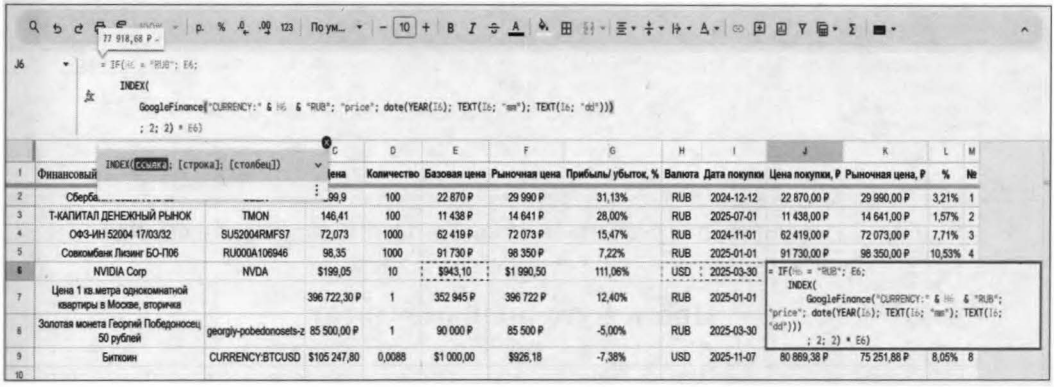


Рис. 7.2. ***Пример формулы для расчета рублевой цены покупки***

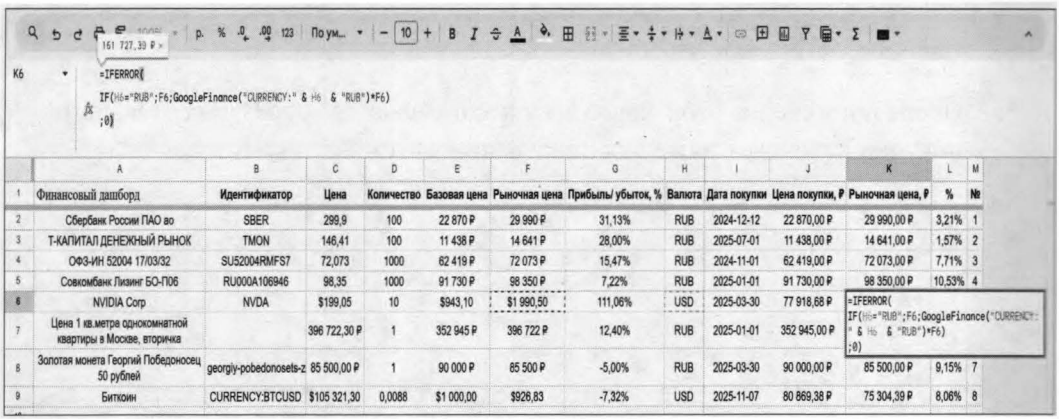
2. Рыночная цена, ₽

Следующая формула делает то же самое, но уже с текущим курсом валюты – чтобы посчитать актуальную рыночную стоимость.

```
=IFERROR(IF(H2="RUB"; F2; GOOGLEFINANCE("CURRENCY:" & H2 & "RUB")
* F2); 0)
```

- `GOOGLEFINANCE("CURRENCY:" & H2 & "RUB")` берёт текущий курс валюты.
- `IFERROR(...; 0)` защищает нас от ошибок, если Google временно не может вернуть данные (например при неправильном тикере).
- Если актив в рублях – просто используем поле **Рыночная цена** без пересчета.

Теперь таблица автоматически обновляет значения каждый раз, когда вы её открываете – без единого макроса. Это превращает статический список активов в живую финансовую панель.



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Финансовый дашборд	Идентификатор	Цена	Количество	Базовая цена	Рыночная цена	Прибыль/ убыток, %	Валюта	Дата покупки	Цена покупки, ₽	Рыночная цена, ₽	%	№
2	Сбербанк России ПАО ао	SBER	299,9	100	22 870 ₽	29 990 ₽	31,13%	RUB	2024-12-12	22 870,00 ₽	29 990,00 ₽	3,21%	1
3	Т-КАПИТАЛ ДЕНЕЖНЫЙ РЫНОК	TMON	146,41	100	11 438 ₽	14 641 ₽	28,00%	RUB	2025-07-01	11 438,00 ₽	14 641,00 ₽	1,57%	2
4	ОФЗ-ИН 52004 17/03/32	SU52004RMFS7	72,073	1000	62 419 ₽	72 073 ₽	15,47%	RUB	2024-11-01	62 419,00 ₽	72 073,00 ₽	7,71%	3
5	Совкомбанк Лизинг БО-П06	RU000A106946	98,35	1000	91 730 ₽	98 350 ₽	7,22%	RUB	2025-01-01	91 730,00 ₽	98 350,00 ₽	10,53%	4
6	NVIDIA Corp	NVDA	$199,05	10	$943,10	$1 990,50	111,06%	USD	2025-03-30	77 918,68 ₽	=IFERROR(		
7	Цена 1 кв.метра однокомнатной квартиры в Москве, вторичка		396 722,30 ₽	1	352 945 ₽	396 722 ₽	12,40%	RUB	2025-01-01	352 945,00 ₽	IF(H6="RUB";F6;GoogleFinance("CURRENCY:" & H6 & "RUB")*F6) ;0)		
8	Золотая монета Георгий Победоносец 50 рублей	georgiy-pobedonosets-z	85 500,00 ₽	1	90 000 ₽	85 500 ₽	-5,00%	RUB	2025-03-30	90 000,00 ₽	85 500,00 ₽	9,15%	7
9	Биткоин	CURRENCY:BTCUSD	$105 321,30	0,0088	$1 000,00	$926,83	-7,32%	USD	2025-11-07	80 869,38 ₽	75 304,39 ₽	8,06%	8

Рис. 7.3. ***Формула для расчета текущей рублевой стоимости актива***

Промежуточный результат

На этом этапе у нас готов функциональный каркас. В него можно вносить данные вручную – таблица сама посчитает всё остальное.

Мы уже видим:

- сколько реально стоит каждый актив в рублях;
- какова общая сумма портфеля;
- где и какая доходность.

7.2. Уровень 1: автоматический сбор котировок с помощью встроенных формул

Здесь мы "оживляем" столбец "Рыночная цена" и превращаем нашу таблицу в постоянно обновляющийся источник. Теперь подключим автоматические источники данных, чтобы таблица сама обновляла цены – без копирования с сайтов. То есть опираемся на три слоя инструментов:

- GOOGLEFINANCE – самый простой вариант,
- IMPORTXML – гибкий парсер для Мосбиржи и сайтов,
- кастомные функции на Google Apps Script для JSON и сложных случаев.

GOOGLEFINANCE – самый простой способ для мировых рынков

Начнём с самой лёгкой и очевидной опции. Для многих зарубежных акций и валют Google уже умеет отдавать котировки "из коробки".

Примеры (мы вставляем формулы прямо в столбец "Цена"):

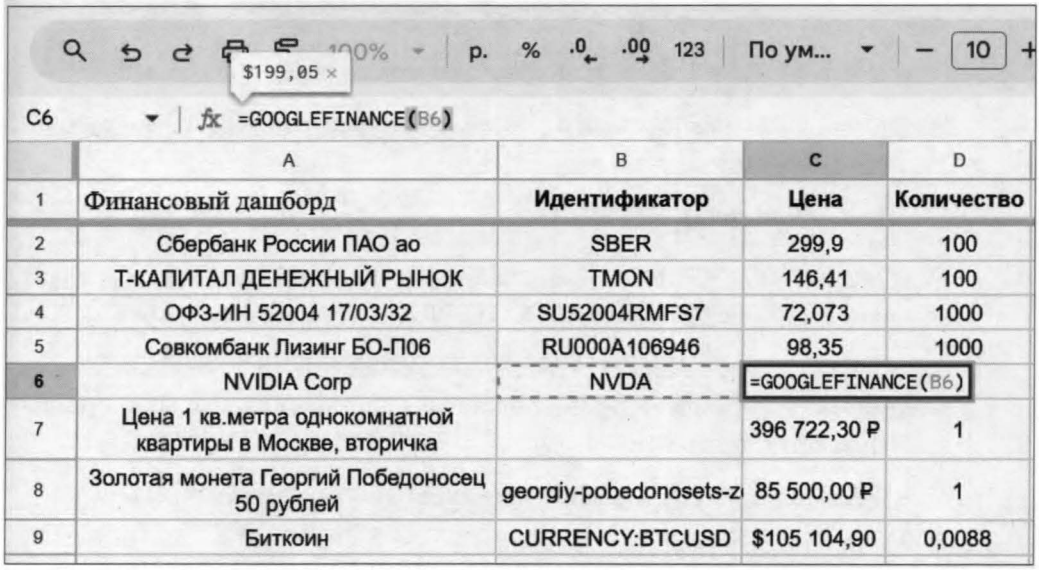
- NVIDIA: `=GOOGLEFINANCE("NVDA")` – вернёт текущую цену акции NVDA.
- Биткоин: `=GOOGLEFINANCE("CURRENCY:BTCUSD")` – вернёт цену BTC в долларах.

Плюсы:

- Простота – в одну строку вставляем формулы и всё работает.
- Надёжность для популярных западных тикеров и основных валют.
- Автообновление при открытии таблицы.

Минусы:

- Не работает для российских компаний и облигаций.
- Не поддерживает многие биржевые инструменты (облигации, биржевые фонды некоторых бирж).
- Иногда данные с задержкой.



	A	B	C	D
1	Финансовый дашборд	Идентификатор	Цена	Количество
2	Сбербанк России ПАО ао	SBER	299,9	100
3	Т-КАПИТАЛ ДЕНЕЖНЫЙ РЫНОК	TMON	146,41	100
4	ОФЗ-ИН 52004 17/03/32	SU52004RMFS7	72,073	1000
5	Совкомбанк Лизинг БО-П06	RU000A106946	98,35	1000
6	NVIDIA Corp	NVDA	=GOOGLEFINANCE(B6)	
7	Цена 1 кв.метра однокомнатной квартиры в Москве, вторичка		396 722,30 ₽	1
8	Золотая монета Георгий Победоносец 50 рублей	georgiy-pobedonosets-z	85 500,00 ₽	1
9	Биткоин	CURRENCY:BTCUSD	$105 104,90	0,0088

Рис. 7.4. Схема: как GOOGLEFINANCE наполняет столбец "Рыночная цена"

Итак, для NVDA и других "американцев" GOOGLEFINANCE – наш стартовый инструмент: быстро и надёжно. Но когда мы переходим на Мосбиржу – его нужно дополнить.

IMPORTXML – решение для Московской биржи

Когда GOOGLEFINANCE бессилен, используем IMPORTXML – и это действительно похоже на маленького робота, который заходит на страницу или в XML-API и вытаскивает конкретную строчку.

IMPORTXML берёт URL и XPath – и возвращает кусок структуры страницы/документа. В случае Мосбиржи у нас есть удобный публичный API в формате XML: `https://iss.moex.com/iss/...securities.xml`. Это не "парсинг HTML с главной страницы", а обращение к официальному интерфейсу биржи в машиночитаемом виде.

Формула выглядит так:

```
=SUBSTITUTE(
IMPORTxml("https://iss.moex.com/iss/engines/stock/markets/
shares/boards/TQBR/securities.xml?iss.meta=off&iss.
only=securities&securities.columns=SECID,PREVLEGALCLOSEPRICE";
concatenate("//row[@SECID='";B2;"']/@PREVLEGALCLOSEPRICE"))
 ;"."; ",")
```

Если эта формула кажется громоздкой, ее можно разбить на части.

Например, URL можно хранить в отдельной ячейке, а XPath – в другой. Это упростит отладку.

Что здесь происходит – по шагам:

1. *IMPORTxml(URL; XPath)* – делает запрос к API Мосбиржи и возвращает значение атрибута PREVLEGALCLOSEPRICE у строки с нужным SECID.
2. *concatenate("//row[@SECID='";B2;"']/@PREVLEGALCLOSEPRICE")* – динамически формирует XPath, подставляя значение из ячейки B2 (наш идентификатор/тикер).
3. *SUBSTITUTE(...; "."; ",")* – меняет точку на запятую, чтобы число выглядело как "русское" (локаль), и корректно форматировалось в таблице.

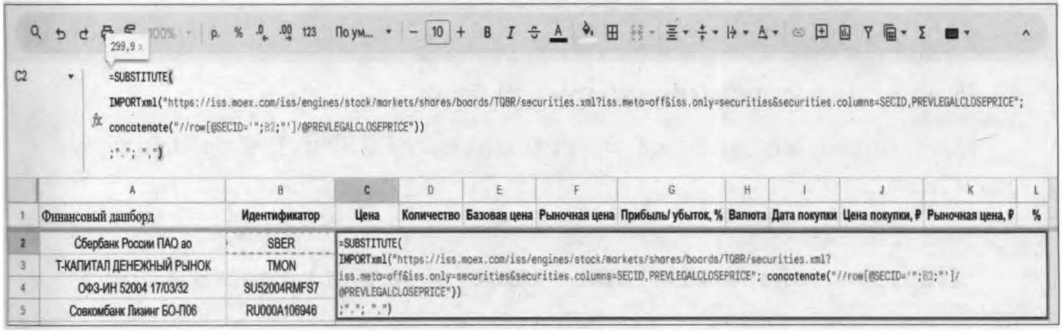


Рис. 7.5. ***Пример запроса к API Мосбиржи и выборка атрибута PREVLEGALCLOSEPRICE***

Как менять URL для других типов активов с Московской биржи

Я также написал готовые формулы для:

- фондов Мосбиржи (TQTF):

```
=SUBSTITUTE(
IMPORTxml("https://iss.moex.com/iss/engines/stock/markets/
shares/boards/TQTF/securities.xml?iss.meta=off&iss.
only=securities&securities.columns=SECID,PREVLEGALCLOSEPRICE";
concatenate("//row[@SECID='";B3;"']/@PREVLEGALCLOSEPRICE"))
;"."; ",")
```

- ОФЗ (TQOB):

```
=SUBSTITUTE(
IMPORTxml("https://iss.moex.com/iss/engines/stock/
markets/bonds/boards/TQOB/securities.xml?iss.meta=off&iss.
only=securities&securities.columns=SECID,PREVLEGALCLOSEPRICE";
concatenate("//row[@SECID='";B4;"']/@PREVLEGALCLOSEPRICE"))
;"."; ",")
```

- корпоративных облигаций (TQCB):

```
=SUBSTITUTE(
IMPORTxml("https://iss.moex.com/iss/engines/stock/
markets/bonds/boards/TQCB/securities.xml?iss.meta=off&iss.
only=securities&securities.columns=SECID,PREVLEGALCLOSEPRICE";
concatenate("//row[@SECID='";B5;"']/@PREVLEGALCLOSEPRICE"))
;"."; ",")
```

Разница – в части **boards** и **shares** или **bonds**.

Остальное остаётся тем же: мы подставляем SECID из B-столбца и получаем цену.

Парсинг сайтов экзотики – монеты, недвижимость и др.

Некоторых активов нет на бирже, и они требуют парсинга коммерческих сайтов.

Здесь у нас есть две связки:

- **Золотая монета Георгий Победоносец 50 рублей**

Формула:

```
=VALUE(REGEXREPLACE(TO_TEXT(IMPORTXML("https://www.zolotoy-zapas.
ru/coins-price/georgiy-pobedonosets-zolotaya-moneta-50-rubley-7-
78-gr-zoloto-chekanka-spmd-2024-god/"; "//div[contains(@class,
'card__table-sale')]//div[@class='price']")); "[^0-9]"; ""))
```

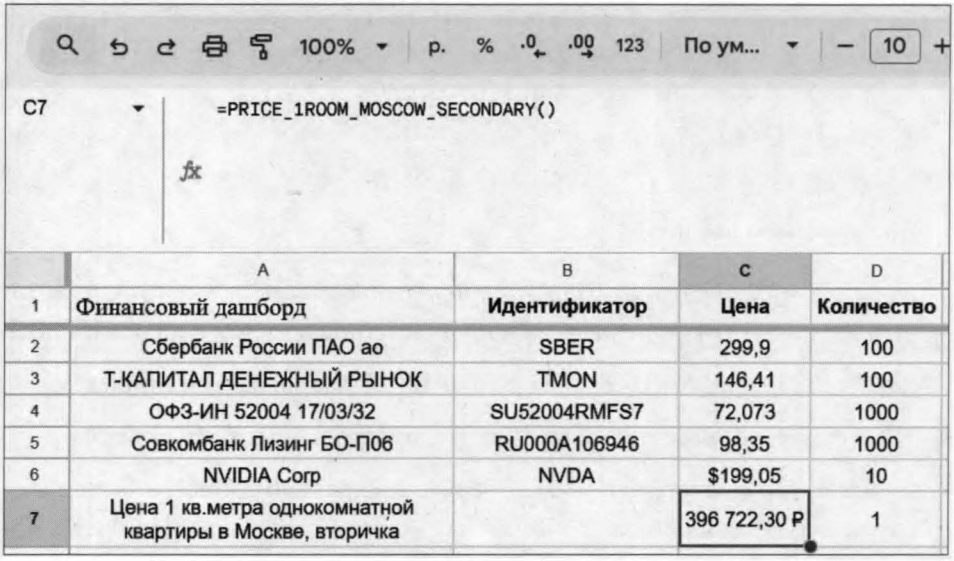
Разбор:

1. IMPORTXML достаёт HTML-узел с ценой.
2. TO_TEXT переводит в текст, чтобы REGEX сработал корректно.
3. REGEXREPLACE(...;"[^0-9]";"") – удаляет все символы, кроме цифр, потому что цена может быть в формате 85 000 ₽ с пробелами и символами.
4. VALUE(...) – превращает очищенную строку в число, пригодное для вычислений.

- **Цена 1 кв.метра однокомнатной квартиры в Москве**

Для цены 1-й комнаты мы используем кастомную функцию

`PRICE_1ROOM_MOSCOW_SECONDARY()` – это типичный пример, когда сайт отдаёт JSON и работать удобнее через Apps Script, а не через IMPORTXML.



	A	B	C	D
1	Финансовый дашборд	Идентификатор	Цена	Количество
2	Сбербанк России ПАО ао	SBER	299,9	100
3	Т-КАПИТАЛ ДЕНЕЖНЫЙ РЫНОК	TMON	146,41	100
4	ОФЗ-ИН 52004 17/03/32	SU52004RMFS7	72,073	1000
5	Совкомбанк Лизинг БО-П06	RU000A106946	98,35	1000
6	NVIDIA Corp	NVDA	$199,05	10
7	Цена 1 кв.метра однокомнатной квартиры в Москве, вторичка		396 722,30 ₽	1

Рис. 7.6. ***Получение цены недвижимости***

Когда простых формул недостаточно – Google Apps Script

IMPORTXML и GOOGLEFINANCE – мощные инструменты, но у них есть предел: JSON-ответы, авторизация, агрегация нескольких полей, сложная логика – всё это лучше делать в скриптах.

Почему Apps Script?

- Может выполнять UrlFetchApp.fetch(url) и получить JSON.
- Позволяет парсить ответ, брать последнюю запись и возвращать скалярное значение в ячейку как функцию =PRICE_1ROOM_MOSCOW_SECONDARY().
- Можно добавлять кэширование, retry-логику и обработку ошибок.

Полная функция PRICE_1ROOM_MOSCOW_SECONDARY():

```
function PRICE_1ROOM_MOSCOW_SECONDARY() {
  const url = "https://msk.restate.ru/action/graph2/data/?region=2
&type=2&period=2&influence=1&money=r&metro=&area=&okrug=&op=1&form=
9&sy=1&cjs=1"; // сайт https://msk.restate.ru/graph/ceny-prodazhi-
1komnatnih/#form9
```

```
  const response = UrlFetchApp.fetch(url);
  const data = JSON.parse(response.getContentText());
  const lastRow = data.rows[data.rows.length - 1];
  return lastRow[2];
}
```

Вот что в нём важно:

- *const url = "https://msk.restate.ru/action/graph2/data/?..."* – это API, который даёт нам структуру данных в JSON (стабильнее, чем парсить HTML).
- *UrlFetchApp.fetch(url)* – скрипт делает HTTP-запрос к этому индексу.
- *JSON.parse(response.getContentText())* – превращает текст в объект.
- *data.rows[data.rows.length - 1]* – берём последнюю запись (обычно там самая актуальная цена).
- *return lastRow[2]* – возвращаем нужное поле.

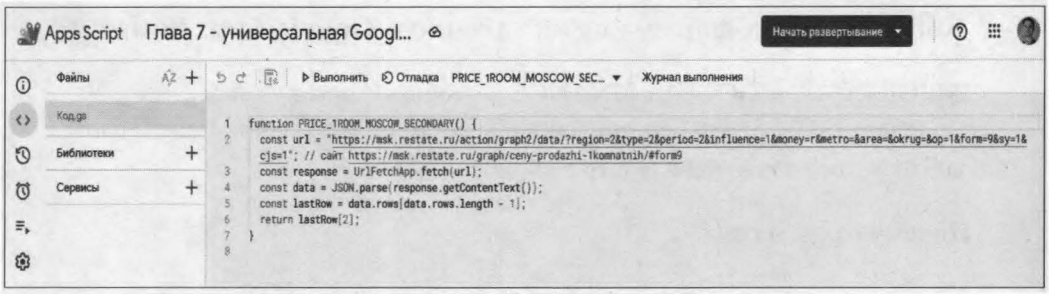


Рис. 7.7. ***Поиск цены через Google Apps Script***

Надёжность, лимиты и безопасная эксплуатация

Google Таблицы/Apps Script имеют свои квоты по UrlFetch и времени выполнения.

В итоге мы получили рабочий алгоритм:

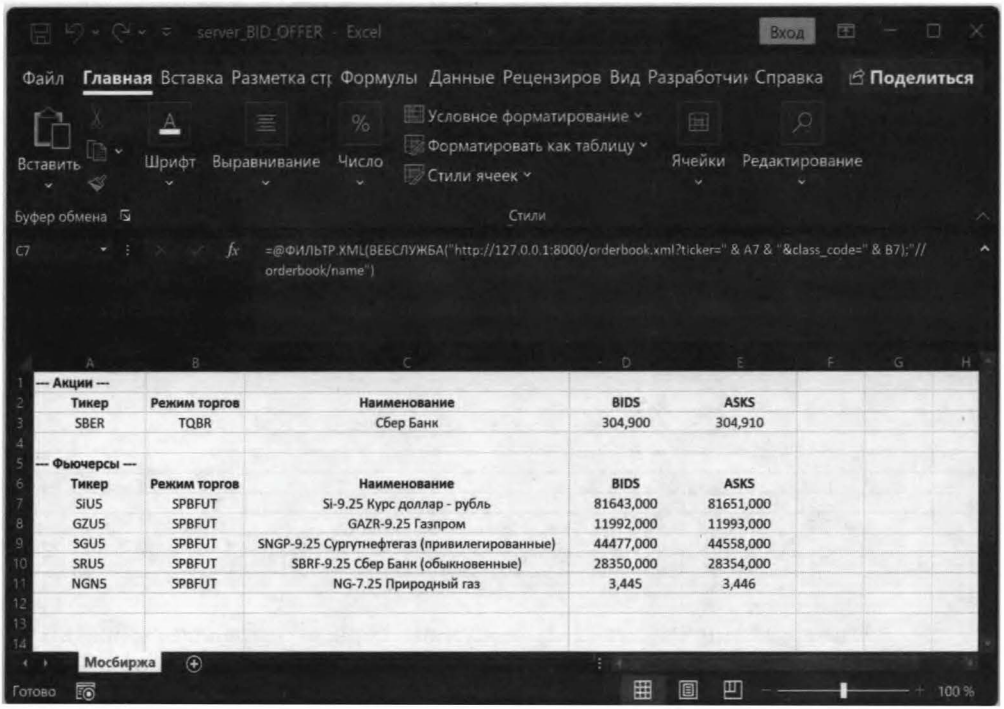
1. Для западных тикеров используем GOOGLEFINANCE.
2. Для Мосбиржи – IMPORTXML с ISS API и динамическим XPath.
3. Для сайтов и "экзотики" – IMPORTXML + REGEXREPLACE + VALUE или кастомная функция на Apps Script (JSON).

7.3. Уровень 2: локальный прокси-сервер: получаем данные, которых нет в открытом доступе

Теперь перейдём к данным, которых нет в открытом доступе. Представим, что вы не просто хотите знать текущую цену, а строите систему, которая отслеживает лучшие цены на покупку и продажу – стакан заявок. Эти данные, лучшие цены BID (покупки) и OFFER (продажи), как правило, недоступны через простые публичные API или веб-сайты без оплаты.

Но выход есть – и он снова лежит в области автоматизации, но уже на более продвинутом уровне. Мы создадим локальный прокси-сервер на вашем собственном компьютере. Это не пугающая инфраструктура в облаке, а просто небольшая программа на Python, запущенная у вас дома. Она будет выступать в роли посредника между вашими таблицами и API вашего брокера.

И да, вместо Google Таблиц будем использовать знакомый всем Excel.



	A	B	C	D	E
1	--- Акции ---				
2	Тикер	Режим торгов	Наименование	BIDS	ASKS
3	SBER	TQBR	Сбер Банк	304,900	304,910
4					
5	--- Фьючерсы ---				
6	Тикер	Режим торгов	Наименование	BIDS	ASKS
7	SiU5	SPBFUT	Si-9.25 Курс доллар - рубль	81643,000	81651,000
8	GZU5	SPBFUT	GAZR-9.25 Газпром	11992,000	11993,000
9	SGU5	SPBFUT	SNGP-9.25 Сургутнефтегаз (привилегированные)	44477,000	44558,000
10	SRU5	SPBFUT	SBRF-9.25 Сбер Банк (обыкновенные)	28350,000	28354,000
11	NGN5	SPBFUT	NG-7.25 Природный газ	3,445	3,446

Рис. 7.8. ***Работает как локальный FastAPI-сервер, который Excel опрашивает через веб-запросы. В ответ скрипт возвращает XML с нужными параметрами.***

Давайте разберем, как это работает, шаг за шагом

1. **Запрос из таблицы**. Мы настраиваем ячейку в Excel или Google Таблицах так, чтобы она делала веб-запрос.

Например,

```
http://127.0.0.1:8000/orderbook.xml?ticker=SBER&class_code=TQBR
```

Этот адрес означает: "Отправь мне XML-файл с данными стакана для акции Сбербанка (SBER) с кодом класса TQBR (акции основного рынка)". 127.0.0.1 – это всегда ваш собственный компьютер, а :8000 – номер порта, на котором будет "слушать" наш сервер.

2. **Запуск сервера**. У вас на компьютере работает Python-скрипт (мы будем использовать фреймворк FastAPI для удобства). Он "открывает" порт 8000 и ждет, когда к нему обратятся по указанному адресу.

3. **Обращение к API брокера**. Получив запрос от таблицы, наш скрипт "понимает", что нужно получить стакан для SBER. Он использует ваш личный токен доступа к API брокера (например, Тинькофф Инвестиции или Алор, Финам, БКС) и делает уже авторизованный запрос к серверу брокера. Именно здесь происходит "магия" – брокер, зная, что запрос от вас, отдает запрашиваемые данные, включая стакан.

4. **Формирование ответа**. Скрипт получает данные стакана (BID и OFFER) от брокера, обрабатывает их и формирует простой XML-ответ. Например:

```
<orderbook>
        <ticker>SBER</ticker>
        <class_code>TQBR</class_code>
        <name>Сбер Банк</name>
        <bid>304,910</bid>
        <offer>304,920</offer>
</orderbook>
```

5. **Ответ таблице**. Этот XML-документ сервер отправляет обратно в Excel или Google Таблицы. Таблица, благодаря встроенным функциям и отображает их в нужных ячейках.

Это, как если бы вы создали собственный, персональный API, который умеет делать то, что не умеет делать публичный API биржи.



Рис. 7.9. ***Схема взаимодействия: Таблица -> Локальный сервер -> API брокера -> Таблица с новыми данными***

Почему это полезно? Потому что вы получаете доступ к любым данным, которые предоставляет ваш API брокера. Стакан заявок – это лишь один из примеров. Вы можете настроить сервер так, чтобы он возвращал:

- Глубокие слои стакана (не только лучшие цены, а, например, первые 5 уровней).
- Информацию о ваших текущих позициях.
- Данные о ваших сделках в реальном времени.
- Расширенные метрики по инструментам (например, греки по опционам).
- Любые другие рыночные данные, доступные через API, но не через публичные сервисы.

Это уже не просто автоматизация учета, а создание профессионального инструмента для анализа и принятия решений, изготовленного под ваши нужды.

Создаем сервер (на примере Тинькофф Инвестиций)

Давайте пройдем основные шаги по созданию такого сервера.

Шаг 1: получение токена брокера.

Вам нужен API-токен от вашего брокера. Для Тинькофф:

- Зайдите в личный кабинет.
- Перейдите в раздел инвестиций.
- Найдите "Настройки" или "Управление токенами".
- Создайте новый токен с ограниченными правами (например, только на чтение рыночных данных и портфеля, без права торговать).

Важно:

Этот токен – ваш ключ к аккаунту. Никогда не публикуйте его и никому не передавайте!

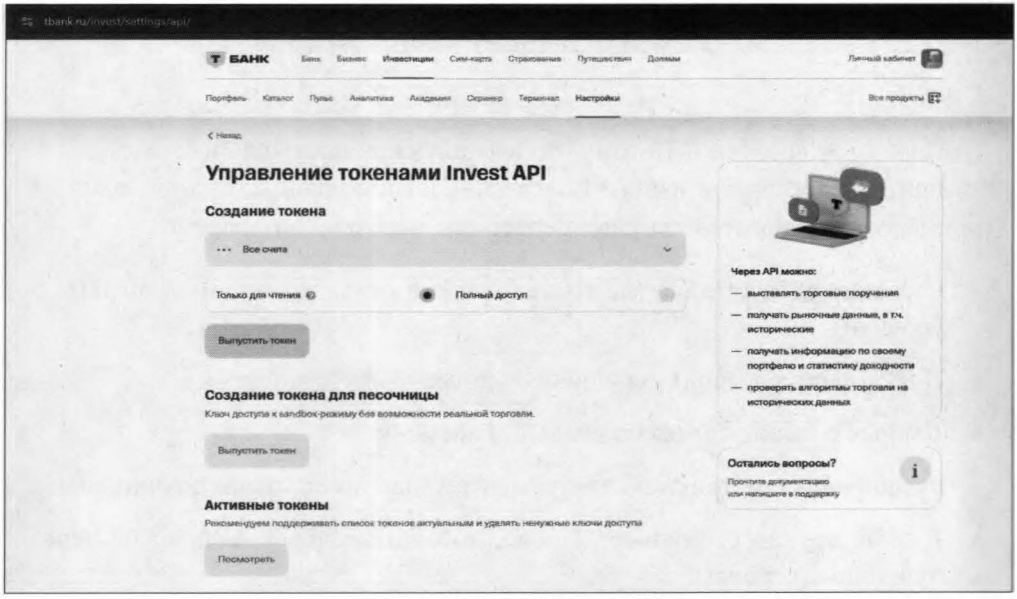


Рис. 7.10. ***Где получить токен***

Сохраните токен в файле `.env` в корневой папке проекта:

```
TINKOFF_TOKEN="ваш_токен_здесь"
```

Шаг 2: Запуск сервера.

Скачайте код проекта с моего GitHub *https://github.com/empenoso/moex2excel_BID_OFFER/*. В проекте будут файлы *server.py* (основной скрипт сервера), *requirements.txt* (список зависимостей) и, возможно, .bat-файлы для Windows.

Установите зависимости. В командной строке, находясь в папке с проектом, выполните:

```
pip install -r requirements.txt
```

Затем запустите сервер:

```
uvicorn server:app --host 127.0.0.1 --port 8000
```

Если всё прошло успешно, вы увидите сообщение о запуске сервера на `localhost:8000`.

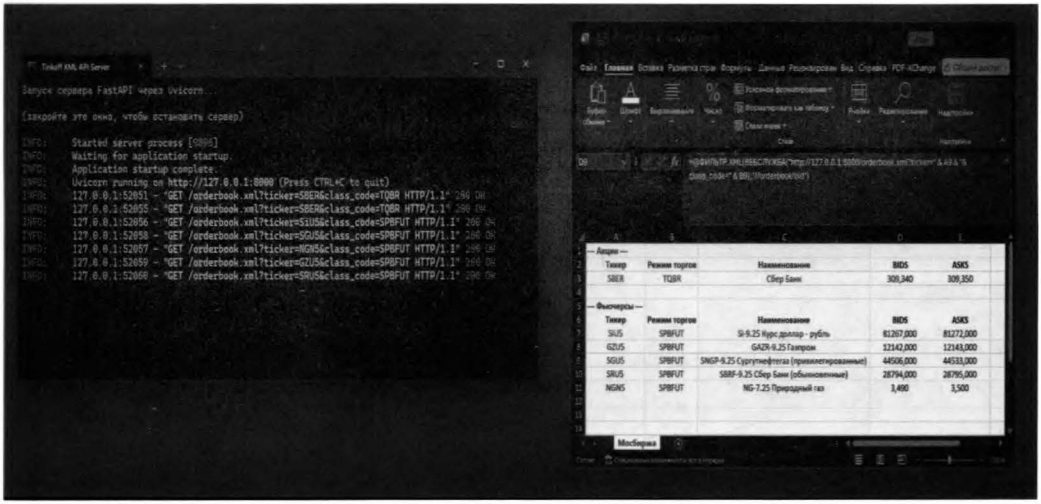


Рис. 7.11. ***Пример запуска сервера в командной строке***

Основной файл *server.py* – он очень короткий:

```
from fastapi import FastAPI, Query
from fastapi.responses import Response
from tinkoff.invest import Client, InstrumentIdType
from tinkoff.invest.exceptions import RequestError
from dotenv import load_dotenv
import os

app = FastAPI()

load_dotenv()
TOKEN = os.getenv("TINKOFF_TOKEN")

def format_price(q):
    return q.price.units + q.price.nano / 1e9

@app.get("/orderbook.xml")
```

```
def get_orderbook_xml(ticker: str = Query(...), class_code: str =
Query(...)):
    with Client(TOKEN) as client:
        try:
            instrument_response = client.instruments.get_instrument_by(
                id_type=InstrumentIdType.INSTRUMENT_ID_TYPE_TICKER,
                id=ticker,
                class_code=class_code,
            )
        except RequestError as e:
            return Response(content=f"<error>{str(e)}</error>", media_
type="application/xml")

        instrument = instrument_response.instrument
        figi = instrument.figi
        name = instrument.name

        orderbook = client.market_data.get_order_book(figi=figi, depth=1)
        best_bid = orderbook.bids[0] if orderbook.bids else None
        best_offer = orderbook.asks[0] if orderbook.asks else None

        bid_price = f"{format_price(best_bid):.2f}".replace('.', ',') if
best_bid else ""
        offer_price = f"{format_price(best_offer):.2f}".replace('.', ',')
if best_offer else ""

        xml = f"""
        <orderbook>
            <ticker>{ticker}</ticker>
            <class_code>{class_code}</class_code>
            <name>{name}</name>
            <bid>{bid_price}</bid>
            <offer>{offer_price}</offer>
        </orderbook>
        """.strip()

        return Response(content=xml, media_type="application/xml")
```

Тестирование: проверяем, что всё работает

Откройте браузер и введите адрес, который будет использовать таблица:

```
http://127.0.0.1:8000/orderbook.xml?ticker=SBER&class_code=TQBR
```

Если вы видите XML-документ с данными по SBER, значит, сервер работает.

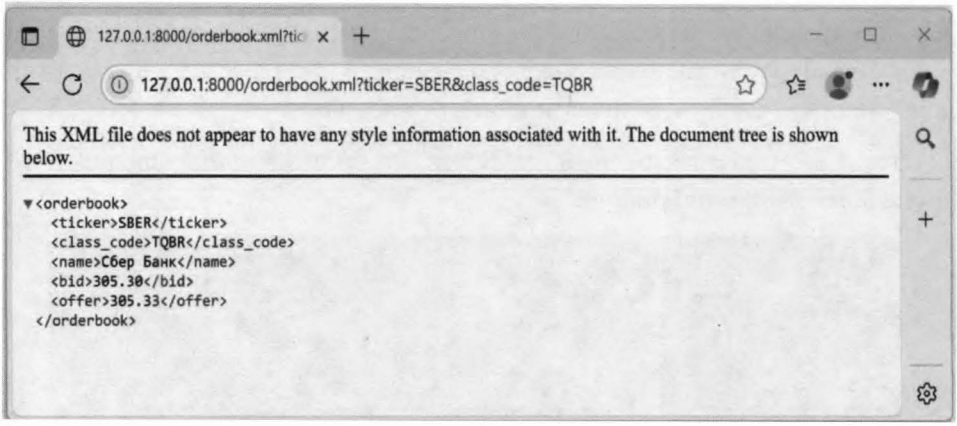


Рис. 7.12. ***Пример XML-ответа сервера в браузере***

В Excel мы используем встроенные инструменты для работы с XML. Создайте запрос к нашему серверу через формулу:

```
=@ФИЛЬТР.XML(ВЕБСЛУЖБА("http://127.0.0.1:8000/orderbook.
xml?ticker=" & A3 & "&class_code=" & B3);"//orderbook/bid")
```

Теперь в этой ячейке будет отображаться актуальная цена покупки по стакану для SBER, обновляемая при пересчете (Ctrl+Alt+F9).

Безопасность и эксплуатация

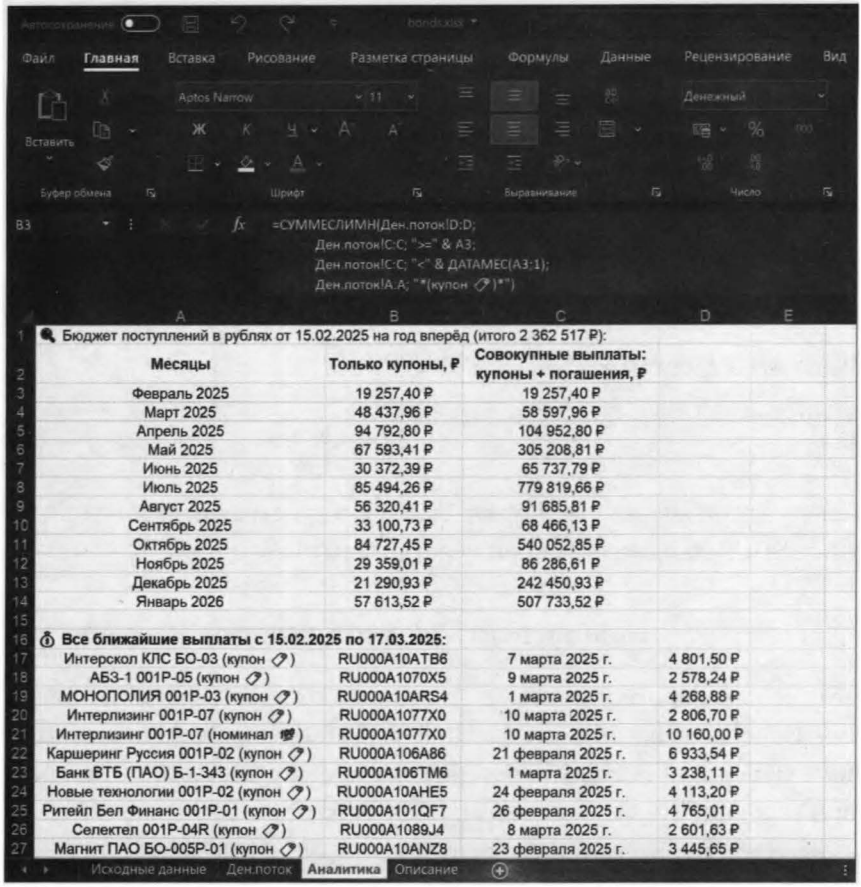
Этот сервер работает локально, на вашем компьютере. Он не открывает доступ извне, если вы этого не настроите. Данные передаются только между вашими таблицами и вашим брокером через вашу программу. Это безопасно, при условии, что вы надежно храните токен.

Чтобы данные в Excel обновлялись, сервер должен быть запущен. Закрыли консоль – сервер остановился, обновление данных прекратилось.

7.4. Уровень 3: автоматизируем поиск и анализ облигаций

Продолжаем работать с Excel. Теперь применим автоматизацию к реальной инвестиционной задаче – поиску и анализу облигаций. Мы не просто будем смотреть на цены – мы научимся автоматически находить выгодные варианты среди тысяч облигаций и прогнозировать будущие денежные потоки. Это превратит наш дашборд из пассивного инструмента учёта в активную систему поддержки инвестиционных решений.

Это мой проект, который выложен на *https://github.com/empenoso/moex-bond-search-and-analysis*



Бюджет поступлений в рублях от 15.02.2025 на год вперёд (итого 2 362 517 ₽):

Месяцы	Только купоны, ₽	Совокупные выплаты: купоны + погашения, ₽
Февраль 2025	19 257,40 ₽	19 257,40 ₽
Март 2025	48 437,96 ₽	58 597,96 ₽
Апрель 2025	94 792,80 ₽	104 952,80 ₽
Май 2025	67 593,41 ₽	305 208,81 ₽
Июнь 2025	30 372,39 ₽	65 737,79 ₽
Июль 2025	85 494,26 ₽	779 819,66 ₽
Август 2025	56 320,41 ₽	91 685,81 ₽
Сентябрь 2025	33 100,73 ₽	68 466,13 ₽
Октябрь 2025	84 727,45 ₽	540 052,85 ₽
Ноябрь 2025	29 359,01 ₽	86 286,61 ₽
Декабрь 2025	21 290,93 ₽	242 450,93 ₽
Январь 2026	57 613,52 ₽	507 733,52 ₽

Все ближайшие выплаты с 15.02.2025 по 17.03.2025:

Интерскол КЛС БО-03 (купон)	RU000A10ATB6	7 марта 2025 г.	4 801,50 ₽
АБЗ-1 001P-05 (купон)	RU000A1070X5	9 марта 2025 г.	2 578,24 ₽
МОНОПОЛИЯ 001P-03 (купон)	RU000A10ARS4	1 марта 2025 г.	4 268,88 ₽
Интерлизинг 001P-07 (купон)	RU000A1077X0	10 марта 2025 г.	2 806,70 ₽
Интерлизинг 001P-07 (номинал)	RU000A1077X0	10 марта 2025 г.	10 160,00 ₽
Каршеринг Руссия 001P-02 (купон)	RU000A106A86	21 февраля 2025 г.	6 933,54 ₽
Банк ВТБ (ПАО) Б-1-343 (купон)	RU000A106TM6	1 марта 2025 г.	3 238,11 ₽
Новые технологии 001P-02 (купон)	RU000A10AHE5	24 февраля 2025 г.	4 113,20 ₽
Ритейл Бел Финанс 001P-01 (купон)	RU000A101QF7	26 февраля 2025 г.	4 765,01 ₽
Селектел 001P-04R (купон)	RU000A1089J4	8 марта 2025 г.	2 601,63 ₽
Магнит ПАО БО-005P-01 (купон)	RU000A10ANZ8	23 февраля 2025 г.	3 445,65 ₽

Рис. 7.13. ***Расчёт для портфеля от февраля 2025 года на 5 млн рублей***

Автоматический расчёт денежного потока

Представим ситуацию: в вашем портфеле десятки разных облигаций. Каждая выплачивает купоны по своему графику – одна раз в квартал, другая дважды в год, третья ежемесячно. Плюс у некоторых есть амортизация номинала, у других – оферты. Отслеживать всё это вручную – значит каждый раз заходить на сайт биржи, искать каждую бумагу по ISIN, смотреть календарь выплат и переносить даты в таблицу.

А главное – брокерские приложения принципиально не учитывают выплаченные купоны в доходности. Вы видите "красный" портфель и думаете об убытках, забывая, что за год получили купонов на круглую сумму. Без учёта всех выплат невозможно понять реальную картину доходности.

Python-скрипт собирает календарь за минуты

Мы напишем скрипт, который за одну команду соберёт все будущие купоны и погашения по всем облигациям из нашей таблицы. Он будет работать следующим образом:

Шаг 1. Готовим входные данные.

В нашем файле *bonds.xlsx* создадим вкладку "Исходные данные" с двумя столбцами:

- **Идентификатор** – ISIN-код облигации (например, RU000A105AE4)
- **Количество** – сколько штук у вас в портфеле

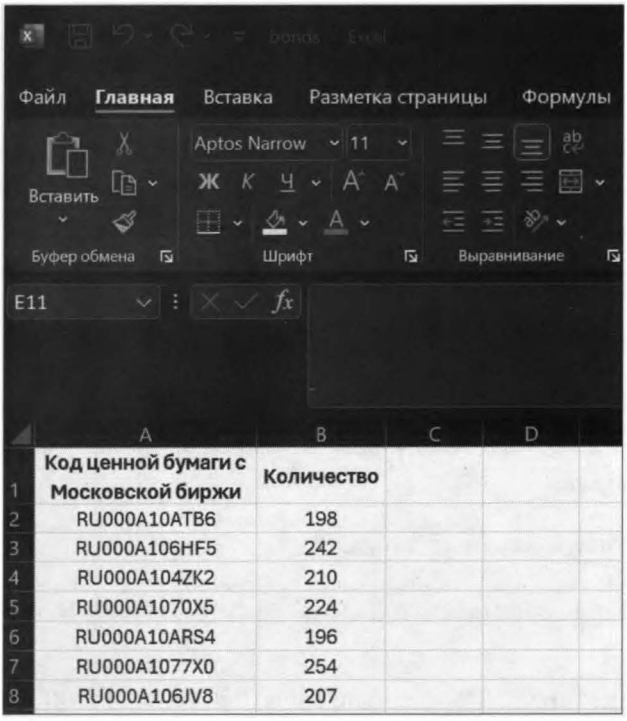
Это всё, что нужно внести вручную.

Шаг 2. Запускаем скрипт.

Скрипт последовательно обрабатывает каждую облигацию из списка:

1. Берёт ISIN-код из первого столбца.
2. Отправляет запрос к API Мосбиржи: *https://iss.moex.com/iss/statistics/engines/stock/markets/bonds/bondization/{ISIN}.json*
3. Получает JSON с полным календарём выплат – и купонами, и амортизациями.

4. Фильтрует только будущие даты (пропуская уже прошедшие).
5. Умножает сумму выплаты на количество облигаций из второго столбца.
6. Записывает результат на вкладку "Ден.поток".



	A	B	C	D
1	**Код ценной бумаги с Московской биржи**	**Количество**		
2	RU000A10ATB6	198		
3	RU000A106HF5	242		
4	RU000A104ZK2	210		
5	RU000A1070X5	224		
6	RU000A10ARS4	196		
7	RU000A1077X0	254		
8	RU000A106JV8	207		

Рис. 7.14. ***Структура вкладки с исходными данными для скрипта***

Вот фрагмент кода:

```
import requests
import openpyxl
from datetime import datetime

# Настройка кодировки для корректного вывода русского текста
import sys
sys.stdout.reconfigure(encoding='utf-8')

# Загружаем Excel-файл
file_path = "bonds.xlsx"
```

```
wb = openpyxl.load_workbook(file_path)
sheet_data = wb["Исходные данные"]
sheet_result = wb["Ден.поток"]

# Очищаем лист с результатами
sheet_result.delete_rows(1, sheet_result.max_row)
sheet_result.append(["Название", "Идентификатор", "Дата выплаты", "Денежный
поток, ₽ (купон | выплата номинала)"])

# Считываем данные из листа "Исходные данные"
ArraySymbolQuantity = []
for row in sheet_data.iter_rows(min_row=2, max_row=sheet_data.max_row,
values_only=True):
    if row[0] and row[1]:  # Проверяем, что данные не пустые
        ArraySymbolQuantity.append(row)

print(f"Считано {len(ArraySymbolQuantity)} облигаций для обработки.")

CashFlow = []

# Обрабатываем каждую облигацию
for ID, number in ArraySymbolQuantity:
    print(f"\nОбрабатываем {ID}, количество: {number} шт.")
    url = f"https://iss.moex.com/iss/statistics/engines/stock/markets/bonds/
bondization/{ID}.json?iss.meta=off"
    print(f"Запрос к {url}")

    response = requests.get(url)
    json_data = response.json()

    # Обработка купонов
    for coupon in json_data.get("coupons", {}).get("data", []):
        name = coupon[1].replace('"', '').replace("'", '').replace("\\", '')
        isin = coupon[0]
        coupon_date = coupon[3]

        # Преобразуем дату в объект datetime
```

```
        coupon_datetime = datetime.strptime(coupon_date, "%Y-%m-%d")

        if coupon_datetime > datetime.now():
            value_rub = (coupon[9] or 0) * number
            CashFlow.append([f"{name} (купон 🎟)", isin, coupon_datetime,
value_rub])
            print(f"Добавлен купон: {CashFlow[-1]}")

    # Обработка выплат номинала
    for amort in json_data.get("amortizations", {}).get("data", []):
        name = amort[1].replace('"', '').replace("'", '').replace("\\", '')
        isin = amort[0]
        amort_date = amort[3]

        # Преобразуем дату в объект datetime
        amort_datetime = datetime.strptime(amort_date, "%Y-%m-%d")

        if amort_datetime > datetime.now():
            value_rub = (amort[9] or 0) * number
            CashFlow.append([f"{name} (номинал 💯)", isin, amort_datetime,
value_rub])
            print(f"Добавлена выплата номинала: {CashFlow[-1]}")

# Записываем данные в Excel
for row in CashFlow:
    sheet_result.append(row)

# Устанавливаем формат ячеек
for cell in sheet_result["C"][1:]:  # Пропускаем заголовок
    cell.number_format = "DD.MM.YYYY"

for cell in sheet_result["D"][1:]:
    cell.number_format = '# ##0,00 ₽'

# Добавляем запись об обновлении
update_message = f"\nДанные автоматически обновлены {datetime.now().strftime
('%d.%m.%Y в %H:%M:%S')}"
sheet_result.append(["", update_message])
```

```
print(update_message)

# Сохраняем изменения в файле
wb.save(file_path)
print(f"Файл {file_path} успешно обновлён.")
```



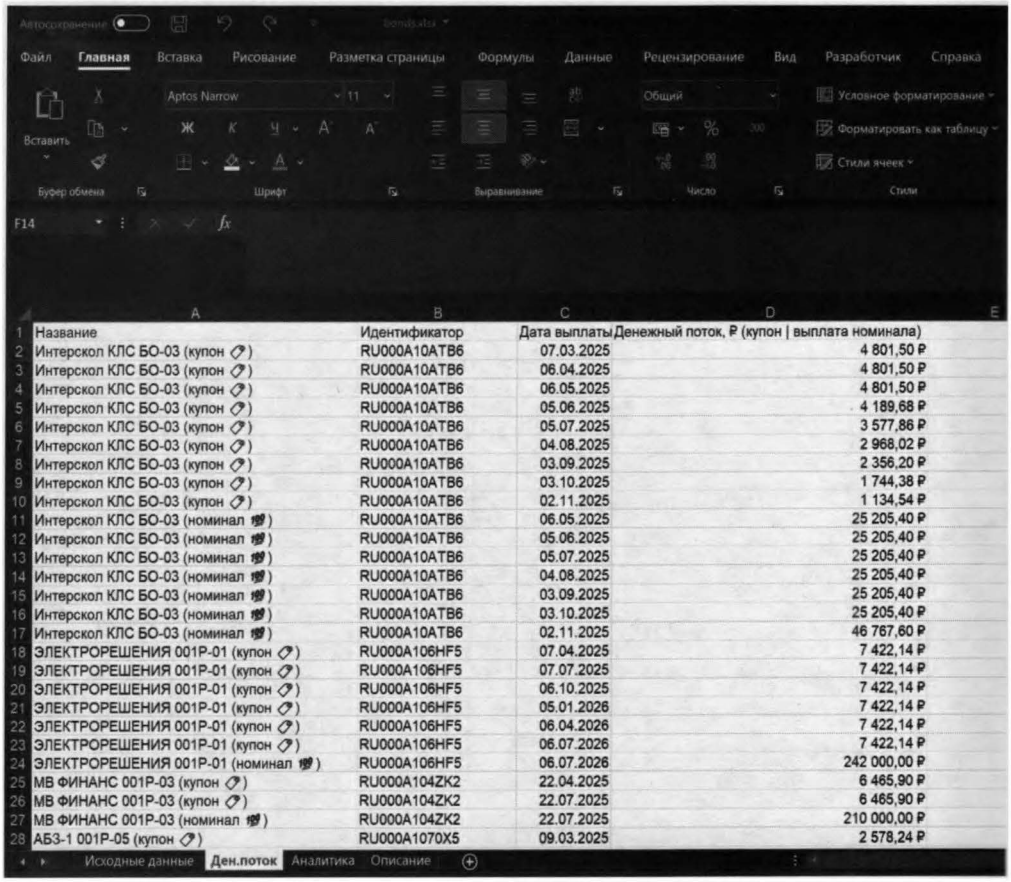
Рис. 7.15. ***Пример лога работы скрипта в консоли***

Шаг 3. Анализируем результат.

После выполнения скрипта, на вкладке "Ден.поток" появляется таблица со всеми будущими выплатами. Каждая строка – это конкретное событие, а именно купон по конкретной облигации в конкретную дату на конкретную сумму.

Теперь мы можем построить сводную таблицу и увидеть:

- Сколько денег придёт в каждом месяце
- Какие облигации приносят больше всего кэша
- Когда ожидается погашение номинала



	A	B	C	D
1	Название	Идентификатор	Дата выплаты	Денежный поток, ₽ (купон | выплата номинала)
2	Интерскол КЛС БО-03 (купон)	RU000A10ATB6	07.03.2025	4 801,50 ₽
3	Интерскол КЛС БО-03 (купон)	RU000A10ATB6	06.04.2025	4 801,50 ₽
4	Интерскол КЛС БО-03 (купон)	RU000A10ATB6	06.05.2025	4 801,50 ₽
5	Интерскол КЛС БО-03 (купон)	RU000A10ATB6	05.06.2025	4 189,68 ₽
6	Интерскол КЛС БО-03 (купон)	RU000A10ATB6	05.07.2025	3 577,86 ₽
7	Интерскол КЛС БО-03 (купон)	RU000A10ATB6	04.08.2025	2 968,02 ₽
8	Интерскол КЛС БО-03 (купон)	RU000A10ATB6	03.09.2025	2 356,20 ₽
9	Интерскол КЛС БО-03 (купон)	RU000A10ATB6	03.10.2025	1 744,38 ₽
10	Интерскол КЛС БО-03 (купон)	RU000A10ATB6	02.11.2025	1 134,54 ₽
11	Интерскол КЛС БО-03 (номинал)	RU000A10ATB6	06.05.2025	25 205,40 ₽
12	Интерскол КЛС БО-03 (номинал)	RU000A10ATB6	05.06.2025	25 205,40 ₽
13	Интерскол КЛС БО-03 (номинал)	RU000A10ATB6	05.07.2025	25 205,40 ₽
14	Интерскол КЛС БО-03 (номинал)	RU000A10ATB6	04.08.2025	25 205,40 ₽
15	Интерскол КЛС БО-03 (номинал)	RU000A10ATB6	03.09.2025	25 205,40 ₽
16	Интерскол КЛС БО-03 (номинал)	RU000A10ATB6	03.10.2025	25 205,40 ₽
17	Интерскол КЛС БО-03 (номинал)	RU000A10ATB6	02.11.2025	46 767,60 ₽
18	ЭЛЕКТРОРЕШЕНИЯ 001P-01 (купон)	RU000A106HF5	07.04.2025	7 422,14 ₽
19	ЭЛЕКТРОРЕШЕНИЯ 001P-01 (купон)	RU000A106HF5	07.07.2025	7 422,14 ₽
20	ЭЛЕКТРОРЕШЕНИЯ 001P-01 (купон)	RU000A106HF5	06.10.2025	7 422,14 ₽
21	ЭЛЕКТРОРЕШЕНИЯ 001P-01 (купон)	RU000A106HF5	05.01.2026	7 422,14 ₽
22	ЭЛЕКТРОРЕШЕНИЯ 001P-01 (купон)	RU000A106HF5	06.04.2026	7 422,14 ₽
23	ЭЛЕКТРОРЕШЕНИЯ 001P-01 (купон)	RU000A106HF5	06.07.2026	7 422,14 ₽
24	ЭЛЕКТРОРЕШЕНИЯ 001P-01 (номинал)	RU000A106HF5	06.07.2026	242 000,00 ₽
25	МВ ФИНАНС 001P-03 (купон)	RU000A104ZK2	22.04.2025	6 465,90 ₽
26	МВ ФИНАНС 001P-03 (купон)	RU000A104ZK2	22.07.2025	6 465,90 ₽
27	МВ ФИНАНС 001P-03 (номинал)	RU000A104ZK2	22.07.2025	210 000,00 ₽
28	АБЗ-1 001P-05 (купон)	RU000A1070X5	09.03.2025	2 578,24 ₽

Рис. 7.16. ***Сводная таблица денежных потоков по месяцам***

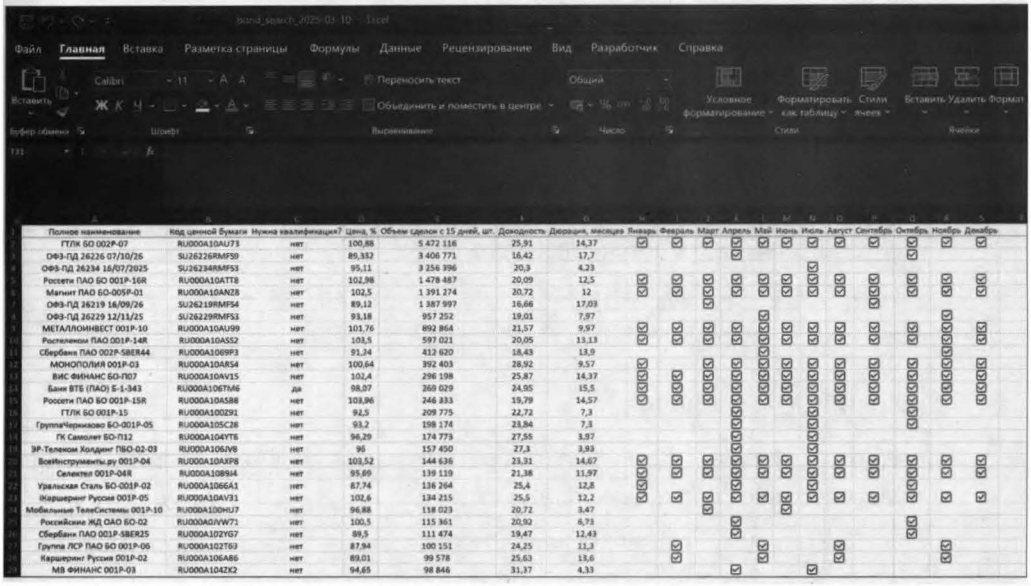
Эта автоматизация решает сразу несколько задач:

- **Прогнозирование**. Вы видите, что в июле придёт 15 000 рублей купонов, в августе – 8 000, а в сентябре – 22 000 плюс погашение одной бумаги. Можно заранее планировать реинвестирование.
- **Контроль доходности**. Сравнивая суммы купонов с вложениями, вы видите реальную доходность портфеля – не ту, что показывает брокер, а фактическую, с учётом всех поступлений.
- **Оптимизация структуры**. Если вы только формируете портфель, можно подобрать такое соотношение облигаций, чтобы купоны приходили равномерно каждый месяц или даже каждую неделю. Просто меняйте количество в столбце "Количество" и запускайте скрипт снова – он пересчитает всё за секунды.

Автоматизация поиска выгодных облигаций. Задача: найти иголку в стоге сена

Вообще, на Московской бирже торгуется более 2500 облигаций. Звучит как огромный выбор, но подавляющее большинство этих бумаг совершенно непригодны для частного инвестора. Вот что мы обнаруживаем при детальном анализе:

- **Неликвидные выпуски**. По большинству облигаций в стакане почти нет предложений, сделок совершается крайне мало или вообще не совершается. Формально бумага торгуется, но купить её невозможно – либо придётся ждать неделями, либо покупать по завышенной цене.
- **Низкая доходность**. Многие облигации дают доходность ниже банковских вкладов – зачем тогда связываться с риском дефолта эмитента?
- **Скрытые ловушки**. Оферты через месяц, флоатеры с непредсказуемыми выплатами, бумаги эмитентов на грани банкротства.
- **Вручную просматривать тысячи строк на сайтах вроде *rusbonds.ru* или в приложении брокера** – это не просто утомительно, это практически невозможно делать регулярно. А главная проблема – известные публичные сервисы не суммируют объёмы торгов за период, поэтому сложно быстро найти действительно ликвидные бумаги.



Полное наименование	Код ценной бумаги	Нужна квалификация?	Цена, %	Объем сделок с 15 дней, шт.	Доходность	Дюрация, месяцев	Январь	Февраль	Март	Апрель	Май	Июнь	Июль	Август	Сентябрь	Октябрь	Ноябрь	Декабрь
ГТЛК БО 002P-07	RU000A10AU73	нет	100,88	5 472 116	25,91	14,37	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
ОФЗ-ПД 26226 07/10/25	SU26226RMFS0	нет	89,332	3 406 771	16,42	17,7				☑						☑		
ОФЗ-ПД 26234 18/07/2025	SU26234RMFS3	нет	95,11	3 256 396	20,3	4,23							☑					
Россети ПАО БО 001P-16R	RU000A10ATT8	нет	102,98	1 478 487	20,09	12,5	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
Магнит ПАО БО-005P-01	RU000A10ANZ8	нет	102,5	1 391 274	20,72	12	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
ОФЗ-ПД 26219 16/09/26	SU26219RMFS4	нет	89,12	1 387 997	16,66	17,03				☑								
ОФЗ-ПД 26229 12/11/25	SU26229RMFS3	нет	93,18	957 252	19,01	7,87					☑						☑	
МЕТАЛЛОИНВЕСТ 001P-10	RU000A10AU99	нет	101,76	892 864	21,57	9,97	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
Ростелеком ПАО 001P-14R	RU000A10AS52	нет	103,5	597 021	20,05	13,13	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
Сбербанк ПАО 002P-SBER44	RU000A10B9P3	нет	91,34	412 620	18,43	13,9					☑							
МОНОПОЛИЯ 001P-03	RU000A10ARS4	нет	100,64	392 403	28,92	9,57	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
ВИС ФИНАНС БО-П07	RU000A10AV15	нет	102,4	296 198	25,87	14,37	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
Банк ВТБ (ПАО) Б-1-343	RU000A106TM6	да	98,07	260 029	24,95	15,5	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
Россети ПАО БО 001P-15R	RU000A10AS88	нет	103,96	246 333	19,79	14,57	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
ГТЛК БО 001P-15	RU000A100Z91	нет	92,5	209 775	22,72	7,3				☑	☑	☑	☑	☑	☑	☑	☑	☑
ГруппаЧерномор БО-001P-05	RU000A105C28	нет	93,2	198 174	23,84	7,3				☑	☑	☑	☑	☑	☑	☑	☑	☑
ГК Самолет БО-П12	RU000A1049T5	нет	96,29	174 773	27,56	3,97				☑	☑	☑	☑	☑	☑	☑	☑	☑
ЭР-Телеком Холдинг ПБО-02-03	RU000A106JV8	нет	96	157 450	27,3	3,93				☑	☑	☑	☑	☑	☑	☑	☑	☑
ВсеИнструменты.ру 001P-04	RU000A10AXP8	нет	103,52	144 636	23,31	14,67	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
Селектел 001P-04R	RU000A10B9J4	нет	95,69	139 119	21,38	11,97	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
Уральская Сталь БО-001P-02	RU000A1066A1	нет	87,74	136 264	25,4	12,8	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
Каршеринг Руссия 001P-05	RU000A10AV31	нет	102,6	134 215	25,5	12,2	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
Мобильные ТелеСистемы 001P-10	RU000A100HU7	нет	96,88	118 023	20,72	3,47				☑								
Российские ЖД ОАО БО-02	RU000A0JNW71	нет	100,5	115 361	20,92	6,73					☑					☑		
Сбербанк ПАО 001P-SBER25	RU000A102YG7	нет	89,5	111 474	19,47	12,43				☑						☑		
Группа ЛСР ПАО БО 001P-06	RU000A102T6J	нет	87,94	100 151	24,25	11,3		☑			☑			☑			☑	
Каршеринг Руссия 001P-02	RU000A10BA86	нет	89,01	99 578	25,63	13,6		☑			☑			☑			☑	
МВ ФИНАНС 001P-03	RU000A104ZK2	нет	94,65	98 846	31,37	4,33				☑			☑					

Рис. 7.17. ***Результаты поиска на дату, файл bond_search_2025-03-25.xlsx***

Мы создадим Python-скрипт, который за несколько секунд просканирует весь рынок облигаций и вернёт только те бумаги, которые соответствуют строгим критериям ликвидности и доходности. Это, как если бы у вас был личный аналитик, который круглосуточно мониторит биржу и отсеивает весь мусор.

Шаг 1. Определяем критерии отбора.

Скрипт использует многоуровневую систему фильтрации:

Критерий №1: Доходность

Эффективная доходность облигации – главный параметр. Задаём диапазон, например, от 15% до 30%. Важно понимать: этот показатель не включает налог с купонов (13% для резидентов) и комиссии брокера, но даёт объективную базу для сравнения.

```
MIN_YIELD = 15.0  # Минимальная доходность к погашению, %
MAX_YIELD = 30.0  # Верхняя граница (слишком высокая может
говорить о проблемах)
```

Критерий №2: Текущая цена

Облигации торгуются по разным ценам относительно номинала. Этот параметр позволяет выбрать стратегию получения дохода:

» **Цена ниже номинала** (например, 86%): основная доходность формируется к моменту погашения за счёт разницы между ценой покупки и номиналом 100%. Пример: ЕвроТранс БО-001Р-03 с купонной доходностью 13,6% и доходностью к погашению 26,78% – разница идёт именно от дисконта к номиналу.

» **Цена около номинала** (98-102%): основной доход – это регулярные купонные выплаты в течение всего срока.

```
MIN_PRICE = 85.0  # Минимальная цена, % от номинала
MAX_PRICE = 105.0  # Максимальная цена
```

Критерий №3: Дюрация

Показатель дюрации позволяет выбрать бумаги с нужным сроком жизни и чувствительностью к изменению ставок.

```
MIN_DURATION = 365  # Минимальная дюрация, дней (отсекаем слишком
короткие)
MAX_DURATION = 1095  # Максимальная дюрация, дней (3 года)
```

Критерий №4: Ликвидность – ключевой параметр

Это то, ради чего создавался скрипт. Анализируются два показателя:

```
# Минимальное число сделок за каждый из последних 15 дней
MIN_TRADES_PER_DAY = 10
# Общий объём торгов за 15 дней, млн ₽
MIN_TOTAL_VOLUME = 15.0
```

Первый параметр исключает облигации, которые могут внезапно "замереть" – если хотя бы в один день было меньше 10 сделок, бумага отсеивается. Второй параметр выявляет устойчивый спрос – 15 миллионов за две недели означают, что вы сможете купить и продать без проблем.

Критерий №5: Прозрачность выплат

```
EXCLUDE_FLOATERS = True  # Исключить флоатеры (переменный купон)
MIN_DAYS_TO_OFFER = 180  # Минимум дней до оферты
```

Флоатеры исключаются, потому что Мосбиржа не передаёт данные о будущих платежах по ним – невозможно спрогнозировать кэш-флоу. Оферты ближе 180 дней тоже отсекаются – слишком короткий горизонт планирования.

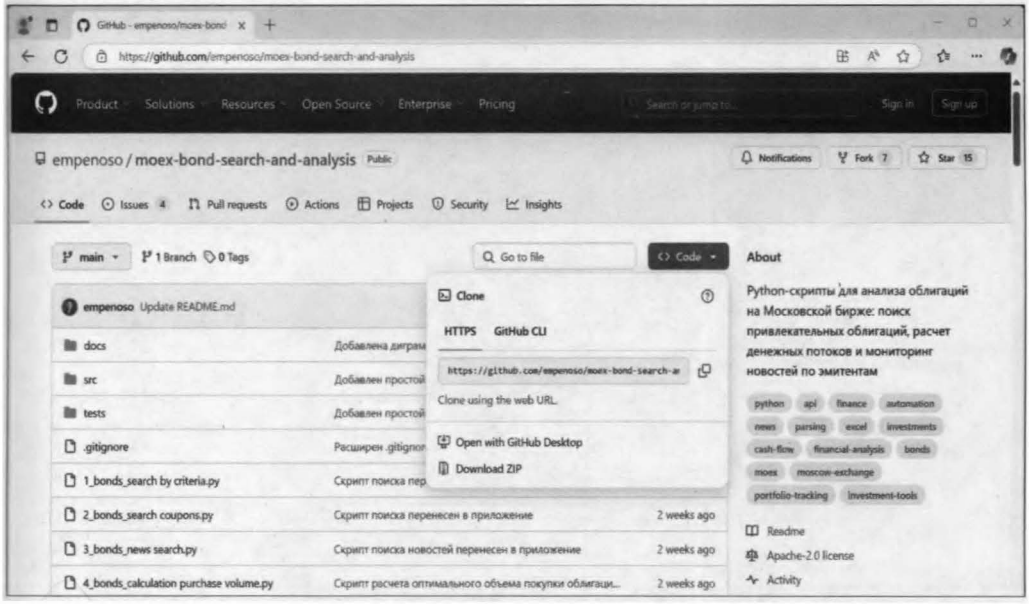


Рис. 7.18. ***Скрипт слишком большой и поэтому я не привожу его в книге, но он всегда свободно доступен на*** *GitHub: https://github.com/empenoso/moex-bond-search-and-analysis*

Шаг 2. Скрипт собирает данные с биржи.

Скрипт работает в два этапа. Сначала получает общий список облигаций:

```
import requests
import pandas as pd
from datetime import datetime, timedelta

# Запрашиваем все облигации
url = "https://iss.moex.com/iss/engines/stock/markets/bonds/boards/
TQCB/securities.json"
params = {
    "iss.meta": "off",
    "iss.only": "securities,marketdata",
    "securities.columns": "SECID,ISIN,SHORTNAME,FACEVALUE,COUPONVALUE,
MATDATE,OFFERDATETO",
    "marketdata.columns": "SECID,LAST,YIELD,DURATION"
}

response = requests.get(url, params=params)
data = response.json()
Затем для каждой облигации запрашивает историю торгов за последние 15
дней:
python
def get_trading_history(secid):
    """Получает объёмы торгов за последние 15 дней"""
    end_date = datetime.now()
    start_date = end_date - timedelta(days=15)

    url = f"https://iss.moex.com/iss/history/engines/stock/markets/
bonds/boards/TQCB/securities/{secid}.json"
    params = {
        "from": start_date.strftime("%Y-%m-%d"),
        "till": end_date.strftime("%Y-%m-%d")
    }

    response = requests.get(url, params=params)
    history = response.json()['history']['data']
```

```
    # Подсчитываем сделки и объёмы по дням
    daily_trades = [row[6] for row in history if row[6]]  # Количество
сделок
    total_volume = sum([row[5] for row in history if row[5]]) /
1_000_000  # Объём в млн

    return min(daily_trades) if daily_trades else 0, total_volume
```

Шаг 3. Применяем фильтры последовательно

Теперь каждая облигация проходит через все проверки:

```
filtered_bonds = []

for security in securities_data:
    secid = security['SECID']

    # Получаем историю торгов
    min_trades, total_volume = get_trading_history(secid)

    # Фильтр: ликвидность
    if min_trades < MIN_TRADES_PER_DAY:
        continue
    if total_volume < MIN_TOTAL_VOLUME:
        continue

    # Фильтр: доходность
    if not security['YIELD'] or float(security['YIELD']) < MIN_YIELD:
        continue

    # Фильтр: цена
    price = float(security['LAST'])
    if price < MIN_PRICE or price > MAX_PRICE:
        continue

    # Фильтр: дюрация
    duration = int(security['DURATION'])
    if duration < MIN_DURATION or duration > MAX_DURATION:
```

```
            continue

        # Фильтр: оферта
        if security['OFFERDATETO']:
            days_to_offer = (datetime.strptime(security['OFFERDATETO'],
"%Y-%m-%d") - datetime.now()).days
            if days_to_offer < MIN_DAYS_TO_OFFER:
                continue

        # Облигация прошла все фильтры!
        filtered_bonds.append({
            'ISIN': security['ISIN'],
            'Название': security['SHORTNAME'],
            'Доходность, %': round(float(security['YIELD']), 2),
            'Цена, %': round(price, 2),
            'Дюрация, дней': duration,
            'Объём 15д, млн ₽': round(total_volume, 2),
            'Мин. сделок/день': min_trades
        })
```

Шаг 4. Сохраняем результат.

Скрипт создаёт Excel-файл с датой в названии:

```
df = pd.DataFrame(filtered_bonds)
df = df.sort_values('Доходность, %', ascending=False)

filename = f"bond_search_{datetime.now().strftime('%Y-%m-%d')}.xlsx"
df.to_excel(filename, index=False)

print(f"Найдено {len(filtered_bonds)} ликвидных облигаций из
{len(securities_data)} проанализированных")
print(f"Результат сохранён в {filename}")
```

Вместо 2500 строк на сайте биржи у нас теперь компактная таблица с 15-20 облигациями, которые реально можно купить прямо сейчас.



Рис. 7.19. ***Работа скрипта***

Заключение главы: от ручного учета к системе поддержки решений

Эта глава помогла нам свести данные из совершенно разных источников в одну Google Таблицу. Мы научились собирать данные для этой таблицы несколькими способами:

- через встроенные функции Google;
- через парсинг функцией IMPORTXML;
- и для самых сложных случаев приготовили скрипты на Google Apps Script.

А ещё мы расширили функционал знакомого всем Microsoft Excel – научили его через небольшую Python-прослойку получать данные напрямую из API брокера.

А ещё мы решали конкретные инвестиционные задачи: прогнозирование будущего денежного потока по портфелю облигаций и автоматический поиск наиболее ликвидных и доходных бумаг на рынке.

В результате, наши таблицы перестали быть просто инструментом для учета, они превратились в полноценную систему поддержки принятия решений.

Глава 8.

Взламываем повседневность: подключаемся к API любимых сервисов

Представьте, что вы находитесь в ресторане и голодны. А кухня в виде сервера (это веб-сайты или приложения) предлагает вам только то, что у них есть в списке меню. А если вы захотите чтобы приготовилось ваше собственное блюдо из тех же компонентов? Или хотите получить уведомление, когда нужное блюдо уже приготовится? Когда всё получится, вы станете не просто клиентом, а умелым интегратором и API станет вашим пультом управления.

Целью этой главы является выход за пределы привычного интерфейса, чтобы напрямую разговаривать с теми системами, от которых зависят ваша работа и жизнь. В этой главе разберёмся с API – и это не страшная аббревиатура, доступная только программистам, а универсальный способ говорить с цифровым миром. Ещё научимся находить служебные входы в любимые сервисы, будь то сайт фитнес-клуба, личный кабинет брокера или даже ваш собственный Telegram-канал. Научимся использовать как официальные, так и скрытые API, чтобы автоматизировать рутину и собирать данные.

8.1. API как секретный вход в любой сервис

Давайте забудем о скучном определении "программный интерфейс приложения". Представьте, что вы пришли в ресторан и ваша программа – это голодный клиент. Она хочет получить конкретное блюдо (данные), но не знает, как его готовить, и не имеет доступа на кухню (сервер). Здесь на сцену выходит API – персональный официант.

Программа дает официанту (API) короткий и понятный заказ (запрос), например: "Пожалуйста, принесите мне расписание групповых тренировок

на завтра". Официант уходит на кухню (сервер), передает заказ поварам (системе), и через некоторое время возвращается с готовым блюдом (данными). Вашей программе не нужно знать рецепт, бегать за продуктами или стоять у плиты. Она получает результат в удобном виде. API работает точно так же: это посредник, который принимает ваши простые команды и возвращает готовые данные, скрывая всю сложность серверной "кухни".

Теперь, когда мы поняли принцип, разберемся, какими бывают эти "официанты". Условно их можно разделить на два типа, которые мы назовем "белый" и "серый":

1. **Официальные (белый путь):** это парадный вход в сервис. Как мы уже делали в главе 7, когда подключались к API Московской биржи, официальные API созданы специально для внешних разработчиков. У них есть подробная документация, где описаны все "блюда в меню" (какие данные можно запросить), четкие правила использования и часто ключи доступа – ваш персональный пропуск.
2. **"Скрытые" (серый путь):** это служебный вход, который мы научимся находить. Когда вы заходите на сайт фитнес-клуба и нажимаете кнопку "показать расписание на следующую неделю", страница не перезагружается целиком. Данные подгружаются незаметно, и происходит это как раз через API. Этот API не предназначен для широкой публики, у него нет документации, но он существует и работает. Мы можем "подслушать" запрос, который сайт отправляет сам себе, и научиться делать такие же запросы для наших целей. Это и есть наш "серый" путь – использование внутренних механизмов сайта для решения своих задач.

> Как найти этот служебный вход? Для этого не понадобится сложное оборудование. Наш главный инструмент уже встроен в ваш браузер – это **инструменты разработчика** (DevTools). Открыть просто – нажмите клавишу **F12** или используйте комбинацию клавиш **Ctrl+Shift+I**.

Нас интересует вкладка **Network (Сеть)**. На ней отображаются абсолютно все запросы, которые ваш браузер отправляет в Интернет при загрузке и использовании сайта.

Ключевой момент для понимания: любая информация, которая появляется на сайте без полной перезагрузки страницы (новые товары в каталоге, сообщения в чате, расписание на другую дату), скорее всего, была получена через такой скрытый API.

Давайте проведем эксперимент. Откройте сайт с динамической подгрузкой контента (например, тот же фитнес-клуб или маркетплейс), перейдите на вкладку "Сеть", а затем на самом сайте нажмите кнопку "Показать еще" или выберите другую дату в календаре. Вы увидите, как в списке на вкладке "Сеть" появится одна или несколько новых строк. Это и есть те самые запросы к API.

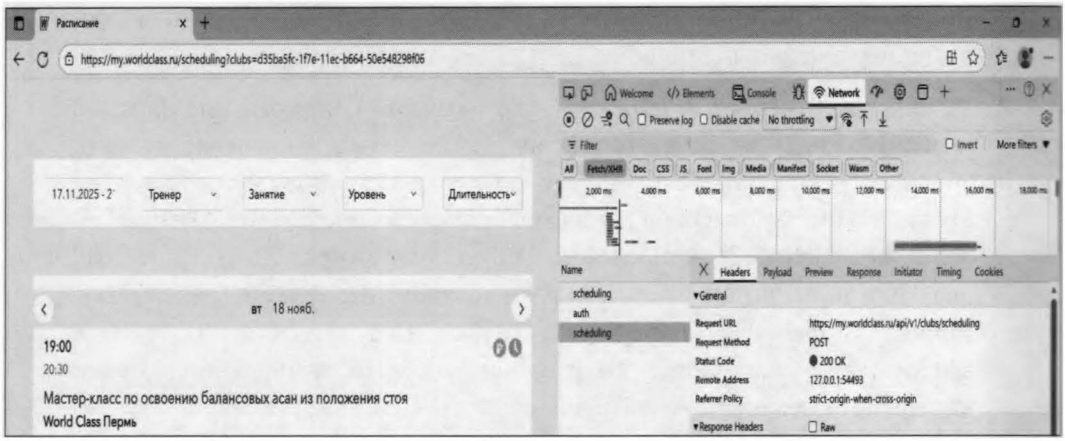


Рис. 8.1. ***Вкладка Network в инструментах разработчика, где виден XHR-запрос к API сайта за получением новых данных***

Чтобы отфильтровать только запросы за данными и скрыть загрузку картинок и стилей, нажмите на фильтр **Fetch/XHR**. Это покажет нам "чистые" API-запросы. Кликнув на любой из них, мы можем изучить детали: адрес, на который ушел запрос, какие параметры были переданы, и самое главное – посмотреть ответ от сервера во вкладке **Preview** или **Response**. Часто эти данные приходят в удобном структурированном формате JSON, с которым мы уже умеем работать ещё в Python. Так мы и находим наш "секретный вход".

8.2. API в действии: автоматизируем расписание

Представим, что каждую неделю у нас одно и то же: нужно открыть сайт фитнес-клуба, выбрать свой филиал, закрыть несколько рекламных баннеров и, наконец, найти расписание любимых тренировок. Потом – вручную перенести их в Google Календарь, чтобы не пропустить. На всё это уходит минут пятнадцать, и каждый раз я обещал себе: "вот на следующей неделе точно напишу скрипт".

Мы решим задачу, знакомую тем, кто пользуется онлайн-расписаниями – будь то фитнес-центр, детский кружок или расписание вебинаров. Мы узнаем, как подглядеть за тем, как сайт сам получает данные с сервера, научимся повторять эти запросы и полностью автоматизируем обновление календаря.

Шаг 1. Расследование: ищем "скрытый" API.

Открываем сайт нашего фитнес-клуба (в моём примере – World Class) и включаем инструменты разработчика.

Теперь выполняем простое действие – например, на сайте переключаем неделю расписания. В этот момент браузер что-то подгружает, страница не перезагружается, а данные вдруг обновляются – значит, сайт обратился к серверу за свежим расписанием.

Нам нужно найти этот момент.

Смотрим на список запросов во вкладке *Network* и отфильтровываем их по типу **XHR** или **Fetch**. Среди десятков строк с картинками и стилями видим что-то вроде:

```
POST https://my.worldclass.ru/api/v1/clubs/scheduling
```

Это и есть наш "секретный вход", внутренняя конечная точка API, через которую сайт получает данные, показан.

Если кликнуть по строке запроса, откроется панель с подробностями:

- во вкладке **Headers** – адрес запроса и параметры,
- во вкладке **Payload** – содержимое тела запроса (то, что отправляется на сервер),

- во вкладке **Response** – ответ сервера, чаще всего в формате JSON.

Мы видим структуру вроде:

```
{
  "gymList": ["d35ba5fc-1f7e-11ec-b664-50e548298f06"],
  "startDate": "2025-09-02T00:00:00",
  "endDate": "2025-09-09T00:00:00",
  "chain": 1,
  "token": "70a69ca6-XXXX-XXXX-a975-005056b1372d"
}
```

Именно так сайт узнаёт, какие тренировки идут на неделе. А нам остаётся только научиться повторять этот запрос своими силами.

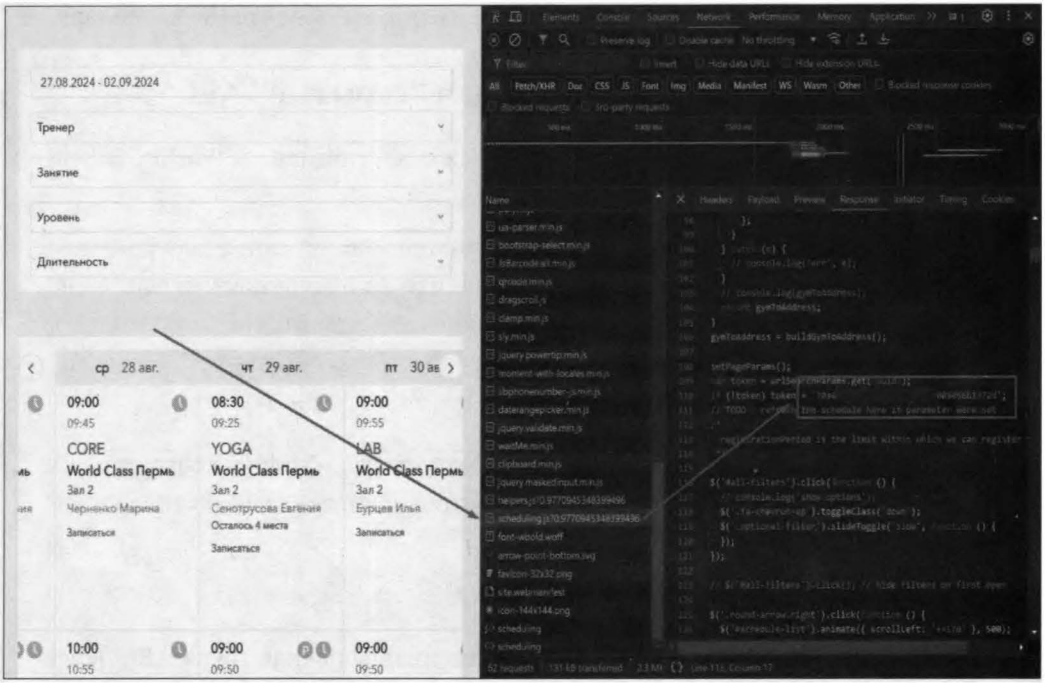


Рис. 8.2. ***Файл, где можно посмотреть токен, который понадобится для обращения к API***

Шаг 2. Разбор полётов: что значат эти поля.

Мы не просто скопируем запрос – давайте разберёмся, что здесь происходит.

- **gymList** – список идентификаторов клубов. Каждый клуб сети имеет свой уникальный UUID, и именно по нему сервер понимает, из какого города и филиала запрашиваются данные.

  В нашем случае: "d35ba5fc-1f7e-11ec-b664-50e548298f06" – это World Class Пермь.

- **startDate** и **endDate** – границы недели, за которую мы хотим получить расписание. Указываются в формате ISO-даты (YYYY-MM-DDTHH:MM:SS). Мы можем легко генерировать эти значения прямо в скрипте.

- **chain** – внутренний идентификатор.

- **token** – ключ доступа, который сайт передаёт сам себе. Он не является секретным – токен подставляется автоматически при загрузке страницы и хранится в открытом виде. Нам нужно лишь скопировать его из вкладки *Network* и вставить в наш скрипт.

Если мы сформируем точно такой же запрос, сервер любезно вернёт нам расписание в том же виде, как и для сайта. Осталось научиться делать это автоматически.

Шаг 3. Пишем решение: Google Apps Script в роли API-клиента

Мы могли бы использовать Python, но давайте пойдём другим путём – воспользуемся Google Apps Script, который встроен в Google Drive и умеет работать напрямую с нашим Google Календарём.

Потому что всё выполняется прямо "в облаке", без установки зависимостей, и у нас сразу есть доступ к сервисам Google. А значит, один скрипт сможет и получить данные, и добавить их в календарь.

Открываем Google Drive, создаём новый проект → "Ещё" → "Скрипт Apps Script". Появляется окно редактора. Удаляем шаблонную функцию myFunction() и вставляем код:

```
function WorldclassAPISchedulingPerm() {
  const url = 'https://my.worldclass.ru/api/v1/clubs/scheduling';
  const currentDate = new Date();
  const endDate = new Date(currentDate);
  endDate.setDate(currentDate.getDate() + 7);
```

```
  const payload = {
    "gymList": ["d35ba5fc-1f7e-11ec-b664-50e548298f06"],
    "startDate": currentDate.toISOString().split('.')[0],
    "endDate": endDate.toISOString().split('.')[0],
    "chain": 1,
    "token": "70a69ca6-XXXX-XXXX-a975-005056b1372d"
  };

  const options = {
    method: 'post',
    contentType: 'application/json',
    payload: JSON.stringify(payload),
    headers: {
      'brand': 'WorldClass',
      'language': 'Ru',
      'chain': '1',
      'Accept': '*/*',
      'Origin': 'https://my.worldclass.ru',
      'Referer': 'https://my.worldclass.ru/scheduling?clubs='
    }
  };

  const response = UrlFetchApp.fetch(url, options);
  const data = JSON.parse(response.getContentText()).data;

  const filtered = data.filter(item =>
    item.service.name === "ICG Color Cycle (сайкл)" || item.service.name
=== "CORE"
  );

  filtered.forEach(item => {
    const start = new Date(item.startDate);
    const end = new Date(start);
    end.setHours(start.getHours() + 1);

    CalendarApp.getDefaultCalendar().createEvent(
```

```
      '${item.service.name} - ${item.employee.firstName} ${item.employee.
lastName}',
      start,
      end,
      {
        description: item.service.shortDescription,
        location: 'Фитнес-клуб World Class, Пермь'
      }
    );
  });
}
```

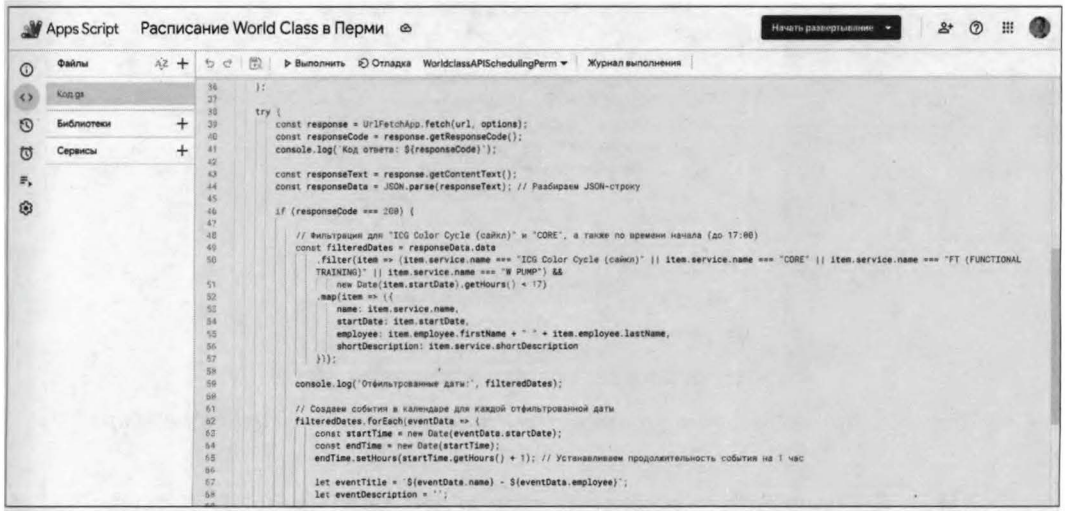


*Рис. 8.3. **Редактор Google Apps Script с вставленным кодом скрипта***

Разбираем скрипт:

- **payload** – наш "заказ для официанта": мы говорим API, какой клуб и за какой период хотим получить.
- **UrlFetchApp.fetch()** – отправляем этот заказ на сервер и получаем ответ.
- **JSON.parse()** – разбираем полученное "блюдо": сервер вернул JSON-структуру, где каждая запись – отдельная тренировка.
- **filter()** – выбираем только нужные занятия, например, ICG Color Cycle и CORE.

- **CalendarApp.createEvent()** – создаём события в Google Календаре, чтобы они появились на нужные даты и время.

Результат – календарь сам наполняется актуальными тренировками на неделю вперед.

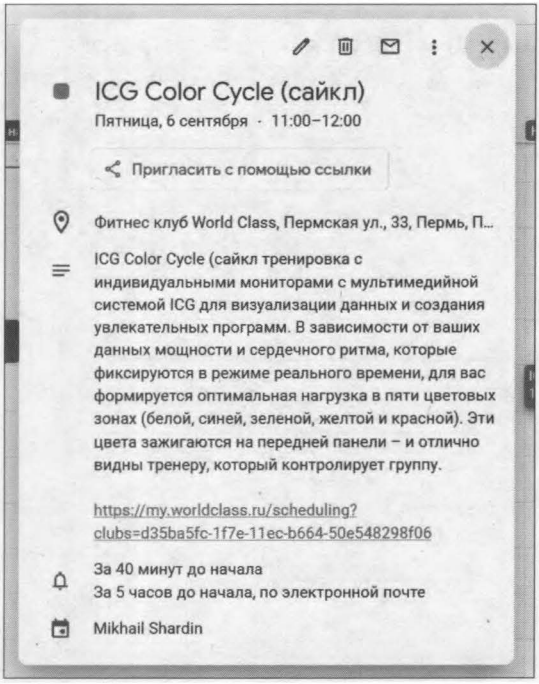


Рис. 8.4. ***Пример автоматически созданных событий в Google Календаре***

Шаг 4. Проверяем и настраиваем автоматический запуск.

- Сохраняем скрипт (Ctrl + S) и запускаем ▶.
- При первом запуске Google попросит разрешения: подтвердите доступ к Календарю и сетевым запросам. После этого – всё заработает.
- Чтобы скрипт выполнялся сам, без нашего участия, идём в раздел **Триггеры** (иконка часов) → "Добавить триггер" → выбираем:
  - » функцию: WorldclassAPISchedulingPerm
  - » расписание: раз в неделю (например, понедельник в 02:00 ночи)

Теперь каждую неделю скрипт сам обращается к API, получает свежие данные и обновляет наш календарь.

Всё, что нам остаётся – просто смотреть, когда ближайшая тренировка.

Этика и границы допустимого

Важно отметить: мы не взламываем систему и не обходим пароли. Мы просто используем **открытые данные**, которые сам сайт свободно передаёт пользователю. Это похоже на то, как сфотографировать расписание автобусов на остановке – мы не нарушаем ничего, просто фиксируем общедоступную информацию для собственного удобства.

8.3. API для данных: создаем единый центр для личной аналитики

А ещё наши данные живут везде – кроме того места, где они нам действительно нужны, например, если мы ведём блог, пишем статьи, работаем с аудиторией, анализируем свои спортивные достижения. И каждый раз сталкиваемся с одним и тем же: данные разбросаны по разным платформам, форматам и интерфейсам, а собрать их воедино часто сложно.

Но ведь именно в данных рождается понимание: что "выстреливает", что работает, куда двигаться дальше. И мы хотим вернуть себе эту власть. В этом разделе мы научимся строить свою "командную панель" – место, где все наши метрики собираются автоматически.

Кому может пригодиться свой личный центр сбора данных?

- **Если мы ведём блог**, нам важны просмотры, комментарии, рейтинг статей, динамика публикаций, вовлечённость аудитории. Платформы дают статистику, но у каждой свой интерфейс, свои ограничения, и сравнивать всё это неудобно.
- **Если мы аналитики или маркетологи**, мы можем собирать упоминания бренда, отзывы или скорость реакции аудитории.
- **Если мы коллекционеры**, мы можем отслеживать лоты на нескольких аукционных площадках.
- **Если мы SMM-специалисты**, мы можем собирать данные о постах из разных соцсетей: охваты, лайки, комментарии, сохранения.

Общий паттерн прост: информация находится на чужих площадках, а мы хотим владеть её полным "срезом". И API здесь работает как наш универсальный экскаватор: это позволяет аккуратно и регулярно считывать нужные данные – без ручной рутины.

Информационная раздробленность

Хочешь посмотреть статистику статей – открываешь Хабр, Т–Ж, ещё несколько платформ. Хочешь проанализировать динамику – нужно вручную переносить цифры просмотров в таблицу. Хочешь составить рейтинг – снова копипаст.

Через неделю всё забывается, через месяц перестаёшь вести статистику, а через год понимаешь, что потерял огромный пласт информации.

Мы эту проблему решим. Создадим свой "центр управления полётами", где данные будут появляться автоматически – как по волшебству.

Создаём одну Google Таблицу, которая становится нашим личным аналитическим хабом. В ней будут строки со статьями, постами, товарами, достижениями – чем угодно. В соседних колонках – просмотры, комментарии, цены, рейтинги, любые показатели.

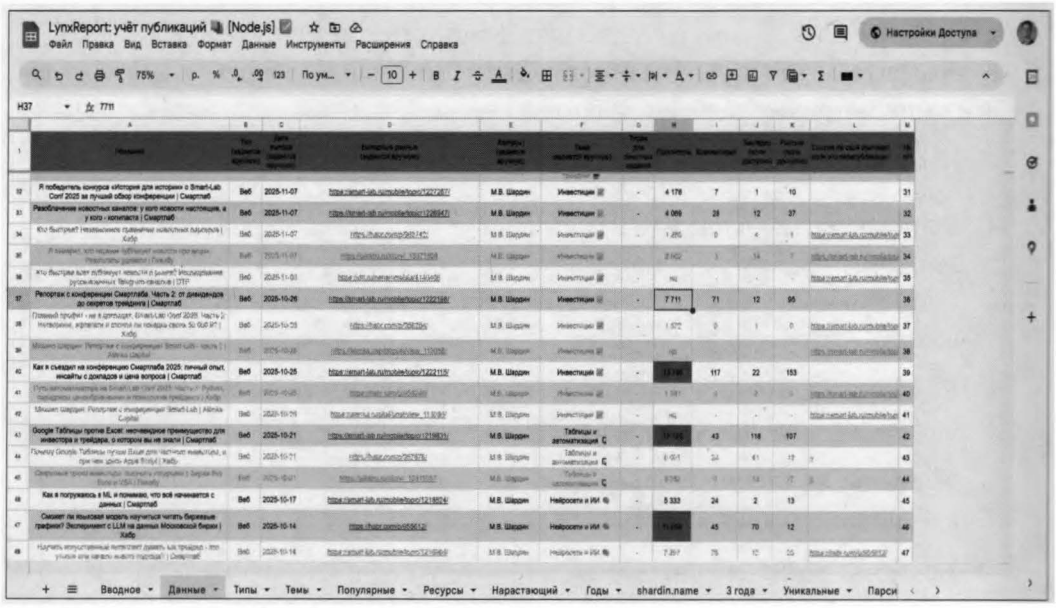


Рис. 8.5. ***Результаты парсинга в Google Таблице***

И всё это будет обновляться автоматически с помощью Google Apps Script.

Схема выглядит так:

```
Сайт → API или HTML → Скрипт Google Apps Script → Google Таблица → Готовая аналитика
```

Парсинг HTML на примере сайта Хабр

Если API нет или оно недоступно, то приходится брать данные непосредственно со страницы – то есть парсить HTML. Мы загружаем страницу:

```
var html = UrlFetchApp.fetch(url).getContentText();
```

HTML – это просто текст. И если мы знаем, где расположена нужная нам цифра, мы можем её извлечь. Например, количество просмотров на Хабре часто записано так:

```
<span class="tm-icon-counter__value">3.4K</span>
```

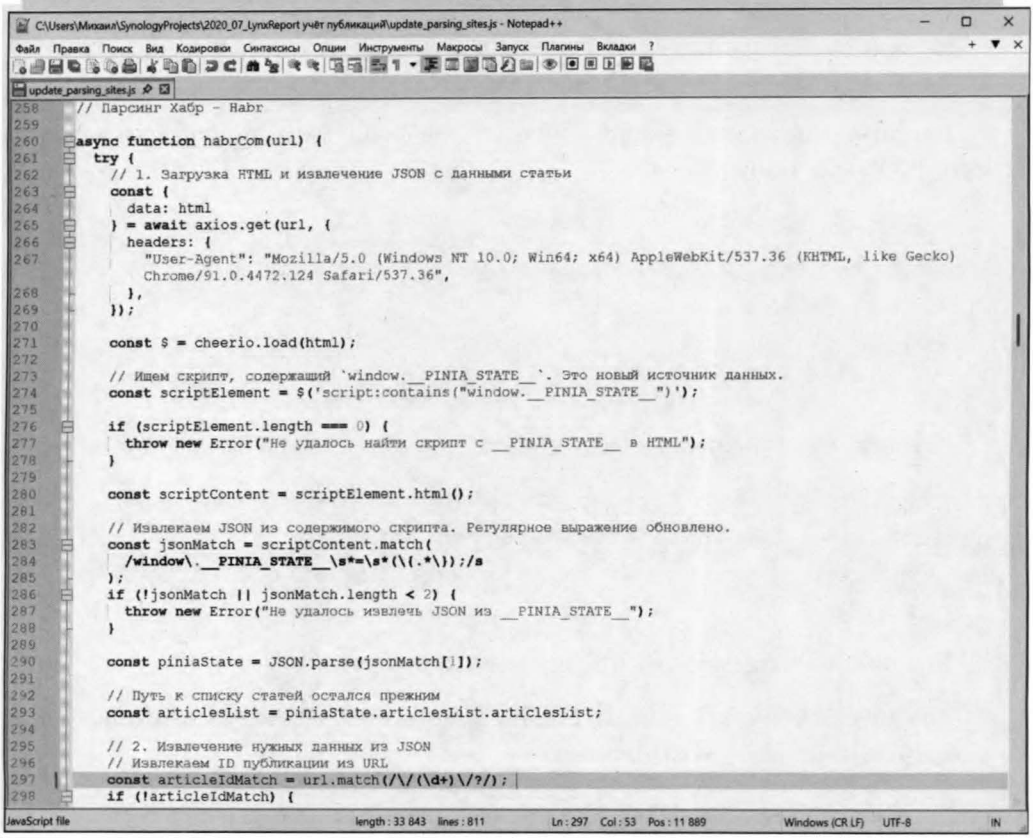


*Рис. 8.6. **Функция парсинга habrCom(), написанная на Node.js***

Мы ищем этот кусочек через регулярное выражение:

```
let Views = +html.match(/<span class="tm-icon-counter__value">
(.*?)K<\/span>/)[1]
    .replace(/\,/g, '.') * 1000
```

Мы выдернули цифру, превратили "3.4K" в число 3400 – и записали в таблицу.

То же самое можно сделать с комментариями, закладками, рейтингом.

Работа со "скрытым" API

Парсинг HTML работает, но он хрупкий: стоит сайту поменять верстку – и всё ломается. Поэтому мы всегда ищем более надёжный путь – скрытый API.

Многие сайты подгружают данные в фоновом режиме, и если открыть вкладку Network → Fetch/XHR, мы увидим структуру запросов.

Например, некоторые медиа-сайты возвращают статистику статей в формате JSON, но это редкость:

```
{
  "views": 5234,
  "favorites": 121,
  "comments": 47
}
```

И тогда код становится в разы проще:

```
const response = UrlFetchApp.fetch(apiUrl);
const data = JSON.parse(response.getContentText());
return [data.views, data.comments, data.favorites];
```

Мы просто читаем JSON и забираем нужные поля.

Так можно собирать данные с Т–Ж, Smart-Lab, форумов, маркетплейсов, каталогов и многих других сайтов.

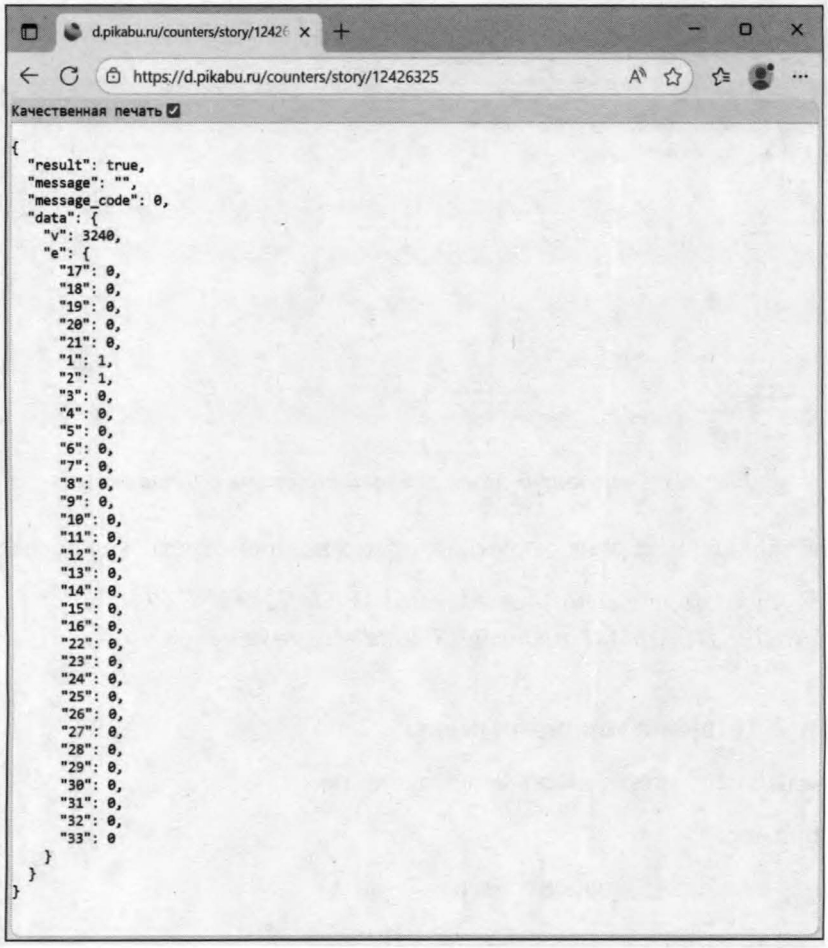


Рис. 8.7. ***Пример ответа скрытого API***

Строим свою единую панель данных

Шаг 1. Создаем таблицу.

Мы открываем Google Таблицы и создаём новую страницу.

В первой колонке – ссылки на наши статьи или сайты.

В следующих – колонки для просмотров, комментариев, рейтингов.

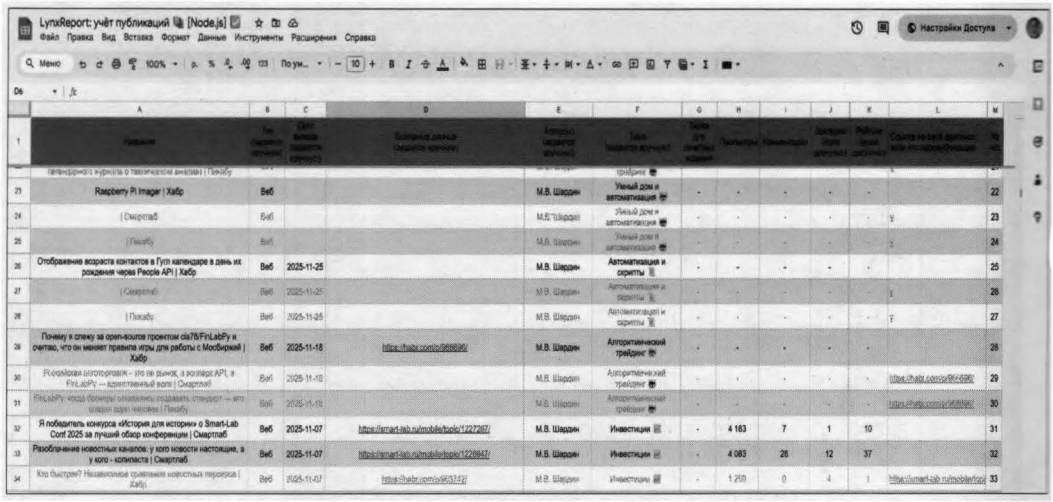


Рис. 8.8. ***Структура моей рабочей таблицы с метриками***

Моя таблица кстати не секретная, её можно посмотреть, как образец:

https://docs.google.com/spreadsheets/d/1V0dgN2KI3HEA9sn-u1LM6JRKDGXtbMaLu_DjymuhWds/edit?usp=sharing

Шаг 2. Пишем функции-парсеры.

Каждый сайт может иметь свою функцию.

Например:

- `habr_com(url)` – парсит Хабр
- `tg_stat(url)` – достаёт данные с *tgstat.ru*
- `smartlab(url)` – получает статистику публикации через скрытый *endpoint*
- `wishlist(url)` – парсит цену товара

Мы создаём набор функций – маленькую библиотеку, которую можем расширять.

Шаг 3. Организуем автоматическое обновление.

В Google Apps Script создаём функцию:

```
function updateAll() {
```

```
  const sheet = SpreadsheetApp.getActive().getSheetByName('Data');
  const rows = sheet.getRange(2, 1, sheet.getLastRow()-1).
getValues();

  rows.forEach((row, i) => {
    const url = row[0];
    const host = url.match(/https?:\/\/([^\/]+)/)[1];

    if (host.includes('habr.com')) {
      const result = habr_com(url);
      sheet.getRange(i+2, 2).setValue(result[0]); // views
      sheet.getRange(i+2, 3).setValue(result[1]); // comments
    }
    // ...другие сайты
  });
}
```

И привязываем триггер "раз в сутки".

После нескольких лет использования таблицы, мне пришлось перейти с Google Apps Script на Node.js, потому что у меня скопилось несколько сотен статей, и я просто не укладывался в бесплатный лимит серверного времени. К тому же некоторые сайты активно защищаются и для сбора данных с них требуется безголовый браузер.

Вот пример habrCom(), написанная на Node.js:

```
// Парсинг Хабр - Habr

async function habrCom(url) {
  try {
    // 1. Загрузка HTML и извлечение JSON с данными статьи
    const {
      data: html
    } = await axios.get(url, {
      headers: {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36",
      },
    });
```

```
    const $ = cheerio.load(html);

    // Ищем скрипт, содержащий 'window.__PINIA_STATE__'. Это новый источник
данных.
    const scriptElement = $('script:contains("window.__PINIA_STATE__")');

    if (scriptElement.length === 0) {
      throw new Error("Не удалось найти скрипт с __PINIA_STATE__ в HTML");
    }

    const scriptContent = scriptElement.html();

    // Извлекаем JSON из содержимого скрипта. Регулярное выражение обновлено.
    const jsonMatch = scriptContent.match(
      /window\.__PINIA_STATE__\s*=\s*(\{.*\});/s
    );
    if (!jsonMatch || jsonMatch.length < 2) {
      throw new Error("Не удалось извлечь JSON из __PINIA_STATE__");
    }

    const piniaState = JSON.parse(jsonMatch[1]);

    // Путь к списку статей остался прежним
    const articlesList = piniaState.articlesList.articlesList;

    // 2. Извлечение нужных данных из JSON
    // Извлекаем ID публикации из URL
    const articleIdMatch = url.match(/\/(\d+)\/?/);
    if (!articleIdMatch) {
      throw new Error('Не удалось извлечь ID статьи из URL: ${url}');
    }
    const articleId = articleIdMatch[1];

    // Достаем нужную статью по ID
    const article = articlesList[articleId];

    if (!article) {
      throw new Error(
        'Статья с ID ${articleId} не найдена в объекте __PINIA_STATE__.'
```

```
      );
    }

    // Поля в объекте статистики остались теми же, что очень удобно
    const Views = article.statistics.readingCount;
    const Comments = article.statistics.commentsCount;
    const Bookmarks = article.statistics.favoritesCount;
    const Ratings = article.statistics.score;

    // 3. Логирование и возврат результата
    console.log(
      'Для ${url}:\nПросмотры = ${Views} \nКомментарии = ${Comments} \
nЗакладки = ${Bookmarks} \nРейтинг = ${Ratings}.'
    );
    return '${Views}|${Comments}|${Bookmarks}|${Ratings}';
  } catch (error) {
    // Обработка ошибок
    console.error('Ошибка чтения данных для ${url}: ${error.message}');
    return '?|?|?|?';
  }
}
```

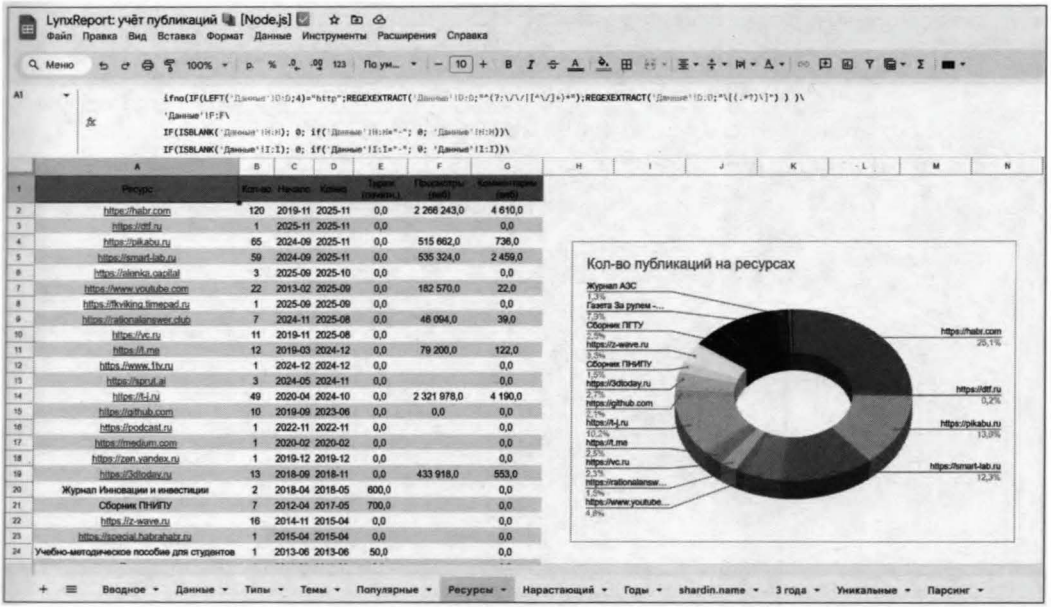


Рис. 8.9. ***Автоматическое выделение ресурсов и статистики***

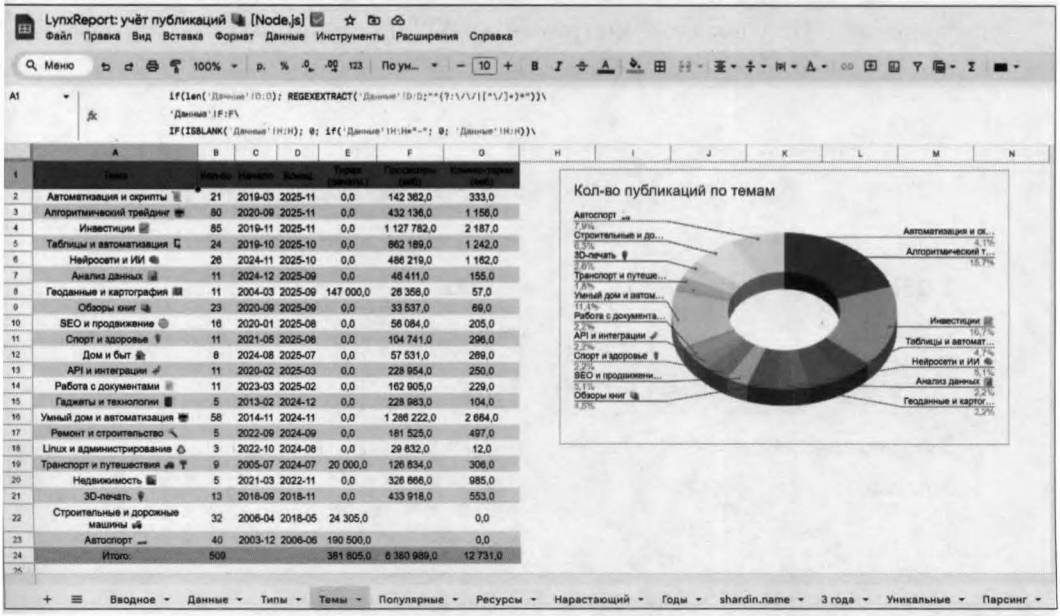
Шаг 4. Создаём сводную аналитику на основе QUERY.

Теперь самое приятное – мы начинаем анализировать данные.

Формула QUERY позволяет строить выборки:

```
= IFERROR(
  {
    QUERY(
        ARRAYFORMULA({
            ifna(IF(LEFT('Данные'!D:D;4)="http";REGEXEXTRACT('Данные'
!D:D;"^(?:\/\/|[^\/]+)*");REGEXEXTRACT('Данные'!D:D;"\[(.*?)\]") ) )\
            'Данные'!F:F\
            IF(ISBLANK('Данные'!H:H); 0; if('Данные'!H:H="-"; 0;
'Данные'!H:H))\
            IF(ISBLANK('Данные'!I:I); 0; if('Данные'!I:I="-"; 0;
'Данные'!I:I))\
            'Данные'!C:C\
            if('Данные'!G:G="-"; 0; 'Данные'!G:G)
        });
        "select Col1, Count(Col1), min(Col5),max(Col5), sum(Col6),
sum(Col3), sum(Col4) where NOT Col1=''
        group by Col1 Order by max(Col5) desc
        label Col1 'Ресурс', Count(Col1) 'Кол-во', sum(Col6) 'Тираж
(печатн.)', sum(Col3) 'Просмотры (веб)', sum(Col4) 'Комментарии
(веб)', min(Col5) 'Начало', max(Col5) 'Конец'
        format sum(Col3) '#,##0', sum(Col4) '#,##0', min(Col5) 'YYYY-
MM', max(Col5) 'YYYY-MM'
        ") ; {
            "Итого:"\
            COUNTA('Данные'!F:F)\
            ""\
            ""\
            SUM('Данные'!G:G)\
            SUM('Данные'!H:H)\
            SUM('Данные'!I:I)
        }
    }
; "Дождитесь окончания загрузки...")
```

Так я получаю статистику всех ресурсов, можно строить динамику по датам, графики, отчёты по автообновлению – всё что угодно.



	Тема	Кол-во	Начало	Конец	Гонорар (рублей)	Просмотры (всего)	Комментарии (всего)
2	Автоматизация и скрипты	21	2019-03	2025-11	0,0	142 362,0	333,0
3	Алгоритмический трейдинг	80	2020-09	2025-11	0,0	432 136,0	1 156,0
4	Инвестиции	85	2019-11	2025-11	0,0	1 127 782,0	2 187,0
5	Таблицы и автоматизация	24	2019-10	2025-10	0,0	862 189,0	1 242,0
6	Нейросети и ИИ	26	2024-11	2025-10	0,0	486 219,0	1 162,0
7	Анализ данных	11	2024-12	2025-09	0,0	46 411,0	155,0
8	Геоданные и картография	11	2004-03	2025-09	147 000,0	26 358,0	57,0
9	Обзоры книг	23	2020-09	2025-09	0,0	33 537,0	69,0
10	SEO и продвижение	16	2020-01	2025-08	0,0	56 084,0	205,0
11	Спорт и здоровье	11	2021-05	2025-08	0,0	104 741,0	296,0
12	Дом и быт	8	2024-08	2025-07	0,0	57 531,0	269,0
13	API и интеграции	11	2020-02	2025-03	0,0	228 954,0	250,0
14	Работа с документами	11	2023-03	2025-02	0,0	162 905,0	229,0
15	Гаджеты и технологии	5	2013-02	2024-12	0,0	228 963,0	104,0
16	Умный дом и автоматизация	58	2014-11	2024-11	0,0	1 266 222,0	2 664,0
17	Ремонт и строительство	5	2022-09	2024-09	0,0	181 525,0	497,0
18	Linux и администрирование	3	2022-10	2024-08	0,0	29 832,0	12,0
19	Транспорт и путешествия	9	2005-07	2024-07	20 000,0	126 834,0	306,0
20	Недвижимость	5	2021-03	2022-11	0,0	326 866,0	985,0
21	3D-печать	13	2018-09	2018-11	0,0	433 918,0	553,0
22	Строительные и дорожные машины	32	2006-04	2018-05	24 305,0		0,0
23	Автоспорт	40	2003-12	2006-06	190 500,0		0,0
24	Итого:	509			381 805,0	6 380 989,0	12 731,0

Рис. 8.10. ***Статистика тем моих публикаций***

Данные работают на нас

Мы построили собственный центр сбора данных – простой, облачный, автономный. Каждый день он поднимает нужную нам статистику, обновляет цифры, строит рейтинги и помогает нам принимать решения:

- какие темы заходят лучше;
- как меняется интерес к нашим проектам;
- что стоит развивать дальше.

8.4. API для контроля: наводим порядок в своих цифровых сообществах

Представим, что мы ведём Telegram-канал или любое другое онлайн-сообщество. Сначала всё гармонично: доброжелательные люди, обсуждения

по теме, нужные посты. Со временем, в канал могут "налить" ботов спамеры в случае конфликтных ситуаций. Ручная чистка в таких случаях отнимает уйму времени. Но у нас есть инструменты: API и немного кода.



Рис. 8.11. ***Статистика телеграм канала***

Проблема: цифровой мусор и его последствия

"Мёртвые" подписчики не приносят ценности: они снижают ER (коэффициент вовлеченности аудитории), портят метрики для рекламодателей и мешают анализу аудитории. Для нас – это не только цифры, но и риск репутационных потерь: рекламодатель видит большое количество подписчиков, но маленькую отдачу – и отказывается от сотрудничества. Поэтому раз в какой-то момент мы должны привести аудиторию в порядок.

Обычные боты Telegram имеют ограниченные права: они не могут действовать от имени пользователя и редко имеют доступ к полной информации

о подписчиках канала. Для задач модерации (удаление людей, массовый аудит) нужен user-бот – скрипт, который работает от имени нашего аккаунта и использует Telegram API, а не Bot API. Для этого идеально подходит Telethon – лёгкая библиотека Python для асинхронной работы с Telegram API.

Преимущества Telethon:

- Работает как юзер (через api_id/api_hash).
- Позволяет получать участников, удалять их (при наличии прав администратора).
- Удобно обрабатывать большие объёмы данных и логировать действия.

Безопасность прежде всего

1. Получаем **api_id** и **api_hash** на *https://my.telegram.org* – это "доступ в API".
2. Храним их в секрете: используем файл *.env* или менеджер секретов. Никогда не кладём эти значения в публичные репозитории.
3. Всегда тестируем в "сухом режиме" – сначала просто сохраняем список и печатаем, кого бы удалили, а только после ручной проверки запускаем удаление.

Пример *.env*:

```
API_ID=1234567
API_HASH=abcd1234abcd5678efgh9012ijkl3456
PHONE=+79991234567
CHANNEL_USERNAME=@homeinv
```

Скрипт рабочего примера на Telethon

Ниже – компактный рабочий пример на Telethon.

Скрипт:

- подключается к аккаунту;
- получает до 200 последних подписчиков;
- фильтрует кандидатов по критериям;
- логирует в CSV;
- в dry-run режиме просто показывает список, в рабочем режиме – удаляет по одному.

```
import os
import csv
import time
import asyncio
import sys
import argparse
from datetime import datetime, timezone
from zoneinfo import ZoneInfo, ZoneInfoNotFoundError
from dotenv import load_dotenv

from telethon.sync import TelegramClient
from telethon.tl.functions.channels import GetParticipantsRequest

# Используем правильный фильтр для получения ПОСЛЕДНИХ
присоединившихся
from telethon.tl.types import ChannelParticipantsRecent,
ChannelParticipantsSearch
from telethon.errors.rpcerrorlist import UserNotParticipantError,
FloodWaitError

# --- ПРИНУДИТЕЛЬНАЯ КОДИРОВКА UTF-8 ДЛЯ ВЫВОДА ---
if sys.stdout.encoding != 'utf-8':
    sys.stdout.reconfigure(encoding='utf-8')
if sys.stderr.encoding != 'utf-8':
    sys.stderr.reconfigure(encoding='utf-8')

# --- КОНФИГУРАЦИЯ ---
load_dotenv()
API_ID = os.getenv('api_id')
API_HASH = os.getenv('api_hash')
PHONE_NUMBER = os.getenv('phone_number')
CHANNEL_USERNAME = 'homeinv'
current_date_str = datetime.now().strftime("%Y_%m_%d") # Получаем
текущую дату и форматируем ее в строку 'ГГГГ_мм_дд'
WHITELIST_CSV = f"{current_date_str}_subscribers_whitelist.csv" #
Собираем полное имя файла с помощью f-строки
KICKED_LOG_FILENAME = 'kicked_users.csv' # <--- ФАЙЛ ДЛЯ ЛОГОВ
```

```
KICK_DELAY_SECONDS = 3  # <--- Увеличил задержку для безопасности

# Создаем клиент
client = TelegramClient('homeinv_session', API_ID, API_HASH)


async def kick_by_date(channel_entity, date_str: str, force_
delete=False):
    """
    Основная функция для удаления подписчиков, присоединившихся ПОСЛЕ
указанной даты.
    Корректно обрабатывает время, указанное в Московском часовом поясе.
    """
    try:
        # Устанавливаем часовой пояс Москвы
        try:
            moscow_tz = ZoneInfo("Europe/Moscow")
        except ZoneInfoNotFoundError:
            print("[!] Ошибка: Часовой пояс 'Europe/Moscow' не найден.")
            print("[!] Для Windows может потребоваться установка
пакета tzdata: pip install tzdata")
            return

        # Парсим введенную строку как "наивное" время (без часового
пояса)
        naive_date = datetime.strptime(date_str, '%Y-%m-%d %H:%M:%S')

        # Присваиваем этому времени часовой пояс Москвы (локализуем)
        localized_date = naive_date.replace(tzinfo=moscow_tz)

        # Конвертируем московское время в UTC для сравнения с данными
от Telegram
        kick_after_date_utc = localized_date.astimezone(timezone.utc)

        print(f"[*] Целевая дата: все, кто подписался ПОСЛЕ {date_str}
(Moscow Time), будут удалены.")
        print(f"    (Это соответствует {kick_after_date_utc.
strftime('%Y-%m-%d %H:%M:%S UTC')})")
```

```
    except ValueError:
        print(f"[!] Ошибка: Неверный формат даты. Используйте 'YYYY-
MM-DD HH:MM:SS'.")
        return

    log_header = ['user_id', 'username', 'first_name', 'last_name',
'join_date', 'is_bot', 'access_hash']
    if not os.path.exists(KICKED_LOG_FILENAME):
        with open(KICKED_LOG_FILENAME, 'w', encoding='utf-8', newline='')
as f:
            writer = csv.DictWriter(f, fieldnames=log_header,
delimiter=';')
            writer.writeheader()

    total_kicked_count = 0
    offset = 0
    limit = 200

    while True:
        print(f"\n[*] Запрашиваю порцию самых новых подписчиков
(offset: {offset})...")
        try:
            result = await client(GetParticipantsRequest(
                channel=channel_entity,
                filter=ChannelParticipantsRecent(),
                offset=offset,
                limit=limit,
                hash=0
            ))
        except FloodWaitError as e:
            print(f"[!] FloodWaitError. Жду {e.seconds} сек...")
            await asyncio.sleep(e.seconds + 5)
            continue
        except Exception as e:
            print(f"[!] Критическая ошибка при получении участников: {e}")
            break
```

```
        if not result.users:
            print("[+] Больше нет участников для проверки. Завершаю
работу.")
            break

        users_map = {user.id: user for user in result.users}
        participants_to_kick = []
        stop_processing = False

        for p in result.participants:
            user = users_map.get(p.user_id)
            if not user or not hasattr(p, 'date'):
                continue

            # Удаляем тех, кто присоединился ПОСЛЕ указанной даты (их
дата больше целевой)
            if p.date > kick_after_date_utc:
                participants_to_kick.append((user, p.date))
                print(f"[DEBUG] Добавлен для удаления: {user.id}
(дата: {p.date.strftime('%Y-%m-%d %H:%M:%S UTC')})")
            else:
                # Если встретили пользователя, который присоединился
ДО или В целевую дату
                # print(f"[*] Найден участник ({user.id}),
присоединившийся до/в целевую дату ({p.date.strftime('%Y-%m-%d
%H:%M:%S UTC')}).")
                # print(f"[*] Сравнение: {p.date.strftime('%Y-%m-%d
%H:%M:%S UTC')} <= {kick_after_date_utc.strftime('%Y-%m-%d %H:%M:%S
UTC')}")

                # Если мы уже собрали пользователей для удаления,
продолжаем до тех пор,
                # пока не найдем достаточно старых пользователей
подряд
                if len(participants_to_kick) == 0:
                    stop_processing = True
                    break
```

```
        if not participants_to_kick:
            if stop_processing:
                print("[+] В этой порции нет подписчиков для удаления
(все старше целевой даты). Чистка завершена.")
                break
            else:
                # Переходим к следующей порции
                offset += limit
                continue

        print("-" * 50)
        print(f"[!] Найдено {len(participants_to_kick)} подписчиков
для удаления:")
        print(f"{'ID':<12} {'Username':<20} {'Full Name':<25} {'Join
Date (UTC)'}")
        print("-" * 80)

        for user, join_date in participants_to_kick:
            username = f"@{user.username}" if user.username else "N/A"
            full_name = f"{user.first_name or ''} {user.last_name or
''}".strip()
            print(f"{user.id:<12} {username:<20} {full_name:<25}
{join_date.strftime('%Y-%m-%d %H:%M:%S')}")
        print("-" * 50)

        if not force_delete:
            try:
                confirm = await asyncio.to_thread(
                    input, f"[?] Вы уверены, что хотите удалить
этих {len(participants_to_kick)} пользователей? (введите 'yes' для
подтверждения): "
                )
                if confirm.lower() != 'yes':
                    print("[*] Операция отменена пользователем.
Завершение работы.")
                    break
            except (KeyboardInterrupt, asyncio.CancelledError):
```

```
                print("\n[*] Операция прервана.")
                break

        kicked_in_batch = 0

        for i, (user, join_date) in enumerate(participants_to_kick):
            try:
                print(f"[*] Удаляю [{i+1}/{len(participants_to_
kick)}]: ID={user.id}, Username={user.username or 'N/A'}")
                await client.kick_participant(channel_entity, user)

                with open(KICKED_LOG_FILENAME, 'a', encoding='utf-8',
newline='') as f:
                    writer = csv.DictWriter(f, fieldnames=log_header,
delimiter=';')
                    writer.writerow({
                        'user_id': user.id,
                        'username': user.username or '',
                        'first_name': user.first_name or '',
                        'last_name': user.last_name or '',
                        'join_date': join_date.strftime('%Y-%m-%d
%H:%M:%S'),
                        'is_bot': user.bot,
                        'access_hash': user.access_hash
                    })

                kicked_in_batch += 1
                print(f"    -> Успешно. Пауза {KICK_DELAY_SECONDS} сек.")
                await asyncio.sleep(KICK_DELAY_SECONDS)

            except UserNotParticipantError:
                print("    -> Ошибка: Пользователь уже не является
участником.")
            except FloodWaitError as e:
                print(f"[!] Получен FloodWaitError. Жду {e.seconds}
секунд...")
                await asyncio.sleep(e.seconds + 5)
```

```
                print("[!] Прерываю текущую пачку из-за FloodWait.
Запустите скрипт снова через некоторое время.")
                stop_processing = True
                break
            except Exception as e:
                print(f"[!] Не удалось удалить пользователя {user.id}.
Ошибка: {e}")

        total_kicked_count += kicked_in_batch
        print(f"[+] В этой порции удалено {kicked_in_batch}
подписчиков.")

        if stop_processing:
            print("[+] Чистка завершена из-за FloodWait или других
ограничений.")
            break

        # Переходим к следующей порции
        offset += limit
        print("[*] Пауза перед запросом следующей порции...")
        await asyncio.sleep(5)

    print(f"\n[+] Всего удалено подписчиков: {total_kicked_count}.")
    print(f"[*] Лог удаленных пользователей сохранен в файл: {KICKED_
LOG_FILENAME}")

async def main():
    parser = argparse.ArgumentParser(description="Скрипт для
управления подписчиками Telegram канала.")
    subparsers = parser.add_subparsers(dest='command', required=True,
help='Доступные команды')

    subparsers.add_parser('save', help='Сохранить доступных
подписчиков в CSV (для каналов - неполный список).')
    subparsers.add_parser('info', help='Показать диагностическую
информацию о канале.')
```

```
    parser_kick = subparsers.add_parser('kickbydate', help='Удалить
подписчиков, присоединившихся ПОСЛЕ указанной даты.')
    parser_kick.add_argument('--after-date', type=str, required=True,
help="Целевая дата в формате 'YYYY-MM-DD HH:MM:SS'. Все, кто новее,
будут удалены.")
    parser_kick.add_argument('--yes', '-y', action='store_true',
help='Пропустить подтверждение и немедленно начать удаление.')

    args = parser.parse_args()

    await client.start(phone=PHONE_NUMBER)
    print("[+] Клиент успешно запущен.")

    try:
        channel = await client.get_entity(CHANNEL_USERNAME)
        print(f"[*] Работаю с каналом: {channel.title} (ID: {channel.
id})")

        if args.command == 'save':
            await save_subscribers_to_csv(channel)
        elif args.command == 'info':
            await get_channel_info(channel)
        elif args.command == 'kickbydate':
            await kick_by_date(channel, args.after_date, args.yes)

    except ValueError:
        print(f"[!] Канал '{CHANNEL_USERNAME}' не найден. Проверьте
правильность имени пользователя.")
    except Exception as e:
        import traceback
        print(f"[!] Произошла критическая ошибка: {e}")
        traceback.print_exc() # Печатаем полный traceback для отладки
    finally:
        await client.disconnect()
        print("[+] Клиент отключен.")
```

```
async def save_subscribers_to_csv(channel_entity):
    print(f"[*] Начинаю сохранение подписчиков канала '{CHANNEL_
USERNAME}' в файл {WHITELIST_CSV}...")
    print("[!] ВНИМАНИЕ: для каналов этот метод сохранит только
последних подписчиков из-за ограничений Telegram API.")

    csv_header = ['user_id', 'username', 'first_name', 'last_name',
'join_date', 'is_bot', 'access_hash']
    all_participants_data = []

    try:
        offset = 0
        limit = 200

        while True:
            print(f"[*] Запрашиваю порцию подписчиков (offset:
{offset})...")

            try:
                # Используем GetParticipantsRequest для получения
информации о дате присоединения
                result = await client(GetParticipantsRequest(
                    channel=channel_entity,
                    filter=ChannelParticipantsRecent(),
                    offset=offset,
                    limit=limit,
                    hash=0
                ))
            except FloodWaitError as e:
                print(f"[!] FloodWaitError. Жду {e.seconds} сек...")
                await asyncio.sleep(e.seconds + 5)
                continue
            except Exception as e:
                print(f"[!] Ошибка при получении участников: {e}")
                break
```

```
            if not result.users:
                print("[+] Больше нет участников для сохранения.")
                break

            # Создаем мапу пользователей
            users_map = {user.id: user for user in result.users}

            # Обрабатываем участников с информацией о дате
присоединения
            for participant in result.participants:
                user = users_map.get(participant.user_id)
                if not user:
                    continue

                # Получаем дату присоединения из объекта participant
                join_date_str = ""
                if hasattr(participant, 'date') and participant.date:
                    join_date_str = participant.date.strftime('%Y-%m-
%d %H:%M:%S')

                all_participants_data.append({
                    'user_id': user.id,
                    'username': user.username or '',
                    'first_name': user.first_name or '',
                    'last_name': user.last_name or '',
                    'join_date': join_date_str,
                    'is_bot': user.bot,
                    'access_hash': user.access_hash
                })

            print(f"[*] Собрано {len(all_participants_data)}
участников...")

            # Если получили меньше участников, чем лимит, значит это
последняя порция
            if len(result.users) < limit:
                print("[+] Достигнут конец списка участников.")
```

```
                break

            offset += limit

            # Небольшая пауза между запросами
            await asyncio.sleep(1)

    except Exception as e:
        print(f"[!] Критическая ошибка при получении участников: {e}")

    print(f"\n[+] Всего собрано {len(all_participants_data)}
подписчиков.")

    if not all_participants_data:
        print("[!] Не удалось получить данные участников. Проверьте
права доступа к каналу.")
        return

    try:
        with open(WHITELIST_CSV, 'w', encoding='utf-8', newline='') as f:
            writer = csv.DictWriter(f, fieldnames=csv_header,
delimiter=';')
            writer.writeheader()
            writer.writerows(all_participants_data)
        print(f"[+] Данные успешно сохранены в {WHITELIST_CSV}")

        # Показываем примеры первых нескольких записей для проверки
        print(f"\n[*] Примеры сохраненных данных (первые 3 записи):")
        for i, participant in enumerate(all_participants_data[:3]):
            username = f"@{participant['username']}" if
participant['username'] else "N/A"
            full_name = f"{participant['first_name']}
{participant['last_name']}".strip()
            print(f"  {participant['user_id']} | {username} | {full_
name} | {participant['join_date']}")

    except Exception as e:
```

```
        print(f"[!] Ошибка при записи в CSV: {e}")


async def get_channel_info(channel_entity):
    print(f"[*] Диагностика канала '{CHANNEL_USERNAME}'...")
    try:
        full_channel = await client.get_entity(channel_entity)
        print(f"[+] Название канала: {full_channel.title}")
        print(f"[+] ID канала: {full_channel.id}")
        print(f"[+] Тип: {'Канал' if full_channel.broadcast else
'Группа'}")
        print(f"[+] Участников (по API): {getattr(full_channel,
'participants_count', 'Неизвестно')}")
        from telethon.tl.functions.channels import
GetFullChannelRequest
        full_info = await client(GetFullChannelRequest(channel_
entity))
        print(f"[+] Могу ли я видеть участников: {full_info.full_chat.
can_view_participants}")
        print("\n[*] Тестирование методов получения участников
(limit=200):")

        for filter_name, filter_obj in [("Recent",
ChannelParticipantsRecent()), ("Search ''", ChannelParticipantsSearch(''))]:
            try:
                result = await client(GetParticipantsRequest(channel=c
hannel_entity, filter=filter_obj, offset=0, limit=200, hash=0))
                print(f"[+] GetParticipantsRequest с фильтром {filter_
name}: получено {len(result.participants)} участников.")
            except Exception as e:
                print(f"[!] GetParticipantsRequest с фильтром {filter_
name} не сработал: {e}")
    except Exception as e:
        print(f"[!] Ошибка при диагностике: {e}")

if __name__ == "__main__":
    asyncio.run(main())
```

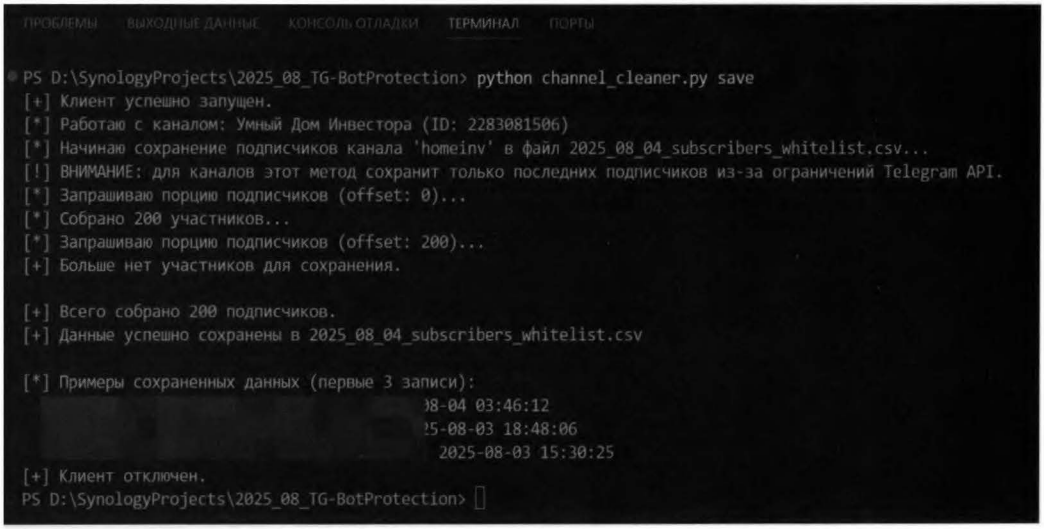


Рис. 8.12. ***Пример CSV с экспортированными подписчиками***

Разъяснения по коду и ограничения

- Telegram API действительно возвращает только последних ~200 участников через ChannelParticipantsRecent. Это ограничение. Скрипт полезен как аварийный инструмент, когда всплеск произошёл недавно.
- Чтобы работать со всеми участниками, требуется менять архитектуру (например, перевод канала в приватный – но это ломает *invite-links*). Поэтому комбинация "скрипт для разовой чистки + бот для отслеживания новых подписок в будущем" – оптимальный путь.
- Перед удалением мы обязательно предлагаем *dry-run* и ручную проверку: отображаем список и ждём подтверждения.

Отсылка к главе 3 – репутационные риски

Мы уже говорили о рисках автоматизации в главе 3: автоматизация это мощный инструмент, но требует уважения к людям и соблюдения правил. Автоматическое удаление подписчиков – это агрессивная операция: неправильные критерии могут убрать реальных людей. Всегда:

- четко формулируем политику канала;
- тестируем алгоритмы на небольших выборках;

- храним логи и возможность объяснить, почему был удалён тот или иной аккаунт.

Мы научились не только читать API и собирать данные – теперь мы умеем отправлять команды, которые изменяют состояние сервиса. Это высший уровень автоматизации: управление сообществом напрямую из нашего кода. Но с большой властью приходит и большая ответственность – автоматизация должна дополнять здравый смысл, а не заменять его.

Резюме главы: Вы – дирижер цифрового оркестра

Изучив эту главу, вы дополнили свои знания в области практического применения самого мощного инструмента автоматизации – работы с API.

Теперь вы не зависите только от того функционала, который вложили разработчики в любое приложение, потому что вы научились быть не просто пользователем, а цифровым дирижёром, который управляет сразу множеством инструментов, извлекая из каждого только нужную информацию.

Мы смогли вместе:

- Разобраться с тем, что такое API, представив его как посредника между вашими запросами и внутренним механизмом сервиса.
- Научились находить "скрытые" API, используя лишь встроенные инструменты браузера.
- А ещё мы поучились на практических примерах: нашли скрытый API большой сети спортивных клубов, смогли найти данные для составления личной статистики публикаций, хотя некоторые сайты сопротивлялись и даже научились прибираться за ботами внутри собственного Телеграмм канала.
- Ну и конечно обсудили вопросы этики и безопасности при работе с API, помня, что автоматизация всё же должна быть разумной.

В следующей главе мы сделаем первые шаги в мире алгоритмического тестирования и узнаем, как проверять инвестиционные гипотезы и не рисковать при этом реальными деньгами.

Глава 9.

Первый шаг к алгоритмам: тестируем инвестиционную идею

Каждый человек, который хотя бы раз смотрел на график котировок, знаком с чувством того, что всё кажется очевидным: "вот здесь надо было покупать внизу, а вот тут, на вершине, продавать".

И задним числом мы все – гениальные трейдеры. Но только кажется, что мы легко все определяем – закройте листочком часть графика и предположите, что там? Это окажется гораздо сложнее.

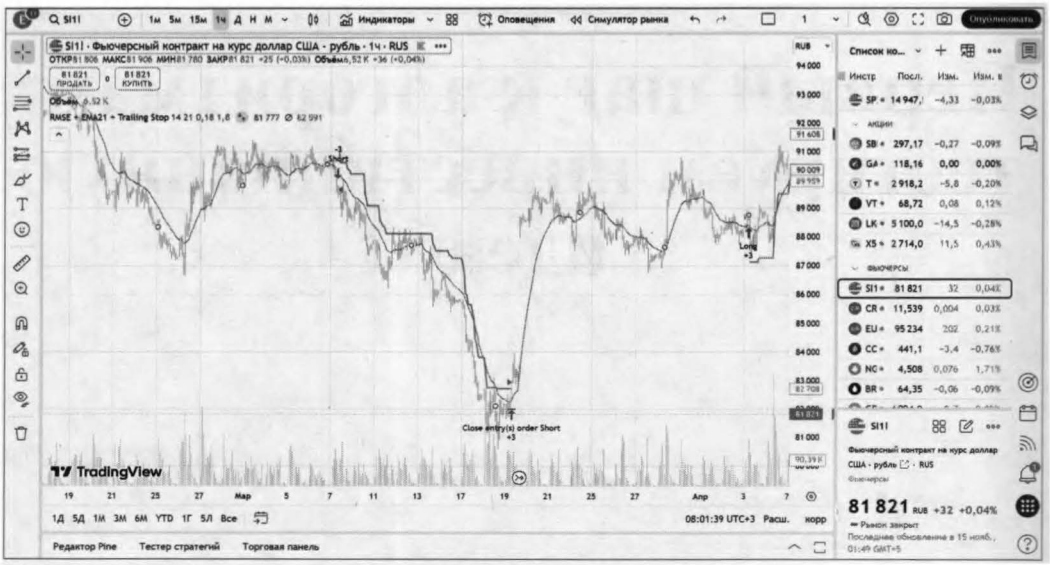


Рис. 9.1. ***Иллюзия идеальных сделок***

Чтобы не играть в угадайку, сделаем шаг в сторону алгоритмического подхода. Но мы не будем строить роботов для высокочастотной торговли или погружаться в дебри высшей математики.

Займёмся бэктестингом (это тестирование на исторических данных) ведь кажется это единственный способ превратить смутное "а что, если?.." в твердый факт. Это как фильтр, отделяющий ваши деньги от ваших заблуждений.

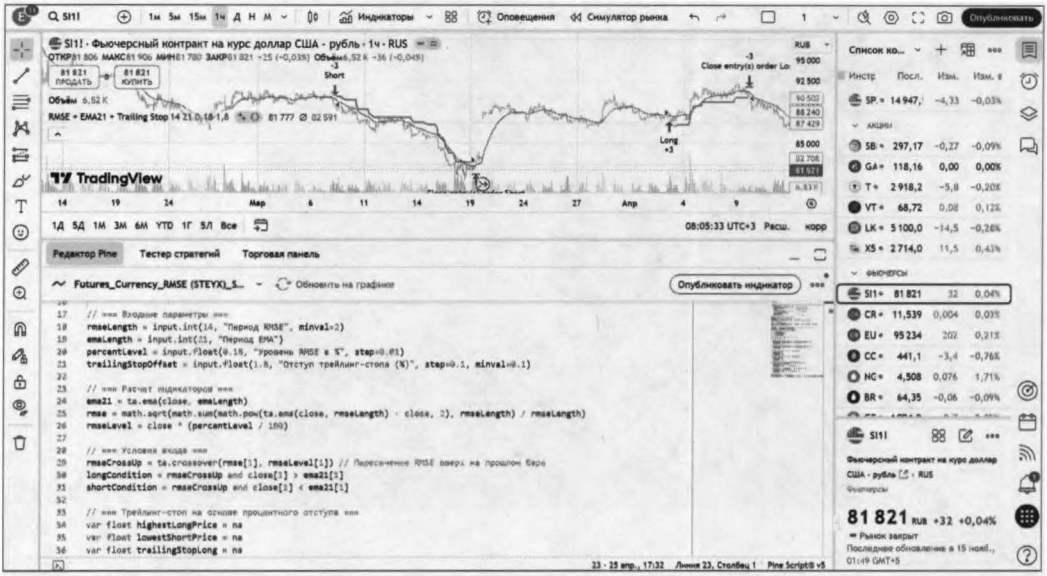


Рис. 9.2. ***Быстрая визуальная проверка идеи в TradingView через Pine Script***

Разделим весь путь в этой главе на два больших этапа:

- "Эскиз на салфетке", где познакомимся с TradingView и их скриптовым языком Pine Script, который позволяет очень быстро и достаточно просто визуально проверить любую идею.
- "Инженерный проект", где мы перейдем к Python. Будем использовать библиотеку Backtrader, которая позволяет тестировать не один актив, а целый портфель, учитывать комиссии, строить тепловые карты оптимизации и анализировать риск.

В конце главы мы проведем интересный эксперимент с машинным обучением – будем попытаемся научить нейросеть предсказывать движение цен, читая график не как набор цифр, а как текстовые описания.

Хотя эта глава однозначно не даст вам "грааль" или формулу вечного богатства, она даст вам набор инструментов, которые можно использовать для того чтобы не наделать ошибок.

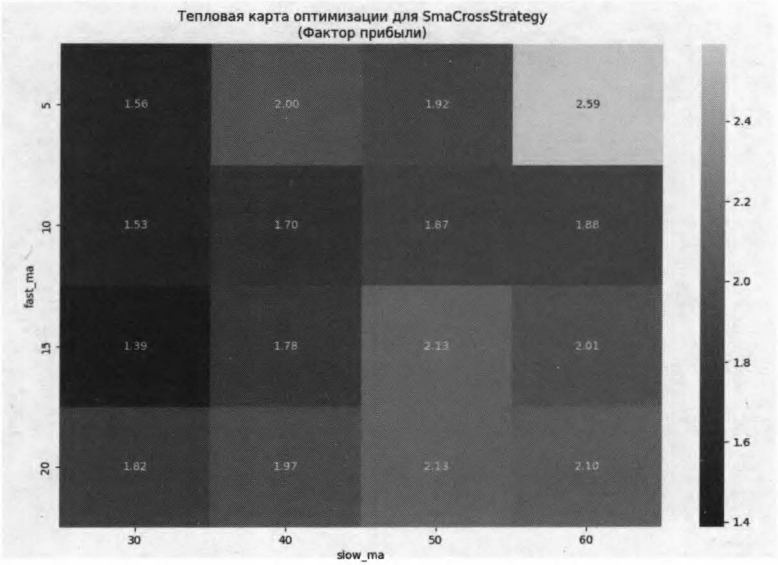


Рис. 9.3. ***Тепловая карта оптимизации в Python***

9.1. Практикум №1: "Эскиз на салфетке". Тестируем идею за 15 минут в TradingView

В практикуме мы повторим реальный путь, который проходит трейдер, когда впервые сталкивается с проверкой идей: возьмём простую концепцию из статьи, напишем код на Pine Script, нажмём "Добавить на график" и увидим, что происходит на исторических данных.

Идея – не секретная, была опубликована в журнале *STOCKS & COMMODITIES* почти двадцать лет назад, но до сих пор служит отличным примером того, как можно превратить интуицию о тренде в формальную проверяемую гипотезу.

Барбара Стар в статье "Confirming Price Trend" в декабре 2007 года

https://traders.com/documentation/feedbk_docs/2007/12/Abstracts_new/Star/star.html

предложила мысль: мы входим в позицию только тогда, когда видим нечто большее, чем случайное колебание – настоящий направленный тренд. Чтобы фильтровать шум, она использовала два инструмента:

1. **Линейную регрессию** – прямая линия, проведённая через цены. Её наклон показывает направление: вверх или вниз.
2. **Коэффициент детерминации (R^2)** – показатель того, насколько хорошо цены "держатся" вдоль этой линии.

   Если R^2 низкий → рынок хаотичен.

   Если высокий → движение устойчивое.

Мы можем представить это проще. Если мы едем по шоссе, линейная регрессия – это сама дорога, а R^2 – это то, насколько уверенно машина идёт по полосе и не виляет из стороны в сторону. Чем больше "сигнал", тем надёжнее направление.

Идея Барбары Стар говорит нам: не надо пытаться ловить каждый чих рынка – действуем только тогда, когда дорога ровная, а машина идёт стабильно. Для нас это отличная отправная точка: логично, понятно и легко реализуемо.

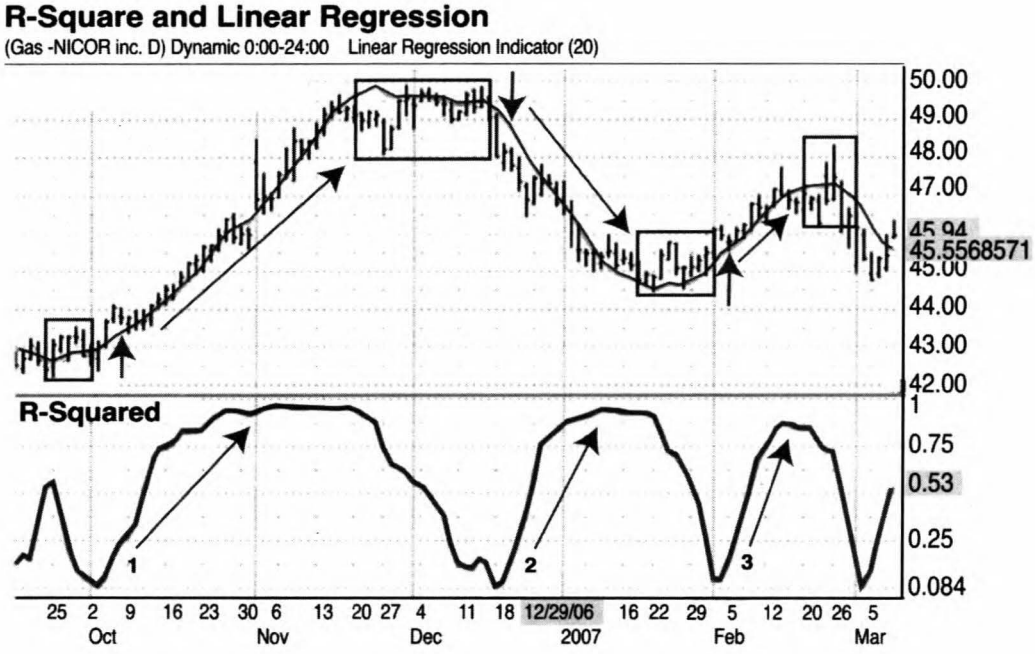


*Рис. 9.4. **Логика идеи: наклон регрессии и рост R^2 как подтверждение тренда***

Реализация на Pine Script

Теперь – самое интересное. Мы переносим идею целиком в TradingView.

Ниже – **полный код**, включающий:

- параметры стратегии и комиссии;
- расчёт линии регрессии и R^2;
- правила входа и выхода;
- трейлинг-стоп;
- вспомогательные элементы интерфейса (таблица со значениями индикаторов).

Полный код стратегии на Pine Script v6. Верхняя панель - код стратегии:

```
//@version=6
strategy("Линейная регрессия + R² стратегия",
     overlay=true,
     commission_type=strategy.commission.percent,   // Тип комиссии:
процент
     commission_value=0.04,                         // Значение
комиссии: 0.04%
     slippage=10,                                   // Проскальзывание в
тиках
     process_orders_on_close=true,
     default_qty_type=strategy.fixed,
     default_qty_value=3,
     initial_capital=500000)                        // Начальный капитал

// === Входные параметры ===
// Параметры линейной регрессии
src = input.source(hlcc4, title="Источник данных")
len = input.int(defval=20, minval=1, title="Длина периода линейной
регрессии")
r_squared_threshold = input.float(0.2, title="Порог R-squared",
step=0.01, minval=0)
trailingStopOffset = input.float(3.9, "Отступ трейлинг-стопа (%)",
step=0.1, minval=0.1)
```

```
// Выбор таймфрейма для расчетов
targetTimeframe = input.timeframe("", title="Таймфрейм для расчетов",
     tooltip="Укажите таймфрейм, на котором должны производиться
расчеты")

// === Расчеты линейной регрессии и R-squared ===
// Получаем данные с выбранного таймфрейма
targetClose = request.security(syminfo.tickerid, targetTimeframe, src)

// Расчет линии линейной регрессии
lrc = ta.linreg(targetClose, len, 0)

// Расчет коэффициента детерминации R-squared
correlation_coeff = ta.correlation(targetClose, lrc, len)
r_squared = math.pow(correlation_coeff, 2)

// Обработка случая, когда стандартное отклонение источника равно нулю
(все значения одинаковы)
is_constant_src = ta.stdev(targetClose, len) == 0
r_squared_adjusted = is_constant_src ? 1.0 : r_squared

// === Условия входа ===
// Условие для лонг: линия регрессии ниже цены закрытия и R² растет и
выше порога
long_condition = lrc < close and r_squared_adjusted > r_squared_
adjusted[1] and r_squared_adjusted > r_squared_threshold

// Условие для шорт: линия регрессии выше цены закрытия и R² растет и
выше порога
short_condition = lrc > close and r_squared_adjusted > r_squared_
adjusted[1] and r_squared_adjusted > r_squared_threshold

// === Трейлинг-стоп на основе процентного отступа ===
var float highestLongPrice = na
var float lowestShortPrice = na
var float trailingStopLong = na
```

```
var float trailingStopShort = na

// Флаг для блокировки новых сигналов при активной позиции
var bool blockNewSignals = false

// Обновляем трейлинг-стопы
if (strategy.position_size > 0)  // Если открыта длинная позиция
    highestLongPrice := na(highestLongPrice) ? close : math.
max(highestLongPrice, close)
    trailingStopLong := highestLongPrice * (1 - trailingStopOffset / 100)
    blockNewSignals := true

if (strategy.position_size < 0)  // Если открыта короткая позиция
    lowestShortPrice := na(lowestShortPrice) ? close : math.
min(lowestShortPrice, close)
    trailingStopShort := lowestShortPrice * (1 + trailingStopOffset / 100)
    blockNewSignals := true

// Проверяем условия выхода
if (strategy.position_size > 0 and close <= trailingStopLong)
    strategy.close("Лонг")
    highestLongPrice := na
    trailingStopLong := na
    blockNewSignals := false

if (strategy.position_size < 0 and close >= trailingStopShort)
    strategy.close("Шорт")
    lowestShortPrice := na
    trailingStopShort := na
    blockNewSignals := false

// === Открытие позиций (только если нет активной позиции) ===
if (long_condition and not blockNewSignals and strategy.position_size == 0)
    strategy.entry("Лонг", strategy.long)
    highestLongPrice := close
    trailingStopLong := close * (1 - trailingStopOffset / 100)
    blockNewSignals := true
```

```
if (short_condition and not blockNewSignals and strategy.position_size == 0)
    strategy.entry("Шорт", strategy.short)
    lowestShortPrice := close
    trailingStopShort := close * (1 + trailingStopOffset / 100)
    blockNewSignals := true

// === Отображение ===
// Отображение линии линейной регрессии
plot(lrc, color = color.blue, title = "Линия линейной регрессии",
     style = plot.style_line, linewidth = 2)

// Отображение трейлинг-стопов
plot(strategy.position_size > 0 ? trailingStopLong : na, title="Лонг
трейлинг-стоп", color=color.green, linewidth=2, style=plot.style_linebr)
plot(strategy.position_size < 0 ? trailingStopShort : na, title="Шорт
трейлинг-стоп", color=color.red, linewidth=2, style=plot.style_linebr)

// Дополнительные информационные панели
var table info = table.new(position.top_right, 3, 4, border_width=1)
table.cell(info, 0, 0, "Linear Regression", bgcolor=color.new(color.
blue, 90), text_color=color.white)
table.cell(info, 0, 1, "Значение:", text_color=color.white)
table.cell(info, 1, 1, str.tostring(lrc, "#.##"), text_color=color.white)
table.cell(info, 0, 2, "R-squared:", text_color=color.white)
table.cell(info, 1, 2, str.tostring(r_squared_adjusted, "#.####"), text_
color=r_squared_adjusted > r_squared_threshold ? color.green : color.red)
table.cell(info, 0, 3, "Сигнал:", text_color=color.white)
table.cell(info, 1, 3, blockNewSignals ? "Заблокирован" : long_condition
? "ЛОНГ" : short_condition ? "ШОРТ" : "Нет сигнала",
           text_color=blockNewSignals ? color.yellow :
                      long_condition ? color.green :
                      short_condition ? color.red :
                      color.white)
```

Нижняя панель – код индикатора:

```
//@version=6
indicator(title="Коэффициент детерминации (R-squared)", shorttitle="R²",
overlay=false, precision=4)

// Настройки индикатора
src = input.source(close, title="Источник данных")
len = input.int(20, minval=2, title="Длина периода") // Для корреляции
нужно минимум 2 точки

// Выбор таймфрейма для расчетов
targetTimeframe = input.timeframe("", title="Таймфрейм для расчетов",
    tooltip="Укажите таймфрейм, на котором должны производиться расчеты
регрессии и R²")

targetSrc = request.security(syminfo.tickerid, targetTimeframe, src,
lookahead=barmerge.lookahead_on)

// Расчет линии линейной регрессии (Ŷ - предсказанные значения)
lrc = ta.linreg(targetSrc, len, 0)

// Расчет коэффициента детерминации R-squared
// R² = (Correlation(Y, Ŷ))²
// Y - это targetSrc (фактические значения)
// Ŷ - это lrc (предсказанные значения)
correlation_coeff = ta.correlation(targetSrc, lrc, len)
r_squared = math.pow(correlation_coeff, 2)

// Обработка случая, когда стандартное отклонение источника равно нулю
(все значения одинаковы)
is_constant_src = ta.stdev(targetSrc, len) == 0
r_squared_adjusted = is_constant_src ? 1.0 : r_squared

// Отображение R-squared на графике
plot(r_squared_adjusted, color=color.new(color.blue, 0), title="R-
squared", style=plot.style_line, linewidth=2)

// Уровни для R-squared для лучшей визуальной интерпретации
hline(1, "Идеальное соответствие", color.gray, linestyle=hline.style_dashed)
```

```
hline(0.8, "Очень сильное", color.new(color.green, 50), linestyle=hline.
style_dotted)
hline(0.5, "Среднее", color.new(color.orange, 50), linestyle=hline.
style_dotted)
hline(0.2, "Слабое", color.new(color.red, 50), linestyle=hline.style_
dotted)
hline(0, "Нет соответствия", color.gray, linestyle=hline.style_dashed)
```

Разные части кода:

- **strategy(...)** – задаём параметры торговли: комиссии, проскальзывание, количество контрактов. Это делает тест реалистичным.
- **ta.linreg(...)** – строим линию регрессии.
- **ta.correlation(...) + pow(… ,2)** – получаем R^2.
- **strategy.entry()** – место, где мы нажимаем на "курок" и открываем позицию.
- **Трейлинг-стоп** – механизм выхода, который поднимается (или опускается) вслед за ценой.

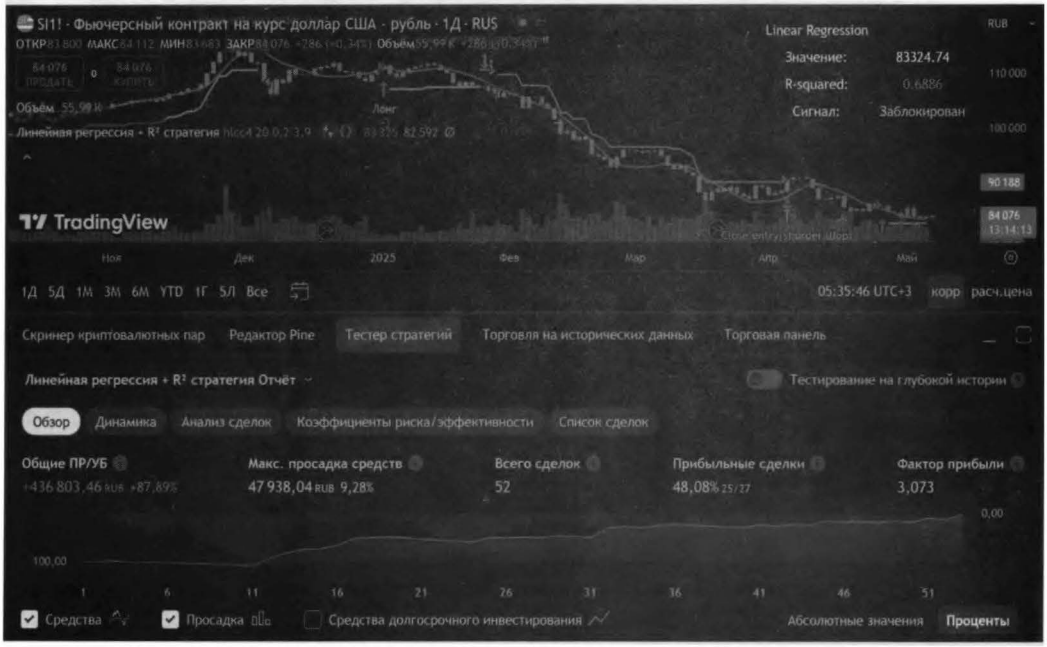


Рис. 9.5. ***Линия регрессии, зоны входа и динамика трейлинг-стопа***

Анализ результатов

Запускаем бэктест на реальных данных фьючерса Si.

Чтобы сравнить устойчивость идеи, тестируем два таймфрейма:

- **дневной** – спокойный, устойчивый, подходит для долгих движений;
- **часовой** – более шумный, но информативный.

Результаты на дневном графике



Рис. 9.6. ***Результаты бэктеста на дневном графике Si***

Результаты на часовом графике

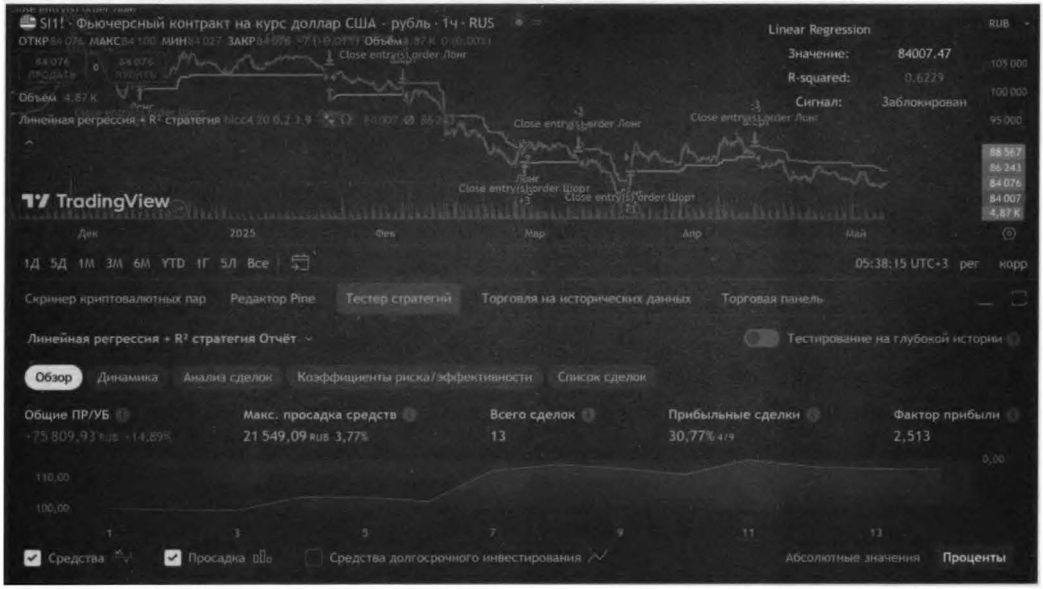


Рис. 9.7. ***Результаты бэктеста на часовом графике Si***

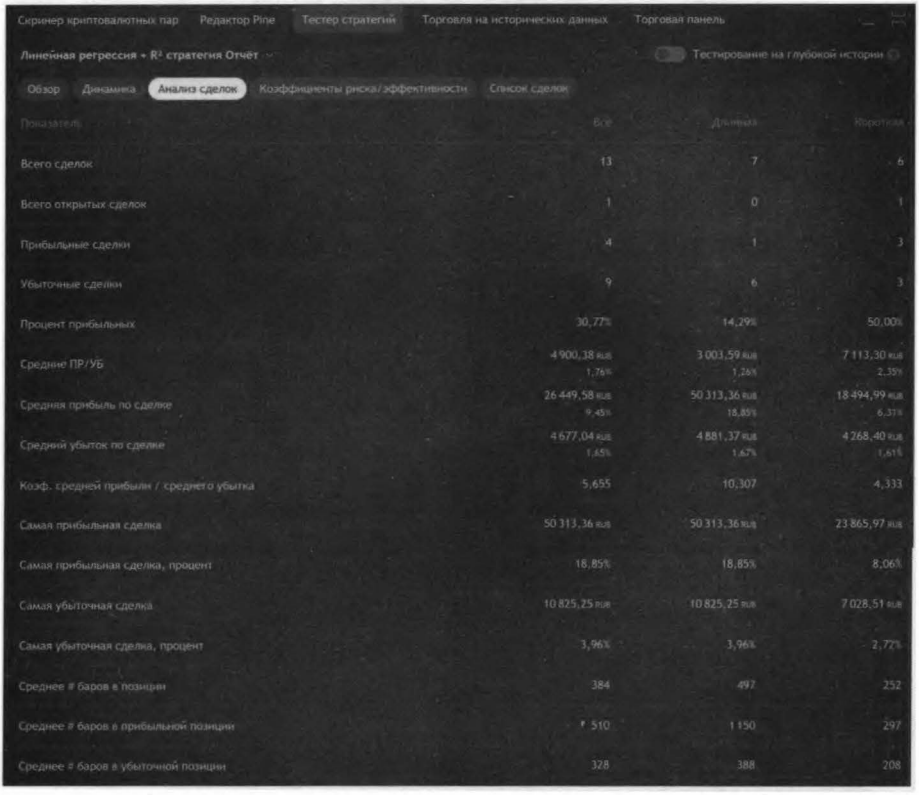


Рис. 9.8. ***Результаты бэктеста на часовом графике Si***

Нам не нужны сверхдоходности, чтобы бэктест оказался полезным.

Это лучше, чем гадать. Это лучше, чем верить собственному "ощущению". Мы получили фундамент: навык, который позволяет принимать решения спокойнее и увереннее.

9.2. Практикум №2. Глубокий анализ в Python

Если тест в TradingView был похож на быстрый эскиз на салфетке, то переход на Python – это как создание чертежа. Меняем удобный, но ограниченный графический интерфейс на мощь программного кода.

Зачем нам Python, если есть TradingView?

Платформы вроде TradingView незаменимы и удобны для быстрой проверки гипотез. Мы в несколько кликов набросали идею, увидели результат на графике и за 15 минут получили ответ: "стоит ли копать дальше?". Но как только мы захотим копнуть глубже, мы неизбежно упремся в ограничения:

- **Ограниченная история:** на бесплатных тарифах глубина исторических данных сильно урезана. Мы не сможем проверить, как вела бы себя идея в кризис 2008 года. Python же позволяет "скормить" стратегии данные за любой период, хоть за 20 лет, если мы их найдем.
- **Только встроенные данные:** Pine Script работает преимущественно с данными платформы, подключение внешних источников там это нетривиальная задача. Мы не можем подключить альтернативного поставщика котировок, добавить данные по инфляции, новости из Telegram-канала или произвольные показатели из отчетов компаний. В Python мы вольны смешивать любые источники: от биржевого API до самодельного парсера сайтов.
- **Изолированность:** стратегия на Pine Script "живет" внутри TradingView. Ее нельзя встроить в другую программу, превратить в телеграм-бота или заставить управлять реальным счетом у брокера, который не поддерживается платформой. Скрипт на Python – это универсальный "кирпичик", который можно встроить куда угодно.

- **Сложность анализа:** Pine Script отлично показывает кривую доходности, но что, если мы захотим провести сложный статистический анализ, применить методы машинного обучения для поиска оптимальных параметров или построить тепловую карту корреляции доходности с фазами луны? Для таких задач нужны специализированные библиотеки, и Python – абсолютный чемпион по их количеству и качеству.

Переходя на Python, мы получаем полный контроль над процессом. Мы сами решаем, откуда брать данные, как их обрабатывать, по какой логике тестировать и как интерпретировать результаты.

Подготовка к тесту: ищем, на чем торговать

Прежде чем строить сложную систему, нам нужен фундамент – качественные данные.

Почему? Представьте, что тест показал сигнал на покупку акции "Рога и копыта Индастриз" по 100 рублей. Мы радостно записываем в отчет прибыльную сделку. Но в реальности в этот день по такой цене никто не продавал нужный нам объем. В биржевом стакане было пусто. В итоге наша идеальная на бумаге сделка никогда бы не состоялась. Тест на таких данных – самообман.

Чтобы наши расчеты были честными, мы должны проводить их на самых ликвидных инструментах – тех, где всегда есть высокий спрос и предложение. Наша первая задача – найти такие акции.

Напишем скрипт, который найдет 15 самых ликвидных акций на Московской бирже за последние 3 месяца:

1. Получаем с биржи полный список всех торгуемых акций (тикеров). Для этого можно использовать как официальный API Московской биржи, так и API брокеров.
2. Запускаем цикл: для каждой акции из списка запрашиваем исторические дневные данные за последние три месяца.
3. Для каждой акции суммируем ее торговый оборот в рублях за этот период.
4. Сортируем полученный список по убыванию оборота и берем первые 15 позиций.

На Python такой скрипт, использующий API будет состоять из нескольких частей. Сначала получим список всех доступных тикеров акций на основном рынке (TQBR) Московской биржи:

```
import requests

# Делаем запрос к серверу
url = "https://iss.moex.com/iss/engines/stock/markets/shares/
boards/TQBR/securities.json"
params = {
    "iss.meta": "off",
    "iss.only": "marketdata",
    "marketdata.columns": "SECID"
}
response = requests.get(url, params=params)
response.raise_for_status()  # проверка на ошибку

# Извлекаем тикеры
data = response.json()
tickers = [item[0] for item in data['marketdata']['data']]

# Формируем строку
security_codes_str = f"security_codes = {tuple(tickers)}"
print(security_codes_str)
```

Результатом будет строка вида `security_codes = ('SBER', 'GAZP', 'LKOH', ...)`, которую можно вставить в библиотеку, которая поможет скачать нам сами котировки.

Получив этот список тикеров, мы скачиваем исторические данные по каждому из них и сохраняем в папку Data/Tinkoff/. Каждый файл называем по шаблону TQBR.Тикер_D1.txt

Я предлагаю использовать библиотеку **cia76/TinvestPy** с *https://github.com/cia76/TinvestPy*, её автор энтузиаст автоматизации и в текущем году активно поддерживает подобные библиотеки для всех брокеров, предоставляющих API. Воспользуемся скриптом **Examples/Bars.py** из библиотеки TinkoffPy (или её аналогом для Алор/Финам/БКС) для скачивания истории, мы получаем набор текстовых файлов.

Имя	Тип	↓Размер	Дата
..		<Папка>	24.05.2025 11:51
TQBR.TRNFP_D1	txt	434 371	24.05.2025 15:16
TQBR.SIBN_D1	txt	416 152	24.05.2025 15:15
TQBR.TATN_D1	txt	401 230	24.05.2025 15:16
TQBR.MSNG_D1	txt	396 764	24.05.2025 15:14
TQBR.MTSS_D1	txt	395 194	24.05.2025 15:14
TQBR.RTKMP_D1	txt	392 711	24.05.2025 15:15
TQBR.TGKB_D1	txt	390 536	24.05.2025 15:16
TQBR.TGKN_D1	txt	390 050	24.05.2025 15:16
TQBR.TGKA_D1	txt	371 945	24.05.2025 15:16
TQBR.ROSN_D1	txt	363 720	24.05.2025 15:15
TQBR.FEES_D1	txt	359 459	24.05.2025 15:12
TQBR.SNGSP_D1	txt	359 296	24.05.2025 15:16
TQBR.TGKBP_D1	txt	359 134	24.05.2025 15:16
TQBR.SBER_D1	txt	355 556	24.05.2025 15:11
TQBR.NVTK_D1	txt	354 234	24.05.2025 15:14
TQBR.RTKM_D1	txt	350 228	24.05.2025 15:15
TQBR.LKOH_D1	txt	345 696	24.05.2025 15:13
TQBR.SNGS_D1	txt	344 991	24.05.2025 15:15
TQBR.CHMF_D1	txt	341 623	24.05.2025 15:12
TQBR.SBERP_D1	txt	340 886	24.05.2025 15:15
TQBR.AFLT_D1	txt	333 909	24.05.2025 15:12
TQBR.PLZL_D1	txt	328 749	24.05.2025 15:15
TQBR.LSRG_D1	txt	327 987	24.05.2025 15:13

Рис. 9.9. ***История в файлах***

Скрипт: находим самые ликвидные акции по кварталам за последние 20 лет

Теперь, когда исторические данные по всем доступным акциям доступны, мы можем приступить к анализу их ликвидности.

Для этой задачи напишем Python-скрипт `liquid_stocks_by_quarter.py`.

Его основная цель – обработать все скачанные файлы с дневными котировками, рассчитать суммарный торговый оборот в рублях для каждой акции поквартально за указанный период и отобрать топ-N наиболее ликвидных бумаг для каждого квартала.

Ключевые шаги работы скрипта:

1. **Поиск файлов**: скрипт сканирует указанную нами папку (Data/Tinkoff/) на наличие файлов с данными (TQBR.ТИКЕР_D1.txt).
2. **Чтение данных**: каждый файл загружается в pandas DataFrame. Производится парсинг дат и необходимых колонок (OHLCV).

3. **Получение размера лота**: для корректного расчета денежного оборота, скрипт пытается подключиться через TinkoffPy к API брокера и получить актуальный размер лота для каждого тикера. Если API недоступно или информация по тикеру отсутствует, используется размер лота равный 1 (это может исказить реальные результаты).

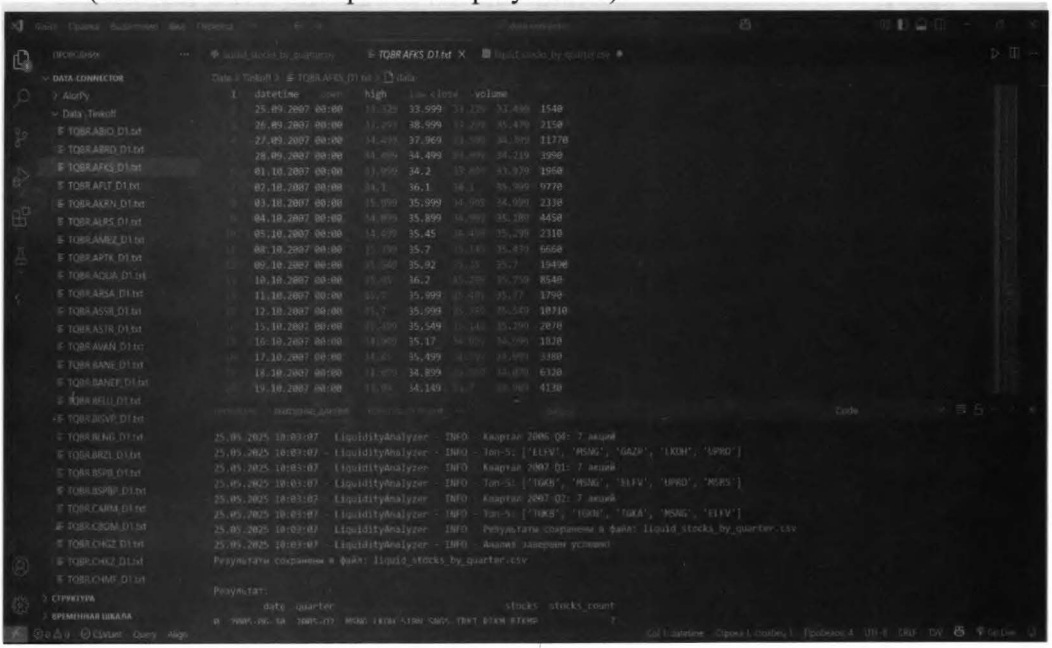


Рис. 9.10. ***TQBR.ТИКЕР_D1.txt.***

4. **Расчет оборота в рублях**: для каждой дневной свечи вычисляется оборот.
5. **Агрегация по кварталам**: дневные обороты суммируются по каждому тикеру внутри каждого квартала.
6. **Отбор топ-N акций**: для каждого квартала акции сортируются по убыванию суммарного рублевого оборота, и отбирается заданное количество лидеров.
7. **Сохранение результата**: итоговая таблица с самыми ликвидными акциями по кварталам сохраняется в CSV-файл.
8. **Скрипт также включает логирование** для отслеживания процесса обработки и возможных ошибок, а также отладочный вывод для проверки корректности чтения форматов файлов. Параметры, такие как *глубина анализа в годах* (years_back) и *количество отбираемых акций* (top_n_stocks), легко настраиваются.

Вот сам скрипт, файл `liquid_stocks_by_quarter.py`:

```
import os
import pandas as pd
import logging
import glob
from datetime import datetime, timedelta
from collections import defaultdict
from TinkoffPy import TinkoffPy

class LiquidityAnalyzer:
    def __init__(self, data_folder="Data/Tinkoff/", years_back=5,
top_n_stocks=40):
        """
        Анализатор ликвидности акций по кварталам

        Args:
            data_folder: путь к папке с данными
            years_back: количество лет назад для анализа
            top_n_stocks: количество самых ликвидных акций для отбора
        """
        self.data_folder = data_folder
        self.years_back = years_back
        self.top_n_stocks = top_n_stocks
        self.tp_provider = None
        self.lot_sizes = {}

        # Настройка логирования
        logging.basicConfig(
            format='%(asctime)s - %(name)s - %(levelname)s -
%(message)s',
            datefmt='%d.%m.%Y %H:%M:%S',
            level=logging.INFO,
            handlers=[
                logging.FileHandler('liquidity_analysis.log', encoding
='utf-8'),
                logging.StreamHandler()
            ]
```

```
        )
        self.logger = logging.getLogger('LiquidityAnalyzer')

    def get_quarter_string(self, date):
        """
        Получение строкового представления квартала

        Args:
            date: дата в формате datetime или Timestamp

        Returns:
            str: строка вида "2024-Q1"
        """
        quarter = (date.month - 1) // 3 + 1
        return f"{date.year}-Q{quarter}"

    def connect_to_tinkoff(self):
        """Подключение к Tinkoff API для получения размеров лотов"""
        try:
            self.tp_provider = TinkoffPy()
            self.logger.info("Подключение к Tinkoff API установлено")
        except Exception as e:
            self.logger.error(f"Ошибка подключения к Tinkoff API: {e}")
            raise

    def get_lot_size(self, ticker):
        """
        Получение размера лота для тикера

        Args:
            ticker: тикер без префикса TQBR (например, SBER)

        Returns:
            int: размер лота
        """
        if ticker in self.lot_sizes:
            return self.lot_sizes[ticker]
```

```
        try:
            dataname = f"TQBR.{ticker}"
            class_code, security_code = self.tp_provider.dataname_to_
class_code_symbol(dataname)
            si = self.tp_provider.get_symbol_info(class_code, security_
code)

            if si:
                lot_size = si.lot
                self.lot_sizes[ticker] = lot_size
                self.logger.info(f"Размер лота для {ticker}: {lot_size}")
                return lot_size
            else:
                self.logger.warning(f"Информация о тикере {ticker} не
найдена")
                return 1  # По умолчанию
        except Exception as e:
            self.logger.error(f"Ошибка получения размера лота для
{ticker}: {e}")
            return 1

    def debug_file_format(self, file_path, max_lines=5):
        """
        Отладочная функция для проверки формата файла

        Args:
            file_path: путь к файлу
            max_lines: максимальное количество строк для вывода
        """
        try:
            with open(file_path, 'r', encoding='utf-8') as f:
                lines = f.readlines()[:max_lines]
                self.logger.info(f"Первые {len(lines)} строк файла
{os.path.basename(file_path)}:")
                for i, line in enumerate(lines):
                    self.logger.info(f"  {i}: {line.rstrip()}")
        except Exception as e:
            self.logger.error(f"Ошибка чтения файла {file_path}: {e}")
```

```
    def load_stock_data(self, file_path):
        """
        Загрузка данных по акции из файла

        Args:
            file_path: путь к файлу с данными

        Returns:
            pd.DataFrame: данные акции
        """
        try:
            # Для отладки показываем формат файла
            if self.logger.level <= logging.DEBUG:
                self.debug_file_format(file_path)

            df = pd.read_csv(file_path, sep='\t')

            # Проверяем наличие колонки datetime
            if 'datetime' not in df.columns:
                self.logger.error(f"Колонка 'datetime' не найдена в
файле {file_path}")
                self.logger.error(f"Доступные колонки: {list(df.
columns)}")
                return None

            # Преобразуем колонку datetime в правильный формат
            try:
                df['datetime'] = pd.to_datetime(df['datetime'],
format='%d.%m.%Y %H:%M')
            except ValueError:
                # Пробуем другие форматы
                try:
                    df['datetime'] = pd.to_datetime(df['datetime'])
                except Exception as e:
                    self.logger.error(f"Не удалось преобразовать
datetime в файле {file_path}: {e}")
                    return None
```

```
            df.set_index('datetime', inplace=True)

            # Проверяем наличие необходимых колонок
            required_columns = ['open', 'high', 'low', 'close', 'volume']
            missing_columns = [col for col in required_columns if col
not in df.columns]
            if missing_columns:
                self.logger.error(f"Отсутствуют колонки в файле {file_
path}: {missing_columns}")
                return None

            self.logger.debug(f"Успешно загружено {len(df)} записей из
{file_path}")
            return df

        except Exception as e:
            self.logger.error(f"Ошибка загрузки файла {file_path}: {e}")
            return None

    def calculate_quarterly_liquidity(self):
        """
        Расчет ликвидности акций по кварталам

        Returns:
            dict: словарь с ликвидностью по кварталам
        """
        # Получаем все файлы с данными
        pattern = os.path.join(self.data_folder, "TQBR.*_D1.txt")
        files = glob.glob(pattern)

        if not files:
            self.logger.error(f"Не найдены файлы в папке {self.data_
folder}")
            return {}

        self.logger.info(f"Найдено {len(files)} файлов для обработки")
```

```
        # Подключаемся к Tinkoff API
        try:
            self.connect_to_tinkoff()
        except Exception as e:
            self.logger.error(f"Не удалось подключиться к Tinkoff API: {e}")
            # Продолжаем работу без API, используя размер лота = 1

        # Определяем диапазон дат
        end_date = datetime.now()
        start_date = end_date - timedelta(days=365 * self.years_back)

        # Словарь для хранения данных по кварталам
        quarterly_data = defaultdict(dict)

        self.logger.info(f"Анализируем период с {start_date.date()} по
{end_date.date()}")

        processed_count = 0
        error_count = 0

        for file_path in files:
            try:
                # Извлекаем тикер из имени файла
                filename = os.path.basename(file_path)
                ticker = filename.split('.')[1].split('_')[0]

                self.logger.info(f"Обрабатываем тикер: {ticker}
({processed_count + 1}/{len(files)})")

                # Загружаем данные
                df = self.load_stock_data(file_path)
                if df is None:
                    error_count += 1
                    continue

                # Фильтруем по датам
                df = df[(df.index >= start_date) & (df.index <= end_date)]
```

```
                if df.empty:
                    self.logger.warning(f"Нет данных для {ticker} в
указанном периоде")
                    continue

                # Получаем размер лота
                lot_size = self.get_lot_size(ticker) if self.tp_
provider else 1

                # Рассчитываем объем в рублях
                df['volume_rub'] = df['volume'] * lot_size * df['close']

                # Группируем по кварталам
                df['quarter'] = df.index.to_period('Q')
                quarterly_volumes = df.groupby('quarter')['volume_
rub'].sum()

                # Сохраняем данные по кварталам
                for quarter, volume in quarterly_volumes.items():
                    quarter_end = quarter.end_time
                    quarterly_data[quarter_end][ticker] = volume

                processed_count += 1

            except Exception as e:
                self.logger.error(f"Ошибка обработки файла {file_path}:
{e}")
                error_count += 1
                continue

        # Закрываем соединение с Tinkoff API
        if self.tp_provider:
            try:
                self.tp_provider.close_channel()
            except Exception as e:
                self.logger.warning(f"Ошибка закрытия соединения с
Tinkoff API: {e}")
```

```
        self.logger.info(f"{'=' * 70}")
        self.logger.info(f"Обработано файлов: {processed_count},
ошибок: {error_count}")
        return quarterly_data

    def select_top_liquid_stocks(self, quarterly_data):
        """
        Отбор самых ликвидных акций по кварталам

        Args:
            quarterly_data: данные о ликвидности по кварталам

        Returns:
            dict: топ акций по кварталам
        """
        top_stocks_by_quarter = {}

        for quarter_date, stocks_data in quarterly_data.items():
            # Сортируем акции по объему торгов
            sorted_stocks = sorted(stocks_data.items(), key=lambda x:
x[1], reverse=True)

            # Берем топ акций
            top_stocks = [stock[0] for stock in sorted_stocks[:self.
top_n_stocks]]
            top_stocks_by_quarter[quarter_date] = top_stocks

            # Исправленное форматирование квартала
            quarter_str = self.get_quarter_string(quarter_date)
            self.logger.info(f"Квартал {quarter_str}: {len(top_
stocks)} акций")
            self.logger.info(f"Топ-5: {top_stocks[:5]}")

        return top_stocks_by_quarter

    def save_to_csv(self, top_stocks_by_quarter, filename="liquid_
stocks_by_quarter.csv"):
        """
```

```
        Сохранение результатов в CSV файл

        Args:
            top_stocks_by_quarter: топ акций по кварталам
            filename: имя файла для сохранения
        """
        rows = []

        for quarter_date, stocks in sorted(top_stocks_by_quarter.items()):
            stocks_str = ",".join(stocks)
            quarter_str = self.get_quarter_string(quarter_date)
            rows.append({
                'date': quarter_date.strftime('%Y-%m-%d'),
                'quarter': quarter_str,
                'stocks': stocks_str,
                'stocks_count': len(stocks)
            })

        df = pd.DataFrame(rows)
        df.to_csv(filename, index=False, encoding='utf-8')

        self.logger.info(f"Результаты сохранены в файл: {filename}")
        return filename

    def run_analysis(self):
        """
        Запуск полного анализа ликвидности

        Returns:
            str: путь к файлу с результатами
        """
        self.logger.info("Начинаем анализ ликвидности акций")

        try:
            # Рассчитываем ликвидность по кварталам
            quarterly_data = self.calculate_quarterly_liquidity()
```

```
            if not quarterly_data:
                self.logger.error("Не удалось получить данные для анализа")
                return None

            # Отбираем топ ликвидных акций
            top_stocks = self.select_top_liquid_stocks(quarterly_data)

            if not top_stocks:
                self.logger.error("Не удалось отобрать топ акций")
                return None

            # Сохраняем результаты
            filename = self.save_to_csv(top_stocks)

            self.logger.info("Анализ завершен успешно")
            return filename

        except Exception as e:
            self.logger.error(f"Критическая ошибка анализа: {e}")
            return None

# Поиск
if __name__ == '__main__':
    # Создаем анализатор с отладочным уровнем логирования
    analyzer = LiquidityAnalyzer(
        data_folder="Data/Tinkoff/",
        years_back=20,
        top_n_stocks=7
    )

    # Устанавливаем уровень отладки
    analyzer.logger.setLevel(logging.DEBUG)

    # Сначала тестируем загрузку одного файла
    test_files = glob.glob(os.path.join(analyzer.data_folder, "TQBR.*_
D1.txt"))
    if test_files:
        test_file = test_files[0]
```

```
        print(f"\nТестируем загрузку файла: {test_file}")

        # Показываем содержимое файла
        analyzer.debug_file_format(test_file, max_lines=3)

        # Пробуем загрузить данные
        df = analyzer.load_stock_data(test_file)
        if df is not None:
            print(f"Успешно загружено {len(df)} записей")
            print(f"Диапазон дат: {df.index.min()} - {df.index.
max()}")
            print(f"Колонки: {list(df.columns)}")
            print(f"Первые 3 записи:")
            print(df.head(3))
        else:
            print("Ошибка загрузки данных")
            exit(1)

    # Запускаем полный анализ
    result_file = analyzer.run_analysis()

    if result_file:
        print(f"Результаты сохранены в файл: {result_file}")

        # Показываем результат
        try:
            df = pd.read_csv(result_file)
            print("\nРезультат:")
            print(df.head())
        except Exception as e:
            print(f"Ошибка чтения результата: {e}")
    else:
        print("Анализ не задался")
```

Результат работы – файл `liquid_stocks_by_quarter.csv` с самыми ликвидными акциями.

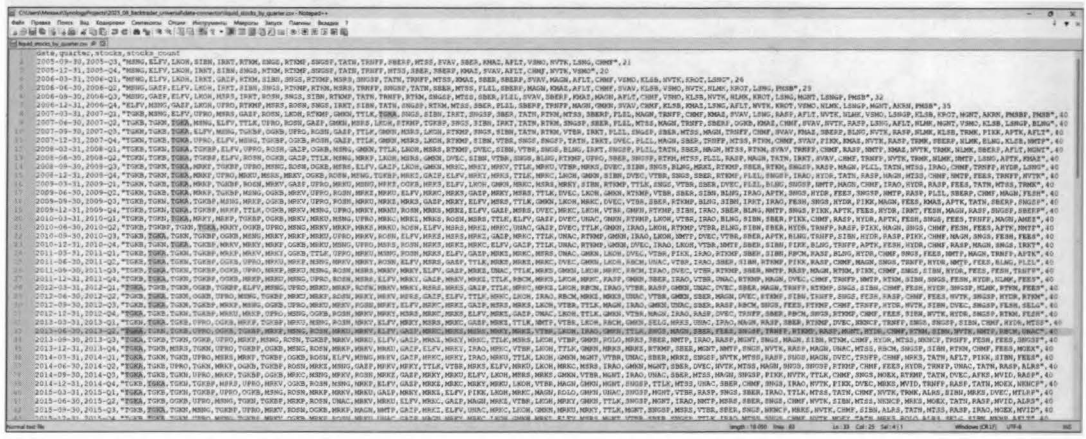


Рис. 9.11. ***Скачанная история***

9.3. Бэктестинг на Python

Вход в алготрейдинг обычно стоит дорого: либо вы покупаете профессиональный софт, либо тратите годы на свой фреймворк. Мы срежем угол и соберем универсальный шаблон на базе бесплатной библиотеки Backtrader. Я называю этот подход "конструктор", где вместо написания кода с нуля, мы собираем стратегию из готовых блоков.

Почему стоит выбрать Backtrader?

Прежде чем писать код, мы должны выбрать библиотеку, которая возьмет на себя тяжелую математику: сведение ордеров, учет комиссий, управление капиталом.

Я изучил множество вариантов. Есть **Freqtrade** – отличная штука, но она намертво заточена под криптовалюты. Есть **Nautilus Trader** – тяжелая артиллерия для высокочастотной торговли (HFT), слишком сложная для наших задач.

Предлагаю остановиться на **Backtrader**. Это библиотека-конструктор, она не навязывает жестких рамок, имеет огромное сообщество и отлично справляется с портфельным тестированием. В отличие от TradingView, где мы тестировали один актив, здесь мы сможем "зарядить" в тест сразу весь

портфель из топ-15 акций Мосбиржи и посмотреть, как стратегия управляет общим капиталом.

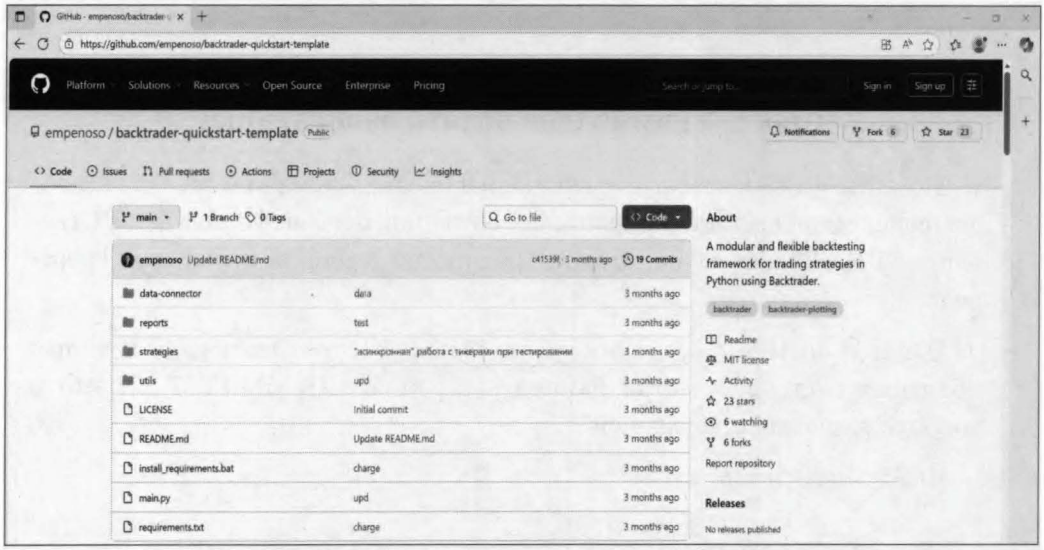


Рис. 9.12. ***https://github.com/empenoso/backtrader-quickstart-template***

Архитектура нашего "Народного шаблона"

Создадим модульную структуру проекта:

1. **Главный пульт (main.py)** – здесь мы нажимаем кнопки "Старт", выбираем режим (*Тест* или *Оптимизация*) и задаем даты.
2. **Папка стратегий (strategies/)** – каждая идея в отдельном файле.
3. **Данные (data-connector/)** – склад наших csv/txt файлов.
4. **Отчеты (reports/)** – сюда будем складывать результаты: красивые графики и текстовые сводки.

Вам не нужно быть профессиональным разработчиком, чтобы этим пользоваться. Это конструктор.

Начнем с установки необходимых библиотек. Нам понадобятся **backtrader** для движка, **matplotlib** для графиков и **pandas** для работы с таблицами.

Создайте файл *requirements.txt*:

```
backtrader
matplotlib
seaborn
pandas
```

Шаг 1. Учим Python читать наши данные.

Помните файлы, которые мы скачали в предыдущем разделе? Они имеют специфический формат: разделители-табуляции, особый формат даты. Стандартный Backtrader их не поймет. Нам нужно написать небольшой "переводчик".

В папке **utils** мы создадим файл *custom_csv.py*. Это класс, который объясняет системе: "Смотри, дата у нас в формате ДД.ММ.ГГГГ ЧЧ:ММ, а колонки разделены табуляцией".

Файл *./utils/custom_csv.py*:

```
import backtrader as bt

class CustomCSVData(bt.feeds.GenericCSVData):
    """
    Класс для загрузки данных из CSV в кастомном формате.
    Формат даты: ДД.ММ.ГГГГ ЧЧ:ММ
    Разделитель: Табуляция (\t)
    """
    params = (
        ('separator', '\t'),  # Указываем, что разделитель - это
табуляция

        ('dtformat', '%d.%m.%Y %H:%M'),
        ('datetime', 0),
        ('open', 1),
        ('high', 2),
        ('low', 3),
        ('close', 4),
        ('volume', 5),
        ('openinterest', -1), # -1 означает, что этого столбца нет
        ('time', -1), # Важно, если время уже включено в datetime
```

```
        ('nullvalue', float('nan')),
        ('reverse', False),
    )
```

Шаг 2. Фундамент для стратегий.

Чтобы не переписывать один и тот же код (комиссии, стартовый капитал, логирование сделок) для каждой новой идеи, мы создадим "Родительскую стратегию". Все наши будущие роботы будут наследовать эти базовые навыки.

Обратите внимание на метод **log**. В алготрейдинге критически важно понимать, что делал робот в каждый момент времени. Мы научим стратегию записывать каждое свое действие.

Файл *./strategies/base_strategy.py*:

```
import backtrader as bt

class BaseStrategy(bt.Strategy):
    """
    Базовый класс для всех стратегий.
    Определяет обязательные атрибуты, которые должны быть установлены в
дочерних классах.
    """
    # --- КОНФИГУРАЦИЯ СТРАТЕГИИ (должна быть переопределена в дочерних
классах) ---
    tickers = []  # Список тикеров для тестирования, e.g., ['TQBR.SBER_
D1.txt']
    start_cash = 1_000_000.0  # Начальный капитал
    commission = 0.001  # Комиссия брокера (0.1%)

    # Параметры для оптимизации. ДОЛЖНО БЫТЬ РОВНО ДВА ПАРАМЕТРА для
построения тепловой карты.
    # e.g. opt_params = {'fast_ma': range(5, 15), 'slow_ma': range(20, 50)}
    opt_params = {}

    def __init__(self):
        """Логика, общая для всех стратегий, может быть здесь."""
```

```
        # Можно добавить логирование или общие индикаторы
        pass

    def log(self, txt, dt=None):
        """Вспомогательная функция для логирования"""
        dt = dt or self.datas[0].datetime.date(0)
        print(f'{dt.isoformat()} | {txt}')

    # --- МЕТОДЫ, КОТОРЫЕ НУЖНО РЕАЛИЗОВАТЬ В ДОЧЕРНИХ СТРАТЕГИЯХ ---
    def next(self):
        """
        Этот метод должен быть реализован в каждой конкретной стратегии.
        Содержит основную логику принятия решений.
        """
        raise NotImplementedError("Метод 'next' должен быть реализован в
дочерней стратегии.")

    def notify_order(self, order):
        """Обработка статуса ордера"""
        if order.status in [order.Submitted, order.Accepted]:
            return
        if order.status in [order.Completed]:
            if order.isbuy():
                self.log(f'BUY EXECUTED, Price: {order.executed.price: .2f},
Cost: {order.executed.value:.2f}, Comm: {order.executed.comm:.2f}')
            elif order.issell():
                self.log(f'SELL EXECUTED, Price: {order.executed.price:.2f},
Cost: {order.executed.value:.2f}, Comm: {order.executed.comm:.2f}')
        elif order.status in [order.Canceled, order.Margin, order.
Rejected]:
            self.log('Order Canceled/Margin/Rejected')

    def notify_trade(self, trade):
        """Обработка закрытия сделки"""
        if not trade.isclosed:
            return
        self.log(f'OPERATION PROFIT, GROSS {trade.pnl:.2f}, NET {trade.
pnlcomm:.2f}')
```

А чтобы система сама находила новые стратегии, добавим файл *__init__.py* в папке стратегий. Это позволит нам просто закидывать новые файлы в папку, и они автоматически появятся в системе.

Файл *./strategies/__init__.py*:

```
import os
import importlib
import inspect
from .base_strategy import BaseStrategy

# Словарь для хранения доступных стратегий: {'ИмяКласса': Класс}
available_strategies = {}

# Путь к текущей директории (папке strategies)
package_dir = os.path.dirname(__file__)

for filename in os.listdir(package_dir):
    if filename.endswith('.py') and filename not in ['__init__.py', 'base_
strategy.py']:
        module_name = f"strategies.{filename[:-3]}"

        try:
            module = importlib.import_module(module_name)

            # Ищем в модуле классы, которые являются наследниками
BaseStrategy
            for name, obj in inspect.getmembers(module, inspect.isclass):
                if issubclass(obj, BaseStrategy) and obj is not
BaseStrategy:
                    available_strategies[name] = obj
                    print(f"Загружена стратегия: '{name}' из модуля
'{module_name}'")

        except ImportError as e:
            print(f"Ошибка импорта модуля {module_name}: {e}")

if not available_strategies:
    print("Внимание: ни одной стратегии не найдено в папке 'strategies'.")
```

Шаг 3. Реализация стратегии пересечения средних.

В TradingView мы тестировали один актив. В Python мы хотим тестировать весь портфель. Но есть подвох: разные акции начинают торговаться в разное время. Если мы просто запустим тест, Backtrader может выдать ошибку, пытаясь купить акцию, которой еще не существует в прошлом.

Поэтому наша стратегия будет умной – добавим методы `prenext` и `nextstart`, которые проверяют: "Накопилось ли достаточно истории по этой конкретной бумаге?". Если да – торгуем её. Если нет – ждем. Это делает код сложнее, но зато он не упадет на реальных данных.

Ниже код стратегии `SmaCrossStrategy`. Это полноценный робот, который управляет портфелем, динамически распределяет капитал (аллокация) и следит за рисками.

Файл *./strategies/sma_cross_strategy.py*:

```
import math
import collections
import backtrader as bt
from .base_strategy import BaseStrategy


class SmaCrossStrategy(BaseStrategy):
    """
    SMA Crossover strategy - асинхронная версия.

    Стратегия работает отдельно для каждого data feed и начинает
торговать по инструменту
    с того момента, когда для него накопилось достаточно баров для
расчёта индикаторов.
    """

    # --- Параметры стратегии (удобно оптимизировать/переопределять) ---
    params = dict(
        fast_ma=20,
        slow_ma=50,
        alloc_percent_per_ticker=0.90,  # доля выделяемого капитала
на тикер
```

```
        min_size=1,                          # минимальный лот/штука для
открытия позиции
        rebal_weekday=None,                  # если нужно использовать
таймеры, задать 0..6 (Mon..Sun), None - отключено
        log_on_connect=True,                 # логировать подключение
инструмента
    )

    # Список тикеров и базовые финансовые параметры (берутся из
BaseStrategy, но можно переопределить)
    tickers = [
        'SBER', 'VTBR', 'GAZP', 'LKOH', 'NVTK', 'YDEX', 'T'
    ]
    start_cash = 500_000.0
    commission = 0.001

    # Параметры для оптимизации (должно быть ровно 2 ключа для
генерации тепловой карты в main.py)
    opt_params = {
        'fast_ma': range(5, 21, 5),
        'slow_ma': range(30, 61, 10),
    }

    def __init__(self):
        super().__init__()

        # ---- структуры данных ----
        # Индикаторы для каждого data feed
        self.inds = collections.defaultdict(dict)

        # Список data feeds, которые уже "живут" (накопили минимально
необходимое число баров)
        self.d_with_len = []  # будет заполняться в prenext /
nextstart

        # Множество data feeds, которые мы уже логировали как
"подключённые"
        self.live_logged = set()
```

```
        # кеш минимальных периодов для каждого data
        self.min_periods = {}

        # Кэш для расчёта капитала на тикер (пересчитывается при
необходимости)
        self._capital_per_active_ticker = None
        self._capital_calc_for_len = None  # длина, для которой
посчитан кэш

        # создаём индикаторы для каждого data (они будут возвращать
nan до накопления данных)
        for d in self.datas:
            # индикаторы создаём в общем виде, чтобы Backtrader
следил за их буферами
            self.inds[d]['fast_sma'] = bt.ind.SMA(d.close,
period=self.p.fast_ma)
            self.inds[d]['slow_sma'] = bt.ind.SMA(d.close,
period=self.p.slow_ma)
            self.inds[d]['crossover'] = bt.ind.CrossOver(self.inds[d]
['fast_sma'],
                                                         self.inds[d]
['slow_sma'],
                                                         plot=False)

            # минимальный период данных для этого инструмента
            self.min_periods[d] = max(int(self.p.fast_ma),
int(self.p.slow_ma))

        # если нужен таймер - добавим его (опционально)
        if self.p.rebal_weekday is not None:
            # таймер вызывает notify_timer по расписанию; можно
использовать для периодического ребаланса
            self.add_timer(when=bt.Timer.SESSION_START,
                           weekdays=[self.p.rebal_weekday],
                           weekcarry=True)

    # --------------------------
    # Хуки для асинхронных данных
```

```
    # ------------------------
    def prenext(self):
        """
        Вызывается когда ещё не все гарантии (т.е. не все индикаторы/
данные готовые),
        но некоторые данные уже приходят. Сохраняем список доступных
data и делегируем в next.
        """
        # Сохраняем data feeds, у которых len > 0
        self.d_with_len = [d for d in self.datas if len(d)]
        # Сбрасываем кэш капитала, т.к. число активных инструментов
изменилось
        self._capital_per_active_ticker = None
        # Делаем торговую логику (защищённую внутри next)
        self.next()

    def nextstart(self):
        """
        Вызывается ровно один раз, когда все гарантии выполнены
впервые.
        Здесь мы считаем, что все data feeds "доступны".
        """
        # Теперь все данные соответствуют требованиям
        self.d_with_len = list(self.datas)
        self._capital_per_active_ticker = None
        self.next()

    # ------------------------
    # Вспомогательные функции
    # ------------------------
    def _ensure_capital_per_ticker(self):
        """
        Вычисляет и кэширует капитал, выделяемый на каждый *активный*
тикер.
       Переоценивается автоматически, если число активных инструментов
изменилось.
        """
        active_n = max(1, len(self.d_with_len) if len(self.d_with_len)
else len(self.datas))
```

```
        if self._capital_per_active_ticker is not None and self._
capital_calc_for_len == active_n:
            return

        cash = float(self.broker.getcash()) if self.broker.getcash()
is not None else float(self.start_cash)
        # делим текущие наличные на число активных инструментов
        capital_per = cash / float(active_n)
        self._capital_per_active_ticker = capital_per
        self._capital_calc_for_len = active_n
        self.log(f"[CAPITAL] cash={cash:.2f}, active_n={active_n},
capital_per_ticker={capital_per:.2f}")

    def _data_ready(self, d):
        """
        Проверка готовности конкретного data feed для торговли:
        - len(d) >= минимального требуемого значения
        - индикаторы дали значения (не NaN)
        - текущая цена положительна
        """
        try:
            # данные должны быть вообще
            if len(d) < 1:
                return False

            # минимальная длина для расчёта индикаторов
            if len(d) < self.min_periods.get(d, 0):
                return False

            # текущее значение цены
            price = d.close[0]
            if price is None or price <= 0 or (isinstance(price, float)
and math.isnan(price)):
                return False

            # индикаторы: fast and slow SMA и crossover
            fast = self.inds[d]['fast_sma'][0]
            slow = self.inds[d]['slow_sma'][0]
```

```
            xo = self.inds[d]['crossover'][0]

            # Проверяем на NaN (Backtrader может вернуть float('nan')
до заполнения)
            for v in (fast, slow, xo):
                if v is None:
                    return False
                if isinstance(v, float) and math.isnan(v):
                    return False

            return True

        except Exception:
            # безопасно отлавливаем любые ошибки доступа к data/
индикаторам
            return False

    # --------------
    # Основной цикл
    # --------------
    def next(self):
        """
        Основная логика - работает только с self.d_with_len
(подмножество доступных данных).
        Все вычисления защищены и не предполагают одинаковой длины
data feeds.
        """
        # Обновляем список доступных data feeds на каждом шаге
(пристаревшая записанная d_with_len могло измениться)
        # - но делаем это быстро, используя len(d)
        current_available = [d for d in self.datas if len(d)]
        # если список изменился - обновим кэш и уведомим
        if set(current_available) != set(self.d_with_len):
            self.d_with_len = current_available
            self._capital_per_active_ticker = None  # пересчитать капитал
        # Убедимся, что вычислен капитал для текущего набора активных
тикеров
        self._ensure_capital_per_ticker()
```

```
        # Перебираем только те data feeds, которые готовы к торговле
        for d in list(self.d_with_len):
            # защита: есть ли достаточно данных и индикаторов для
этого инструмента
            if not self._data_ready(d):
                continue

            # логируем факт "подключения" инструмента только один раз
            if self.p.log_on_connect and d not in self.live_logged:
                self.live_logged.add(d)
                try:
                    current_date = bt.num2date(d.datetime[0]).
strftime('%Y-%m-%d')
                except Exception:
                    current_date = str(d.datetime[0])
                self.log(f"[ADD] {getattr(d, '_name', 'unknown')}
подключен к торгам с {current_date}")

            # значения индикаторов
            xo = self.inds[d]['crossover'][0]
            fast = self.inds[d]['fast_sma'][0]
            slow = self.inds[d]['slow_sma'][0]

            # текущая цена и позиция
            price = d.close[0]
            pos = self.getposition(d)

            # Торговая логика: простое пересечение SMA
            # открываем длинную позицию при пересечении снизу-вверх
            if pos.size == 0:
                if xo > 0:
                    self._execute_buy_signal(d, price)
            else:
                # закрываем позицию при пересечении сверху-вниз
                if xo < 0:
                    self._execute_sell_signal(d, price, pos)
```

```
    # -------------------------
    # Исполнение ордеров
    # -------------------------
    def _execute_buy_signal(self, data, price):
        """Исполняет сигнал на покупку для одного инструмента."""
        try:
            alloc = float(self._capital_per_active_ticker) *
float(self.p.alloc_percent_per_ticker)
            size = int(alloc / price) if price > 0 else 0

            if size < int(self.p.min_size):
                self.log(f"[BUY SKIP] {getattr(data,'_name', 'unknown')}
- недостаточный size (price={price:.2f}, alloc={alloc:.2f},
size={size})")
                return

            self.log(f"[BUY] {getattr(data,'_name','unknown')}
price={price:.2f} size={size} alloc={alloc:.2f}")
            self.buy(data=data, size=size)

        except Exception as e:
            self.log(f"[ERROR BUY] {getattr(data,'_name','unknown')}: {e}")

    def _execute_sell_signal(self, data, price, position):
        """Закрытие позиции по инструменту."""
        try:
            self.log(f"[SELL] {getattr(data,'_name','unknown')}
closing size={position.size} price={price:.2f}")
            self.close(data=data)
        except Exception as e:
            self.log(f"[ERROR SELL] {getattr(data,'_name', 'unknown' )
}: {e}")

    # -------------------------
    # notify hooks
    # -------------------------
    def notify_order(self, order):
        """Расширенное логирование статусов ордера."""
```

```
        super().notify_order(order)

        # стандартная обработка уже выполнена в BaseStrategy; добавим
дополнительные строки при необходимости
        if order.status in [order.Completed]:
            dname = getattr(order.data, '_name', 'unknown')
            if order.isbuy():
                # order.executed.price / .size / .value / .comm обычно
доступны
                self.log(f"[ORDER DONE BUY] {dname} price={order.
executed.price:.2f} size={getattr(order.executed, 'size', 'n/a')}")
            elif order.issell():
                self.log(f"[ORDER DONE SELL] {dname} price={order.
executed.price:.2f} size={getattr(order.executed, 'size', 'n/a')}")

    def notify_trade(self, trade):
        """Логирование закрытых сделок."""
        super().notify_trade(trade)
        if trade.isclosed:
            dname = getattr(trade.data, '_name', 'unknown') if hasattr
(trade, 'data') else 'unknown'
            self.log(f"[TRADE CLOSED] {dname} GROSS {trade.pnl:.2f}
NET {trade.pnlcomm:.2f}")

    def notify_timer(self, timer, when, *args, **kwargs):
        """
        Если активирован таймер (rebal_weekday), можно реализовать
периодические действия.
        По умолчанию - просто логируем событие и оставляем
возможность расширить поведение.
        """
        try:
            self.log(f"[TIMER] when={when} timer={timer} args={args}
kwargs={kwargs}")
            # Пример: можно вызывать ребаланс/функции раз в неделю здесь.
        except Exception as e:
            self.log(f"[TIMER ERROR] {e}")
```

Шаг 4. Анализ и отчетность.

Голый график эквити (кривой капитала) – это хорошо, но нужны цифры. Мы подключим два модуля:

1. **Анализатор Сортино (*sortino_analyzer.py*)**: в отличие от коэффициента Шарпа, он наказывает стратегию только за "плохую" волатильность (падение вниз), а не за резкий рост прибыли.
2. **Генератор отчетов (*report_generator.py*)**: будет создавать красивые текстовые сводки и "тепловые карты" для оптимизации.

Файл *./utils/sortino_analyzer.py*:

```
import math
import backtrader as bt
import numpy as np

class SortinoRatio(bt.Analyzer):
    """
    Analyzer для коэффициента Сортино.
    Параметры (через cerebro.addanalyzer):
      - riskfreerate: годовая безрисковая ставка (в долях,
например 0.01 = 1%)
      - periods_per_year: число периодов в году (252 для дневных
баров)
      - mar: минимально приемлемая годовая доходность (annual), по
умолчанию 0.0
    Возвращает dict с ключами:
      - sortino, annual_return, downside_deviation, n_periods
    """
    params = dict(
        riskfreerate=0.0,
        periods_per_year=252,
        mar=0.0
    )

    def __init__(self):
        self.equity = []
        self.prev_val = None
```

```
    def start(self):
        # инициализация, если хотим что-то в будущем
        pass

    def next(self):
        # собираем значение портфеля в каждом баре
        cur = self.strategy.broker.getvalue()
        if self.prev_val is None:
            self.prev_val = cur
            return
        # считаем периодный (например, дневной) доход
        r = (cur / self.prev_val) - 1.0
        self.equity.append(r)
        self.prev_val = cur

    def stop(self):
        res = {'sortino': None, 'annual_return': None, 'downside_
deviation': None, 'n_periods': len(self.equity)}
        if not self.equity:
            self.rets = res
            return

        returns = np.array(self.equity, dtype=float)
        n = len(returns)
        # годовая геометрическая доходность
        total_growth = (1.0 + returns).prod()
        if total_growth <= 0:
            annual_return = -1.0
        else:
            annual_return = total_growth ** (self.p.periods_per_
year / n) - 1.0

        # переводим годовой MAR / riskfree в периодную величину
        mar_period = (1.0 + self.p.mar) ** (1.0 / self.p.periods_
per_year) - 1.0
        rf_annual = float(self.p.riskfreerate)
```

```
        # downside deviations: отклонения ниже MAR (period)
        downside = np.minimum(0.0, returns - mar_period)
        # среднее квадратичное берём по всем периодам (N) -> более
стандартный вариант
        mean_sq = np.mean(downside ** 2)
        downside_dev = math.sqrt(mean_sq) * math.
sqrt(self.p.periods_per_year)

        numerator = annual_return - rf_annual
        sortino = None
        if downside_dev > 0:
            sortino = numerator / downside_dev
        else:
            # нет даунсайд-волатильности
            sortino = float('inf') if numerator > 0 else None

        res.update({
            'sortino': None if sortino is None else float(sortino),
            'annual_return': float(annual_return),
            'downside_deviation': float(downside_dev),
            'n_periods': n
        })
        self.rets = res

    def get_analysis(self):
        return self.rets
```

Файл *./utils/report_generator.py*:

```
import os
import datetime
import pandas as pd

def generate_backtest_report(strategy_result, strategy_name, from_date,
to_date, reports_dir):
    """Генерирует и сохраняет текстовый отчет по результатам
бэктеста."""
    p = getattr(strategy_result, 'params', None)
```

```
    param_dict = {k: getattr(p, k) for k in p._getkeys()} if p else {}

    # Анализаторы (безопасно)
    try:
        analyzer = strategy_result.analyzers.trade_analyzer.get_
analysis()
    except Exception:
        analyzer = None
    try:
        drawdown = strategy_result.analyzers.drawdown.get_analysis()
    except Exception:
        drawdown = None
    try:
        returns = strategy_result.analyzers.returns.get_analysis()
    except Exception:
        returns = {}
    try:
        sortino_data = strategy_result.analyzers.sortino.get_analysis()
or {}
    except Exception:
        sortino_data = {}

    sortino_value = sortino_data.get('sortino', None)

    # Сделки
    total_closed_trades = getattr(analyzer, 'total', None)
    total_closed_trades = (total_closed_trades.closed if total_closed_
trades is not None else 0) if analyzer else 0
    won_trades = (analyzer.won.total if analyzer and getattr(analyzer,
'won', None) else 0)
    lost_trades = (analyzer.lost.total if analyzer and getattr(analyzer,
'lost', None) else 0)

    # Начальный и конечный капитал
    initial_value = getattr(strategy_result.broker, 'startingcash', None)
    final_value = strategy_result.broker.getvalue()
    pnl = (final_value - initial_value) if initial_value is not None else
None
```

```
    # Расчет процента от начального капитала
    pnl_percent = (pnl / initial_value * 100) if (pnl is not None and
initial_value is not None and initial_value != 0) else 0.0

    # Расчет Buy and Hold: корректно извлекаем первый и последний close
из первого датасета
    buy_and_hold_return = None
    buy_and_hold_absolute = None
    try:
        data = strategy_result.datas[0]
        n_bars = len(data)
        if n_bars >= 1:
            # Первый close: индекс - (n_bars-1)
            first_idx = -(n_bars - 1)
            first_close = data.close[first_idx]
            last_close = data.close[0]
            if first_close != 0:
                buy_and_hold_return = (last_close - first_close) / first_
close
                # Абсолютное значение buy and hold в денежном эквиваленте
                buy_and_hold_absolute = initial_value * buy_and_hold_
return if initial_value else 0
    except Exception:
        buy_and_hold_return = None
        buy_and_hold_absolute = None

    # Расчет Profit Factor
    gross_profit = (analyzer.won.pnl.total if analyzer and
getattr(analyzer, 'won', None) else 0)
    gross_loss = abs((analyzer.lost.pnl.total if analyzer and
getattr(analyzer, 'lost', None) else 0))
    profit_factor = gross_profit / gross_loss if gross_loss != 0 else
(float('inf') if gross_profit > 0 else 0.0)

    # Win rate
    win_rate = (won_trades / total_closed_trades * 100) if total_closed_
trades else 0.0
```

```
    # Максимальная просадка в абсолютных значениях
    max_drawdown_percent = (drawdown.max.drawdown if drawdown and
getattr(drawdown, 'max', None) else 0.0)
    max_drawdown_absolute = initial_value * (max_drawdown_percent / 100)
if initial_value else 0.0

    # Форматирование Sortino безопасно
    def fmt_sortino(v):
        if v is None:
            return "N/A"
        if v == float('inf'):
            return "inf"
        return f"{v:.2f}"

    # Функция для форматирования денежных сумм с разделителем тысяч
    def fmt_money(value):
        if value is None:
            return "N/A"
        # Форматируем с пробелами как разделителями тысяч
        return f"{value:,.2f}".replace(',', ' ')

    # Функция для форматирования buy and hold
    def fmt_buy_hold():
        if buy_and_hold_return is None or buy_and_hold_absolute is None:
            return "Результат 'Купил и держал': N/A"
        return f"Результат 'Купил и держал': {fmt_money(buy_and_hold_
absolute)} [{buy_and_hold_return * 100:.2f}%]"

    report_lines = [
        f"--- ОТЧЕТ ПО БЭКТЕСТУ СТРАТЕГИИ: {strategy_name} ---",
        f"Бумаги в тесте: {[d._name for d in strategy_result.datas]}",
        f"Параметры: {param_dict}",
        f"Период тестирования: с {from_date.strftime('%d.%m.%Y')} по
{to_date.strftime('%d.%m.%Y')}",
        "--- РЕЗУЛЬТАТЫ ---",
        f"Итоговая прибыль/убыток: {fmt_money(pnl)} [{pnl_
percent:.2f}%]" if pnl is not None else "Итоговая прибыль/убыток: N/A",
```

```
        f"Доходность (годовых): {returns.get('rnorm100', 0.0):.2f}%",
        fmt_buy_hold(),
        f"Максимальная просадка: {fmt_money(max_drawdown_absolute)}
[{max_drawdown_percent:.2f}%]",
        f"Всего сделок: {total_closed_trades}",
        f"Процент прибыльных сделок: {win_rate:.2f}% ({won_trades} из
{total_closed_trades})",
        f"Фактор прибыли: {profit_factor:.2f}" if profit_factor not
in (float('inf'), None) else f"Фактор прибыли: {'inf' if profit_
factor==float('inf') else 'N/A'}",
        f"Коэффициент Сортино: {fmt_sortino(sortino_value)}",
        "-------------------------------------------------"
    ]

    # Убираем None-строки
    report_lines = [r for r in report_lines if r is not None]

    report_content = "\n".join(report_lines)
    print(report_content)

    # Сохранение отчета
    timestamp = datetime.datetime.now().strftime('%Y-%m-%d_%H-%M')
    period_str = f"{from_date.strftime('%Y%m%d')}-{to_date.strftime
('%Y%m%d')}"
    filename = f"{strategy_name}_test_{timestamp}_{period_str}.txt"
    filepath = os.path.join(reports_dir, filename)

    with open(filepath, 'w', encoding='utf-8') as f:
        f.write(report_content)
    print(f"\nОтчет сохранен в: {filepath}")


def generate_optimization_report(results, strategy_name, opt_param_
names, from_date, to_date, reports_dir):
    """Генерирует и сохраняет отчет по оптимизации и тепловую карту."""

    import matplotlib.pyplot as plt
    import seaborn as sns
```

```
    timestamp = datetime.datetime.now().strftime('%Y-%m-%d_%H-%M')
    period_str = f"{from_date.strftime('%Y%m%d')}-{to_date.
strftime('%Y%m%d')}"

    # --- 1. Текстовый отчет ---
    txt_filename = f"{strategy_name}_opt_{timestamp}_{period_str}.txt"
    txt_filepath = os.path.join(reports_dir, txt_filename)

    # Берем первую стратегию из результатов для получения тикеров
    first_strategy = results[0][0]
    tickers = getattr(first_strategy.__class__, "tickers", [])

    with open(txt_filepath, 'w', encoding='utf-8') as f:
        f.write(f"--- РЕЗУЛЬТАТЫ ОПТИМИЗАЦИИ СТРАТЕГИИ: {strategy_name}
---\n")
        f.write(f"Бумаги в тесте: {', '.join(tickers)}\n")
        f.write(f"Параметры для оптимизации: {', '.join(opt_param_names)
}\n\n")

        # Заголовок таблицы
        header = f"{opt_param_names[0]:<10} | {opt_param_names[1]:<10} |
{'ProfitFactor':<15} | {'TotalTrades':<12}\n"
        f.write(header)
        f.write("-" * len(header) + "\n")

        opt_data = []
        for run in results:
            strategy = run[0]  # Берем одну стратегию из каждого прогона
            params = strategy.p
            analyzer = strategy.analyzers.trade_analyzer.get_analysis()

            gross_profit = analyzer.won.pnl.total or 0
            gross_loss = abs(analyzer.lost.pnl.total or 0)
            profit_factor = gross_profit / gross_loss if gross_loss != 0
else float('inf')
            total_trades = analyzer.total.closed or 0
```

```
            p1_val = getattr(params, opt_param_names[0])
            p2_val = getattr(params, opt_param_names[1])

            f.write(f"{p1_val:<10} | {p2_val:<10} | {profit_factor:<15.2f}
| {total_trades:<12}\n")

            opt_data.append({
                opt_param_names[0]: p1_val,
                opt_param_names[1]: p2_val,
                'profit_factor': profit_factor
            })

    print(f"Отчет по оптимизации сохранен в: {txt_filepath}")

    # --- 2. Тепловая карта ---
    if not opt_data:
        print("Нет данных для построения тепловой карты.")
        return

    df = pd.DataFrame(opt_data)

    df['profit_factor'] = df['profit_factor'].replace([float('inf'),
-float('inf')], 10)

    try:
        heatmap_data = df.pivot(index=opt_param_names[0], columns=opt_
param_names[1], values='profit_factor')

        # Принудительно устанавливаем backend
        plt.switch_backend('Agg')

        plt.figure(figsize=(12, 8))
        sns.heatmap(heatmap_data, annot=True, fmt=".2f", cmap="viridis")
        plt.title(f'Тепловая карта оптимизации для {strategy_name}\
n(Фактор прибыли)')
        plt.xlabel(opt_param_names[1])
        plt.ylabel(opt_param_names[0])
```

```
        # Получаем имена тикеров из класса стратегии
        # Ограничиваем до 5 тикеров в имени файла
        ticker_names = [os.path.splitext(ticker)[0] for ticker in
tickers[:5]]
        if len(tickers) > 5:
            ticker_names.append("etc")  # добавим "etc", если тикеров
больше
        tickers_str = "_".join(ticker_names)
        png_filename = f"{strategy_name}_opt_heatmap_{tickers_str}_
{timestamp}_{period_str}.png"
        png_filepath = os.path.join(reports_dir, png_filename)

        plt.savefig(png_filepath)
        plt.close()  # Важно закрыть фигуру, чтобы не отображать ее в
jupyter/etc.

        print(f"Тепловая карта сохранена в: {png_filepath}")

    except Exception as e:
        print(f"Не удалось построить тепловую карту: {e}")
        print("Возможно, у вас только один параметр для оптимизации или
данные не подходят для сводной таблицы.")
```

Шаг 5. Главный пульт управления.

И наконец, мозг – файл *main.py*. Задается режим (MODE):

- 'BACKTEST' – одиночный прогон для проверки идеи.
- 'OPTIMIZATION' – поиск лучших параметров.

Скрипт сам подгрузит стратегию, найдет данные, запустит симуляцию и сохранит отчеты.

Файл *./main.py*:

```
# --- Системные и стандартные библиотеки ---
import os
import sys
```

```
import datetime

import matplotlib as mpl
mpl.use('Agg')  # Говорим matplotlib, что мы будем только сохранять в
файлы.
import matplotlib.pyplot as plt
# Принудительно устанавливаем backend для всех последующих импортов
plt.switch_backend('Agg')
from matplotlib.figure import Figure

import backtrader as bt

# --- Настройка кодировки ---
sys.stdout.reconfigure(encoding='utf-8')

# --- Импорты из нашего проекта ---
import strategies
from utils.custom_csv import CustomCSVData
from utils.sortino_analyzer import SortinoRatio
from utils.report_generator import generate_backtest_report, generate_
optimization_report

# --- Импорты для Live Trading (Tinkoff) ---
# Раскомментируйте, если будете использовать Live Trading
# from BackTraderTinkoff import TKStore
# import logging

# ==============================================================================
# --- ГЛАВНАЯ ПАНЕЛЬ УПРАВЛЕНИЯ ---
# ==============================================================================

# 1. ВЫБЕРИТЕ РЕЖИМ: 'BACKTEST' или 'OPTIMIZATION'
MODE = 'BACKTEST'

# 2. ВЫБЕРИТЕ СТРАТЕГИЮ из загруженных (см. вывод в консоли при запуске)
#    Имя должно точно совпадать с именем класса стратегии.
STRATEGY_TO_RUN = 'SmaCrossStrategy'
```

```
# 3. НАСТРОЙКИ ПЕРИОДА ТЕСТИРОВАНИЯ
FROM_DATE = datetime.datetime(2018, 1, 1)
TO_DATE = datetime.datetime(2025, 8, 12)

# 4. НАСТРОЙКИ LIVE TRADING (ЕСЛИ НУЖНО)
#    Для активации установите LIVE_TRADING = True
LIVE_TRADING = False
TINKOFF_TOKEN = "your_tinkoff_invest_api_token_here" # ВАШ ТОКЕН

# 5. ПУТИ К ПАПКАМ
DATA_DIR = 'data-connector/Data/Tinkoff/'
REPORTS_DIR = 'reports'

# ================================================================================
# --- КОНЕЦ ПАНЕЛИ УПРАВЛЕНИЯ ---
# ================================================================================

if __name__ == '__main__':
    # --- 1. Проверка и подготовка окружения ---
    os.makedirs(REPORTS_DIR, exist_ok=True)

    if STRATEGY_TO_RUN not in strategies.available_strategies:
        raise ValueError(f"Стратегия '{STRATEGY_TO_RUN}' не найдена. "
                         f"Доступные стратегии: {list(strategies.
available_strategies.keys())}")

    StrategyClass = strategies.available_strategies[STRATEGY_TO_RUN]

    # --- 2. Инициализация Cerebro для асинхронного режима ---
    cerebro = bt.Cerebro(
        stdstats=False,
        optreturn=False if MODE == 'OPTIMIZATION' else True,
        runonce=False,  # КЛЮЧЕВОЙ ПАРАМЕТР: отключаем синхронизацию данных
        preload=False,  # НЕ предзагружаем все данные в память
        oldsync=False,   # НЕ используем старый механизм синхронизации
        exactbars=-1    # ДОБАВИТЬ: позволяет разную длину данных
    )
```

```
    # --- Наблюдатели ---
    # Включаем стандартные наблюдатели (при желании можно оставить
stdstats=False и добавить только нужные)
    # Показываем Buy/Sell как маркеры над/под свечой (barplot=True) и
дистанцию от свечи (bardist)
    # cerebro.addobserver(bt.observers.BuySell, barplot=True,
bardist=0.015)

    # Показываем значение портфеля / кэша - удобный верхний график
    cerebro.addobserver(bt.observers.Value, plotname='Стоимость
портфеля')    # портфель value (включая cash)
    cerebro.addobserver(bt.observers.Cash, plotname='Свободные средства')
# cash отдельно
    cerebro.addobserver(bt.observers.DrawDown, plotname='Просадка (%)')
# просадка

    # Показываем полную информацию по сделкам (PnL на закрытие и т.д.)
    cerebro.addobserver(bt.observers.Trades, pnlcomm=True, plotname='P&L
по сделкам')

    # --- 3. Конфигурация в зависимости от режима (Live vs Backtest) ---
    if LIVE_TRADING:
        print("РЕЖИМ: РЕАЛЬНАЯ ТОРГОВЛЯ (TINKOFF)")
        # Здесь логика для подключения к Tinkoff, взятая из примера Игоря
Чечета
        # store = TKStore(token=TINKOFF_TOKEN)
        # broker = store.getbroker()
        # cerebro.setbroker(broker)

        # for ticker in StrategyClass.tickers:
        #     # формат тикера для Tinkoff может отличаться от имени файла
        #     dataname = ticker.split('_')[0]
        #     data = store.getdata(dataname=dataname, fromdate=FROM_DATE)
        #     cerebro.adddata(data)

        # cerebro.addsizer(bt.sizers.FixedSize, stake=10) # Пример сайзера
        print("ВНИМАНИЕ: Логика Live Trading закомментирована. "
```

```
            "Раскомментируйте и настройте ее при необходимости.")

    else: # Режим бэктеста или оптимизации
        print(f"РЕЖИМ: {MODE}")

        # --- 4. Загрузка данных ---

        # НАСТРОЙКИ МАРКЕТА И ФОРМАТА ФАЙЛОВ
        MARKET_PREFIX = "TQBR"
        TIMEFRAME_SUFFIX = "_D1.txt"

        for ticker in StrategyClass.tickers:
            filename = f"{MARKET_PREFIX}.{ticker}{TIMEFRAME_SUFFIX}"
            datapath = os.path.join(DATA_DIR, filename)
            if not os.path.exists(datapath):
                print(f"Внимание: Файл данных не найден: {datapath}")
                continue

            data_feed = CustomCSVData(
                dataname=datapath,
                fromdate=FROM_DATE,
                todate=TO_DATE
            )
            cerebro.adddata(data_feed, name=ticker)

            # привязываем observer к этой конкретной серии, чтобы стрелки
рисовались на её графике
            cerebro.addobserver(bt.observers.BuySell, barplot=True,
bardist=0.015, plotmaster=data_feed)

        # --- 5. Настройка брокера ---
        cerebro.broker.setcash(StrategyClass.start_cash)
        cerebro.broker.setcommission(commission=StrategyClass.commission)

        # --- 6. Добавление размера позиции (можно вынести в стратегию) ---
        # cerebro.addsizer(bt.sizers.PercentSizer, percents=95)

        # --- 7. Добавление анализаторов ---
```

```
        cerebro.addanalyzer(bt.analyzers.TradeAnalyzer, _name='trade_
analyzer')
        cerebro.addanalyzer(bt.analyzers.DrawDown, _name='drawdown')
        cerebro.addanalyzer(bt.analyzers.Returns, _name='returns')
        # Кастомный анализатор из файла
        cerebro.addanalyzer(SortinoRatio, _name='sortino', riskfreerate=
0.0, periods_per_year=252, mar=0.0)


    # --- 8. Добавление стратегии в Cerebro ---
    if MODE == 'OPTIMIZATION':
        opt_params = StrategyClass.opt_params
        if not opt_params or len(opt_params) != 2:
            raise ValueError("Для оптимизации с построением тепловой
карты "
                             "в стратегии должно быть определено ровно 2
параметра в 'opt_params'")

        opt_param_names = list(opt_params.keys())
        cerebro.optstrategy(StrategyClass, **opt_params)
        print(f"Запуск оптимизации для {STRATEGY_TO_RUN} по параметрам:
{opt_param_names}")

    else: # 'BACKTEST'
        cerebro.addstrategy(StrategyClass)
        print(f"Запуск бэктеста для {STRATEGY_TO_RUN}")


    # --- 9. Запуск процесса ---
    results = cerebro.run()

    # --- 10. Генерация отчетов ---
    if not LIVE_TRADING:
        if MODE == 'OPTIMIZATION':
            print("--- Генерация отчета по оптимизации ---")
            generate_optimization_report(
                results,
                strategy_name=STRATEGY_TO_RUN,
```

```
                opt_param_names=opt_param_names,
                from_date=FROM_DATE,
                to_date=TO_DATE,
                reports_dir=REPORTS_DIR
            )
        else: # 'BACKTEST'
            print("\n--- Генерация отчета по бэктесту ---")
            generate_backtest_report(
                results[0],
                strategy_name=STRATEGY_TO_RUN,
                from_date=FROM_DATE,
                to_date=TO_DATE,
                reports_dir=REPORTS_DIR
            )

            # --- Сохранение графиков в файлы (индивидуально для каждого
актива) ---
            try:
                # 1. Получаем список тикеров, которые были в бэктесте.
                #    Это нужно для осмысленных имен файлов.
                strategy_instance = results[0]
                tickers_in_test = [d._name for d in strategy_instance.
datas]
                num_figs = max(1, len(tickers_in_test))

                # 2. Вызываем plot() с параметром plotind=True.
                #    Backtrader вернет список фигур, по одной на каждый
актив.
                figures = cerebro.plot(
                    style='candlestick',
                    barup='green',
                    bardown='red',
                    vol=True,
                    plotind=True,
                    # Можно также задать размер для каждой индивидуальной
фигуры
                    fig_w=16, # Ширина каждой фигуры в дюймах
                    fig_h=9,   # Высота каждой фигуры в дюймах
```

```
                    # numfigs=num_figs  # <- добавлено: по одной фигуре на
тикер
                )

                # 3. Проходим по списку фигур и сохраняем каждую в свой
файл.
                timestamp = datetime.datetime.now().strftime('%Y-%m-%d_
%H-%M')
                period_str = f"{FROM_DATE.strftime('%Y%m%d')}-{TO_DATE.
strftime('%Y%m%d')}"

                # figures - это список списков, [[fig1], [fig2], ...].
Проходим по нему.
                for i, fig_group in enumerate(figures):
                    if i >= len(tickers_in_test):
                        # На случай, если фигур больше, чем тикеров
(маловероятно)
                        ticker_name = f"data_{i}"
                    else:
                        ticker_name = tickers_in_test[i]

                    # Имя файла теперь включает тикер
                    plot_filename = f"{STRATEGY_TO_RUN}_{ticker_name}_
plot_{timestamp}_{period_str}.png"
                    plot_filepath = os.path.join(REPORTS_DIR, plot_filename)

                    # Сохраняем фигуру (первый и единственный элемент в
fig_group)
                    fig_group[0].savefig(plot_filepath, dpi=200) # dpi=200-
300 для хорошего качества

                    print(f"График для '{ticker_name}' сохранен в: {plot_
filepath}")

            except Exception as e:
                print(f"Не удалось сохранить графики: {e}")
```

Запуск теста и анализ результатов

Итак, мы собрали нашу "машину".

Жмем на газ и запускаем *python main.py*. Что мы получаем?

В консоли и в папке **reports** появляется подробный отчет:

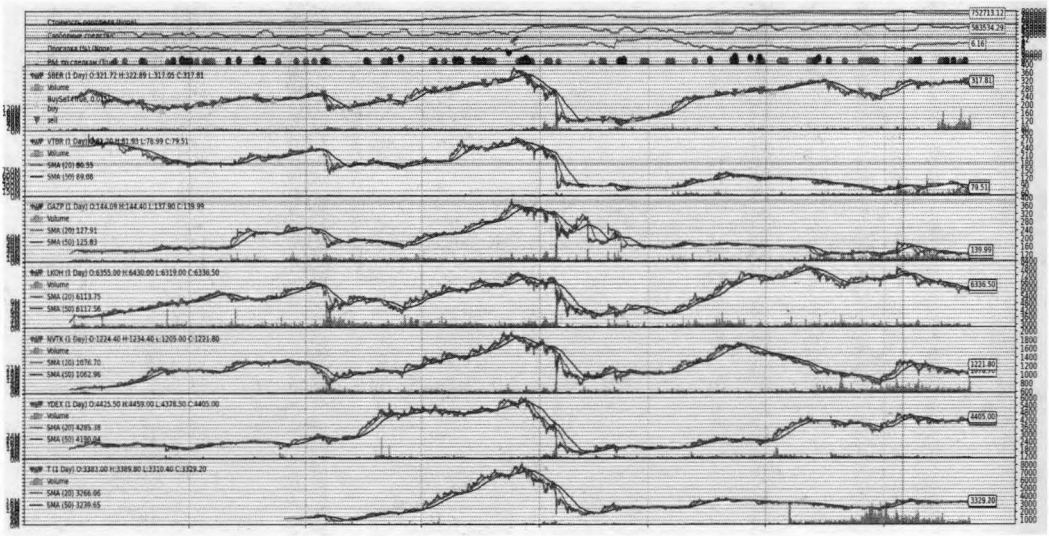


Рис. 9.13. ***Пример отчета работы стратегии на портфеле акций***

Мы видим не только итоговую прибыль, но и сравнение с "Купил и держи", максимальную просадку и фактор прибыли. Но самое главное – мы видим графики для каждого инструмента.

А если мы переключим режим на OPTIMIZATION, система прогонит сотни вариантов параметров (например, разные длины скользящих средних) и выдаст нам "тепловую карту".

Это позволяет увидеть не просто случайную удачную настройку, а целую зону стабильности. Если карта "пятнистая" (хорошие результаты вперемешку с убыточными) – стратегия ненадежна. Если мы видим большое яркое пятно – значит, мы нашли закономерность.

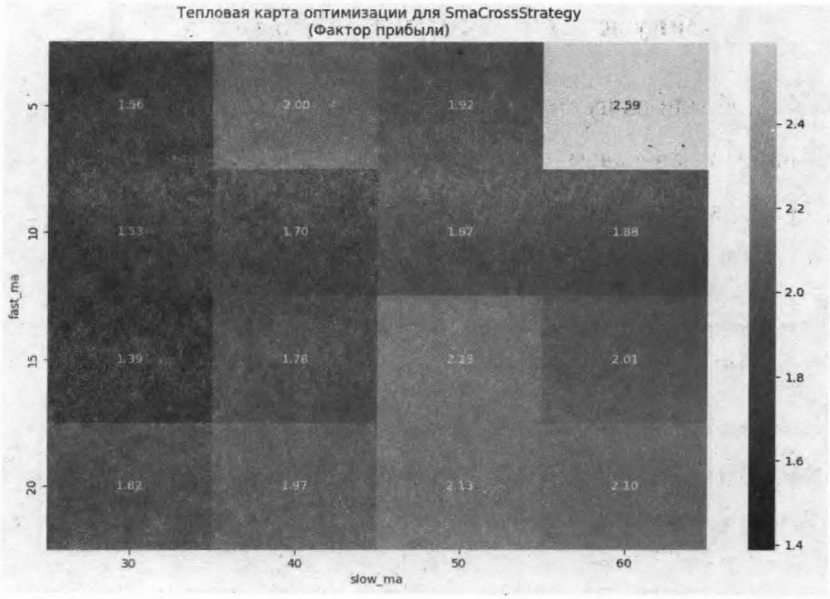


Рис. 9.14. ***Тепловая карта оптимизации: яркие зоны показывают самые устойчивые параметры***

Анализ результатов

Все результаты сохраняются в папку **reports**.

Для бэктеста создается текстовый отчет с полной статистикой и отдельные *.png*-файлы с графиками для каждого тикера.

Пример отчета:

```
--- ОТЧЕТ ПО БЭКТЕСТУ СТРАТЕГИИ: SmaCrossStrategy ---
Бумаги в тесте: ['SBER', 'VTBR', 'GAZP', 'LKOH', 'NVTK', 'YDEX', 'T']
Параметры: {'fast_ma': 20, 'slow_ma': 50, ...}
Период тестирования: с 01.01.2018 по 12.08.2025
--- РЕЗУЛЬТАТЫ ---
Итоговая прибыль/убыток: 252 713.12 [50.54%]
Доходность (годовых): 5.44%
Результат 'Купил и держал': 205 648.56 [41.13%]
Максимальная просадка: 67 124.98 [13.42%]
```

```
Всего сделок: 128
Процент прибыльных сделок: 41.41% (53 из 128)
Фактор прибыли: 2.13
Коэффициент Сортино: 1.08
------------------------------------------------
```

Для оптимизации создается текстовый отчет со всеми комбинациями параметров и **тепловая карта** (.png), на которой цветом показана эффективность (профит-фактор) каждой комбинации.

9.4. Заглядывая за горизонт – можно ли научить ИИ читать графики?

Мы только что научились проверять свои идеи "дедовским" способом: математикой и логикой. Если цена выше средней – покупаем. Если ниже – продаем. Это язык цифр, понятный компьютеру. Но давайте на секунду остановимся и вспомним, как описывают рынок телеграмм-каналы.

Разве они говорят: "Значение RSI пересекло отметку 70 сверху вниз при объеме 1.5 миллиона"? Скорее всего, нет.

Пишут проще: "Рынок перегрет, покупатели выдыхаются, похоже на разворот". Это образы, паттерны и слова.

И тут возникает самый интригующий вопрос этой главы – **а что, если мы попробуем объяснить рыночную ситуацию искусственному интеллекту не цифрами, а именно этими словами?**

Это не просто теоретическое упражнение. Это попытка зайти на территорию, где правят бал нейросети. В этом разделе мы проведем эксперимент, который еще пару лет назад казался фантастикой: мы попытаемся научить языковую модель "читать" график, превращенный в текст.

Революционная идея: от цифр к словам

Традиционный алготрейдинг кормит машину сухими цифрами: O, H, L, C, V (открытие, максимум, минимум, закрытие, объём). Алгоритм ищет математические закономерности.

Но языковые модели, подобные тем, что стоят за ChatGPT (о нем мы подробно поговорим в следующей главе), обучались на колоссальных объемах текста. Они "понимают" контекст, причинно-следственные связи и семантику.

Гипотеза, которую мы проверим, звучит так:

> Если опытный трейдер может взглянуть на график и описать его словами ("цена три дня растет на повышающемся объеме, пробивая сопротивление"), то, может быть, нейросеть сможет найти в этом описании скрытые сигналы, которые ускользают от обычной математики?

Мы не будем показывать модели график. Мы будем рассказывать ей историю о графике.

Для этого эксперимента нам не нужен суперкомпьютер, достаточно ноутбука и немного Python. Мы построим конвейер, который превращает биржевые данные в прогнозы.

Шаг 1. Данные.

Мы используем тот же набор данных, что и в предыдущем практикуме – исторические котировки акций Московской биржи. Это наш фундамент.

Шаг 2. Генерация признаков.

Нужно написать скрипт, который смотрит на последние 10–15 свечей и оценивает ситуацию, как человек. Мы выделяем три ключевых аспекта:

1. **Краткосрочный тренд (3 дня):** растем, падаем или стоим на месте?
2. **Среднесрочный контекст (7–14 дней):** подтверждает ли глобальное движение текущий рывок?

3. **Моментум и объем:** есть ли топливо для движения? Растут ли объемы на росте цены?

Шаг 3. Перевод на "человеческий" язык.

Это самый важный этап. Мы не скармливаем модели цифры "+1.5%" или "Vol=5000".

Мы пишем функцию, которая генерирует английский текст (так как большинство моделей лучше понимают английский).

Вместо строки в таблице мы получаем вот такую фразу:

"Price rising strongly, volume increasing, near resistance" (Цена сильно растет, объем увеличивается, рядом уровень сопротивления).

Или наоборот:

"Price falling, volume low, market is choppy" (Цена падает, объем низкий, рынок рваный).

```
 # Пример псевдокода генерации описания
def describe_market(trend, volume_change):
    description = []
    if trend > 0.02:
        description.append("price rising strongly")
    elif trend < -0.02:
        description.append("price falling sharp")

    if volume_change > 0:
        description.append("volume increasing")

    return ", ".join(description)
```

Шаг 4. Обучение модели.

Мы берем модель **DistilBERT**. Это "младший брат" огромных языковых моделей – он быстрый, легкий и отлично справляется с классификацией текстов.

Мы показываем ему тысячи таких сгенерированных фраз и говорим:

"Смотри, после фразы "цена растет на высоком объеме" на следующий день рынок чаще всего шел вверх".

Мы используем метод скользящего окна, чтобы все было честно и самое главное не было заглядывания в будущее. Модель учится на данных за год, пытается предсказать следующий месяц, затем забывает старое, учит новые данные и двигается дальше.

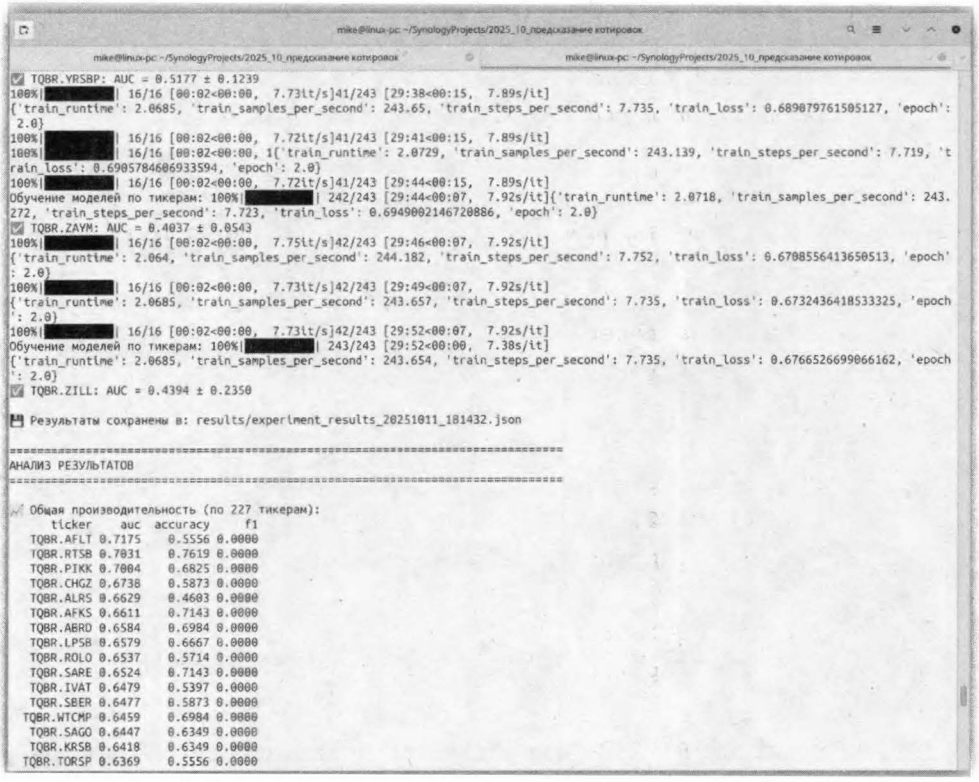


Рис. 9.15. ***Результат 227 бумаг в консоли***

Результаты: холодный душ или новая надежда?

Запустив этот сложный механизм на 200+ акциях, мы затаили дыхание.

Сможет ли искусственный интеллект, не видя ни одной цифры, а читая только наши текстовые "эссе" о рынке, предсказать движение цены?

Для оценки мы использовали метрику **AUC (Area Under Curve)**. Если объяснять просто:

- **AUC = 0.5** – это подбрасывание монетки. Модель не понимает ничего.
- **AUC = 1.0** – идеальный провидец.
- **AUC > 0.5** – есть хоть какая-то польза.

Итог нашего теста: средний AUC ≈ 0.53.

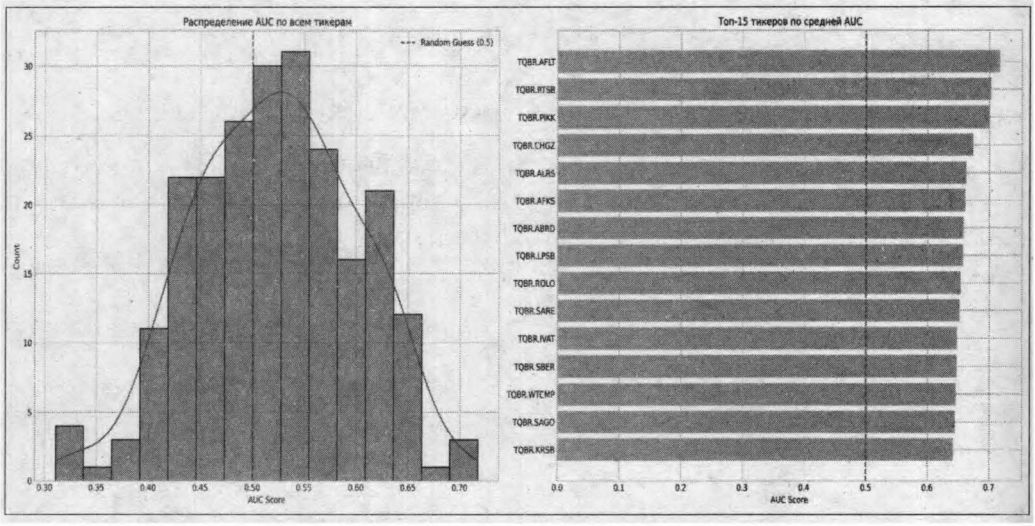


Рис. 9.16. ***Распределение метрики AUC по 227 акциям Мосбиржи***

Что это значит для нас? Торговать по такой модели прямо сейчас нельзя. Преимущество в 3% над случайностью слишком мало, чтобы покрыть комиссии биржи и проскальзывание. Вы не станете миллионером, запустив этот скрипт.

Но если посмотреть с другой стороны, результат **0.53** – это маленькая победа. Это доказательство того, что в *текстовом описании* рынка действительно содержится информация.

Модель, которая никогда в жизни не видела график, а только читала "слова трейдера", смогла угадывать направление движения чуть лучше, чем слепой случай.

На некоторых акциях (например, "Аэрофлот" или "ПИК") показатель доходил до **0.70**, что уже является заявкой на успех. На других (вроде "Полюс") модель откровенно путалась. Это говорит о том, что разные инструменты имеют разную "читаемость" для ИИ.

Подробнее со всеми кодами и исходными данными можно познакомиться в моём репозитории: *https://github.com/empenoso/llm-stock-market-predictor*

Главный вывод раздела

Зачем мы тратили время на этот эксперимент, если он не принес грааля?

Потому что это открывает дверь в будущее. Мы использовали крошечную модель DistilBERT и простейшие правила генерации текста. Представьте, что будет, если вместо простых фраз мы будем скармливать модели полные сводки новостей, отчеты аналитиков и макроэкономические показатели?

> Мы доказали главное: **финансовые данные можно превратить в язык, и ИИ способен этот язык понимать.**

Это был наш эксперимент. Собрали прототип на коленке, используя открытые библиотеки. Но мир технологий не стоит на месте. Пока мы играли с DistilBERT, в мире появились гиганты, способные на гораздо более глубокий анализ.

В следующей, десятой главе, мы отложим Python-скрипты для обучения нейросетей и обратимся к самому мощному инструменту, доступному сегодня каждому из нас. Мы узнаем, как использовать **ChatGPT** не для написания писем, а как полноценного партнера в написании кода, анализе Excel-таблиц и даже в создании тех самых стратегий, которые мы только что учились тестировать.

Заключение: вы сделали первый шаг от инвестора к исследователю

Мои поздравления, теперь вы не просто новичок-любитель, вы познакомились и научились применять системный подход.

Некоторые говорят, глядя на график что-то вроде "моей системе не нужен бэктекст, потому что я уже всё знаю и без него", – но как можно сравнить разные модификации одной и той же системы? Теперь у вас есть инструменты чтобы это сделать.

Давайте вспомним что нам помогало в этой главе:

- За пару кликов можно набросать идею в TradingView, чтобы отбросить откровенно слабые гипотезы.
- Можно не только скачать котировки с Московской Биржи, но и отбросить неликвидные бумаги.
- У нас есть готовый интерфейс для Python-библиотеки. В нём можно считать коэффициенты Сортино и строить тепловые карты оптимизации, чтобы не попасть в ловушку "идеальных параметров" (подгонки под кривую), а начать искать "зоны стабильности".
- Эксперимент с LLM показал, что даже "гуманитарный" подход к рынку – через описание словами, – имеет право на жизнь.

> Самый важный урок этой главы может показаться странным, но **отрицательный результат бэктеста – это тоже победа**. Если вы написали код, прогнали тест и увидели, что стратегия теряет деньги, вы не потерпели неудачу. Вы только что сэкономили свой капитал, который потеряли бы в реальной торговле, доверившись интуиции.

Теперь, когда у вас есть фундамент для проверки идей, возникает логичный вопрос: а как писать код для новых, более сложных стратегий? Но мы ведь живем в удивительное время.

В следующей, последней главе, мы узнаем, как превратить ChatGPT из простого собеседника в вашего персонального младшего разработчика, который возьмет на себя рутину написания кода, оставив вам самое интересное – генерацию идей.



Глава 10.

ChatGPT как ваш личный программист

Поздравляю! Вы уже проделали большой путь: научились укрощать Excel, заставили Python ходить в Интернет вместо вас, разобрались с учётом финансов и даже познакомились с компьютерным зрением. Вы уже сейчас умеете больше, чем 90% офисных сотрудников.

Но всегда быть одному сложно. Подумайте, сколько раз вы застревали на какой-нибудь глупой ошибке? Сколько раз гуглили, как обратиться к конкретной ячейке таблицы из Google Таблиц в Google Скриптах? Программирование, особенно в самом начале пути, может быть очень сложным.

Но эта 10 глава должна вам помочь и поменять правила игры. Мы больше не будем знакомиться с новыми библиотеками. В этой главе вы перестанете быть просто новичком-самоучкой. С этого момента у вас появляется личный ментор, стажер и эрудированный (хоть иногда и привирающий) коллега в одном лице – ChatGPT (и другие LLM).

Эта глава не про то, "как зарегистрироваться в OpenAI из России, получив смс на французский номер телефона". Она про смену роли. Вы больше не просто пишете код. Вы становитесь руководителем своего личного проекта.

Мы разберем:

- Почему "синдром чистого листа" больше не страшен.
- Что такое вайбкодинг и как написать сложный скрипт за 10 минут через диалог.
- Как заставить ИИ прочитать весь ваш проект целиком и найти ошибку, которую вы не видели.
- И главное – как не дать ИИ обмануть вас, выдумав несуществующую библиотеку.

Готовы нанять своего первого (и бесплатного) помощника?

10.1. От "я пишу код" к "я управляю написанием кода"

Держать в голове все методы **pandas**, параметры `requests.get()` или синтаксис сложных регулярных выражений – занятие скучное, неблагодарное и, возможно, совершенно лишнее в XXI веке, когда нет смысла демонстрировать память, а можно продемонстрировать результат. Раньше программист мог соревноваться с другим программистом, кто быстрее вспомнит нужный аргумент функции. Теперь соревнование идёт только с самим собой: насколько точно сможешь сформулировать задачу ИИ.

И здесь происходит самая важная смена парадигмы: мы перестаём быть "исполнителем кода". Мы становимся руководителями, менеджерами, прорабами – людьми, которые описывают, *что* нужно сделать, и проверяют, что ИИ сделал именно это.

Представьте строительную площадку. Раньше мы с утра сами брали мастерок, пачкали руки раствором и вручную укладывали каждый кирпич Python-кода. Ровно, аккуратно, по уровню – но медленно и с риском ошибиться уже на второй сотне строк. Теперь всё по-другому: у нас есть бригада, в которой ИИ – самый быстрый рабочий. Он не устает, не жалуется, не берёт отпуск и не говорит, что "забыл синтаксис". Он кладёт кирпичи по вашему чертежу. Но чертёж по-прежнему делаем мы.

И здесь появляется новая обязанность – проверять стены на отсутствие кривизны. Тот же ChatGPT может дать блестящее решение, а может слегка промахнуться, сделать не то, не там и не так. Это нормально. Мы становимся прорабами, управляющими качеством. Мы задаём направление, уточняем критерии, пересобираем формулировку – и код начинает выправляться.

Проблема чистого листа

У каждого новичка (да и не только новичка) есть знакомый ужас – мигающий курсор в новом файле. Пустой `script.py`. Пустая книга Excel. Кажется, что нужно знать всё сразу: и синтаксис, и библиотеки, и алгоритмы. Руки опускаются ещё до первой строки.

И вот здесь ИИ снимает страх пустого листа. Ему можно задать расплывчатое, нечёткое, даже полусырой запрос. "Мне нужен скрипт, который берёт CSV с продажами и строит график по городам". Пять секунд – и на экране появляется черновик. Конечно – такой запрос очень неточный, в нём нет контекста, но уже *есть*, с чем работать.

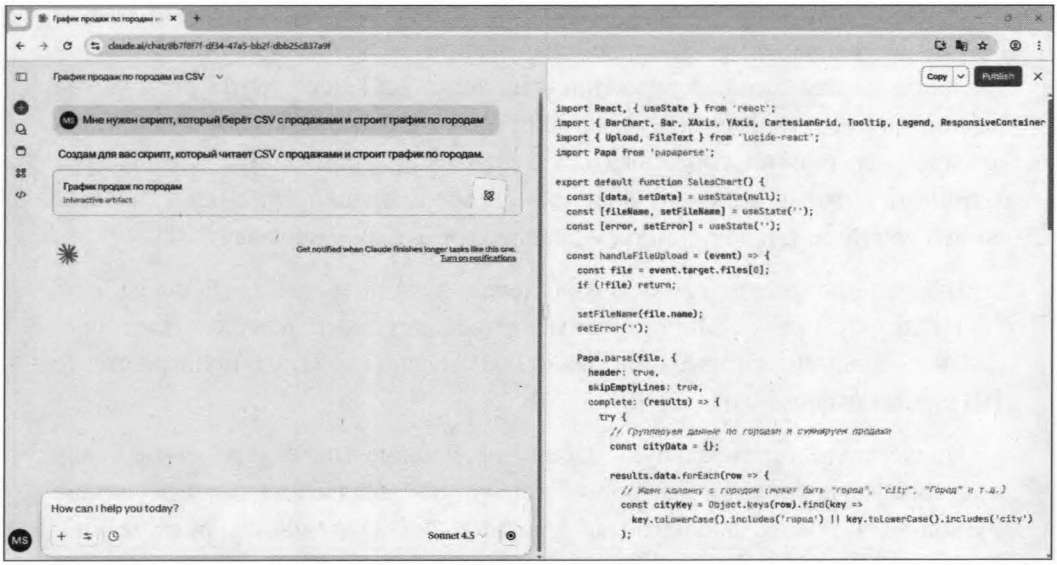


Рис. 10.1. ***Проблема пустого листа и первый черновик от ИИ***

Мы перестаем начинать с нуля – начинаем с черновика – а это психологически и технически в десять раз проще.

Простой пример: "я управляю", а не "я вспоминаю синтаксис"

Допустим, у нас задача: в Excel есть строка с адресами, разделёнными запятыми, и нам нужен текст после второй запятой. Раньше мы бы шли в Google, читали форумы, открывали StackOverflow, вспоминали, что такое ПСТР, как работает ПОДСТАВИТЬ.

Теперь всё иначе. Мы просто формулируем задачу:

```
"Извлеки текст после второй запятой в Excel".
```

И получаем готовую формулу:

```
=ПСТР(A1; НАЙТИ("^^"; ПОДСТАВИТЬ(A1; ","; "^^"; 2)) + 2; 999)
```

Никакого "помню-не помню", управляем процессом, а не карабкаемся по синтаксической стене.

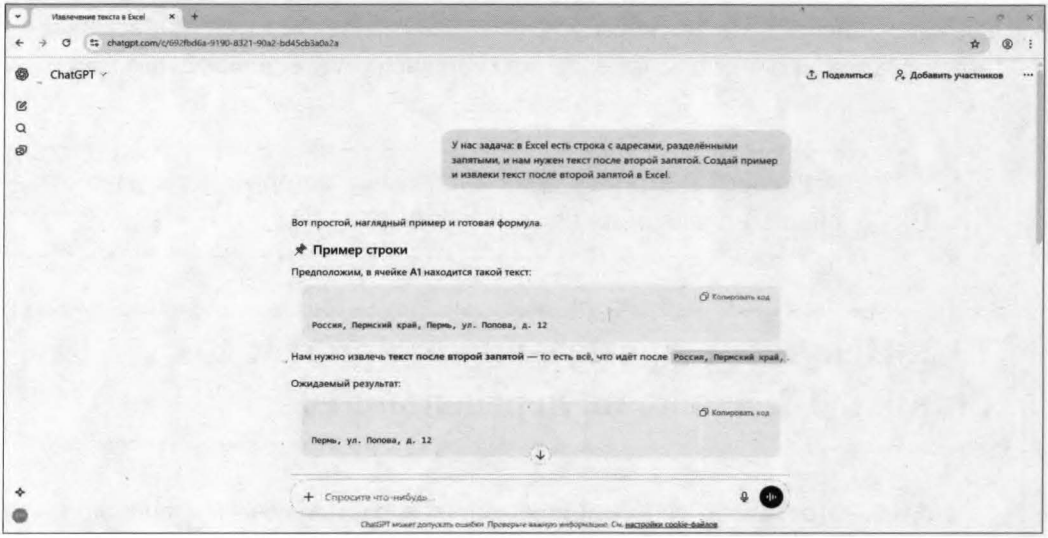


Рис. 10.2. ***Генерация Excel-формулы по запросу***

Более того, мы можем попросить объяснить механизм работы формулы человеческим языком, переписать её под Google Sheets, встроить в Power Query или сделать расширенную версию, учитывающую лишние пробелы. Такой уровень "гибкости без усилий" раньше требовал несколько лет практики. Теперь – грамотной постановки задачи.

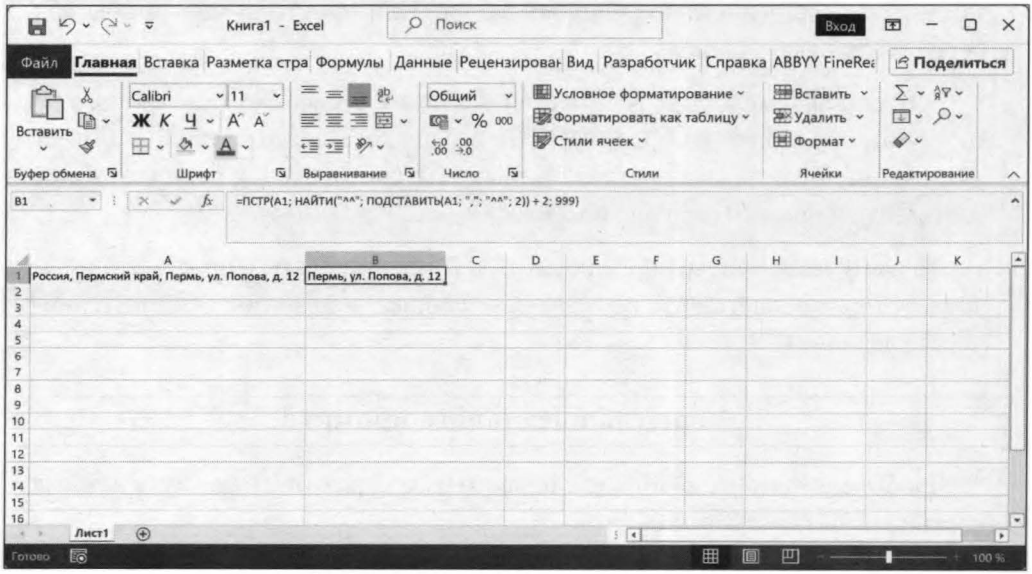


Рис. 10.3. ***Результат сгенерированной формулы***

До этого момента мы писали код как могли. Иногда как умели, иногда как получалось. С ИИ переходим в другой режим. Мы говорим: "что нужно", а не "как написать". Мы оцениваем результат и становимся людьми, которые могут реализовать идею за вечер, даже если вчера ещё боялись слова "регулярка".

ИИ теперь – это ещё один инструмент, который неразумно отрицать или не уметь им пользоваться.

10.2. Искусство промпт-инжиниринга: как ставить ТЗ, чтобы не переделывать

Но ИИ – это как стажер с феноменальной скоростью печати и энциклопедическими знаниями, но с полным отсутствием житейской интуиции. Если мы не скажем ему, что именно делать, он сделает то, что покажется ему наиболее вероятным ответом. А "вероятное" редко совпадает с "нужным".

Главная ошибка, которую можно совершить на старте это думать, что ИИ – телепат. Нам кажется, что фраза "исправь ошибку" понятна. Но для языковой модели это уравнение с сотней неизвестных. Какую ошибку? Синтаксическую? Логическую? Ошибку стиля? Или просто нужно, чтобы код перестал выдавать ошибку?

Есть старая поговорка: "Компьютер делает то, что вы ему прикажете, а не то, что вы от него хотите". С ChatGPT и его аналогами это правило работает аналогично. Если задача поставлена плохо (мусор на входе), результат будет соответствующим (мусор на выходе).

Поэтому надо учиться не "просить", а "проектировать запрос". Это и называется словосочетанием *промпт-инжиниринг*, но фактически это техническое задание (ТЗ).

Формула идеального промпта

Чтобы получать рабочий код с первого раза существует формула которая состоит из четырех компонентов.

Если вы пропустите один из них, модель начнет гадать, и скорее всего, не в вашу пользу.

1. Роль модели.

Мы задаем контекст экспертизы. Если мы не укажем роль, модель будет отвечать как "усредненный помощник". Однако часто нужен именно узкий специалист.

» *Плохо:* "Как мне это сделать?"

» *Хорошо:* "Действуй как Python-разработчик уровня Senior с опытом в автоматизации отчетности".

» *Почему это важно:* Задавая роль "Senior", мы намекаем, что нам не нужны объяснения, что такое переменная, но нужен чистый, оптимизированный код.

2. Контекст

ИИ не знает, что лежит у нас на рабочем столе. Он не видел наши предыдущие файлы. Мы должны описать ситуацию.

» *Пример:* "Я пишу скрипт для сортировки семейных фотографий. У меня есть папка, в которой 5000 файлов с названиями вида IMG_20230101_120000.jpg".

3. Задача

Четкое действие. Что именно надо сделать?

» *Пример:* "Напиши функцию, которая извлекает дату из имени файла и создает папки **Год/Месяц**, а затем перемещает туда файлы".

4. Ограничения и формат

Это "защита от дурака" (в хорошем смысле). Мы сразу отсекаем ненужные варианты решения.

» *Пример:* "Используй только стандартные библиотеки Python (os, shutil, datetime), не используй сторонние модули. Добавь подробные комментарии на русском языке к каждой строке кода".

Кажется, что это долго – писать такой длинный текст. Но мы сэкономим времени, которое потратили бы на уточнение и исправление галлюцинаций модели.

Разбираем на примере

Вспомним Главу 6, где мы создавали систему учета финансов. Допустим, данные лежат в CSV. Мы уже делали отчеты в Google Таблицах, но теперь хотим визуализировать расходы с помощью Python. Мы хотим видеть, куда уходят деньги, но при этом сами не помним, как работает библиотека **matplotlib**, а разбираться с её документацией в пятницу вечером нет никакого желания.

Рассмотрим два подхода к решению этой задачи.

Сценарий 1. Подход "Телепат" (плохой запрос)

Мы пишем в чат: *"Нарисуй график расходов"*.

Что происходит в "голове" у ИИ модели?

1. Каких расходов? У меня нет данных. Я их выдумаю.
2. Какой график? Линейный? Столбчатый? Круговой? Сделаю линейный, он проще.
3. На каком языке? Python? JavaScript? Excel?
4. Какие библиотеки использовать?

В итоге мы получаем код, который генерирует случайные числа и строит какой-то абстрактный график, который никак не применим к нашему файлу *expenses.csv*. Мы смотрим на это и расстраиваемся: "ИИ глупый, он меня не понял". Нет, это мы не дали ему инструкцию.

Сценарий 2. Подход "Главный инженер" (хороший запрос)

Теперь применим нашу формулу. Мы помним, что у нас есть реальный файл, и у нас есть конкретная проблема с круговыми диаграммами: если категорий слишком много, график превращается в нечитаемую секцию из узких секторов. Нам нужно решить эту проблему логически, еще до написания кода.

Составляем промпт:

> **(Роль)** Действуй как аналитик данных с экспертизой в визуализации на Python.
>
> **(Контекст)** У меня есть CSV-файл *my_expenses.csv*. В нем есть колонки: Дата, Сумма, Счёт, Категория, Описание.
>
> **(Задача)** Напиши скрипт на Python, используя библиотеку **matplotlib**. Нужно построить круговую диаграмму расходов по категориям за текущий месяц.

(Ограничения и уточнения)

1. Данные нужно сгруппировать по категориям, просуммировав Сумма.
2. Очень важно: сделай так, чтобы категории, доля которых составляет менее 5% от общей суммы, автоматически объединялись в одну общую группу "Прочее". Иначе график получается перегруженным и нечитаемым.
3. Подпиши проценты прямо на графике.
4. Код должен быть готов к запуску, с комментариями на русском языке.

Чувствуете разницу? Мы не просто просим написать код, мы закладываем в него бизнес-логику. ИИ знает синтаксис, знает, как объединить массивы, но только мы знаем, что диаграмма из 40 кусков – это плохо.

Получив такой запрос, ИИ выдаст решение, которое:

1. Прочитает CSV.
2. Отфильтрует даты (потому что мы упомянули "текущий месяц", он добавит datetime.now()).
3. Рассчитает проценты.
4. Напишет цикл **if**, который проверит каждую категорию: если меньше 5% – кидаем в копилку "Прочее".
5. Нарисует круг.

Мы получаем инструмент, который сразу можно использовать. Если вдруг код выдаст ошибку (например, не найдет файл), мы просто копируем текст ошибки и пишем ИИ: "Я получил ошибку: [текст ошибки]. Исправь". И он исправит.

Итеративное улучшение: диалог, а не монолог

Редко когда удается получить идеал с первой попытки, даже с хорошим промптом. И это нормально. Работа с ИИ – это пинг-понг.

> – Код работает, но подписи на диаграмме накладываются друг на друга.
>
> – Окей, – *отвечает ИИ,* – давай добавим параметр *labeldistance* или вынесем легенду сбоку.
>
> – Мне не нравятся цвета, они слишком бледные.
>
> – Без проблем, используем палитру tab20 или зададим свои цвета.

Мы "докручиваем" решение. В этом процессе мы учимся. Глядя на то, как ИИ меняет код, чтобы раздвинуть подписи, мы запоминаем: "Ага, в **matplotlib** за это отвечает bbox_to_anchor". Так обучение происходит не через зубрежку учебника, а через решение собственной насущной задачи.

10.3. Вайбкодинг на практике: пишем скрипт с нуля за 10 минут

Пора перейти от теории к живому вайбкодингу. В этом разделе мы создадим маленькую утилиту, которая решает проблему количества файлов скриншотов.

Практикум: создаём утилиту для чистки рабочего стола от скриншотов

Каждый, кто работает на компьютере, хотя бы раз видел такую картину: рабочий стол превращается в хламовник PNG-файлов со скриншотами:

```
Снимок экрана 2025-12-02 171232.png
Снимок экрана 2025-11-29 095029.png
Снимок экрана 2025-11-27 203154.png
...
```

Но мы начинаем не с кода, а с постановки задачи. Формулируем промпт по нашей четырёхшаговой схеме:

> **Роль**: "Действуй как Python-разработчик с опытом автоматизации файловых операций на Windows."
>
> **Контекст**: "У меня в папке *c:\Users\Muxaил\Pictures\Screenshots* лежат десятки PNG-скриншотов вида "Снимок экрана YYYY-MM-DD HHMMSS.png"."
>
> **Задача**: "Напиши скрипт, который находит все такие файлы и раскладывает их в подпапки *Год/Месяц*."
>
> **Ограничения**: "Используй только стандартные библиотеки **os**, **pathlib** и **shutil**. Добавь комментарии."

Мы отправляем такой запрос – и через пару секунд получаем стартовый черновик.

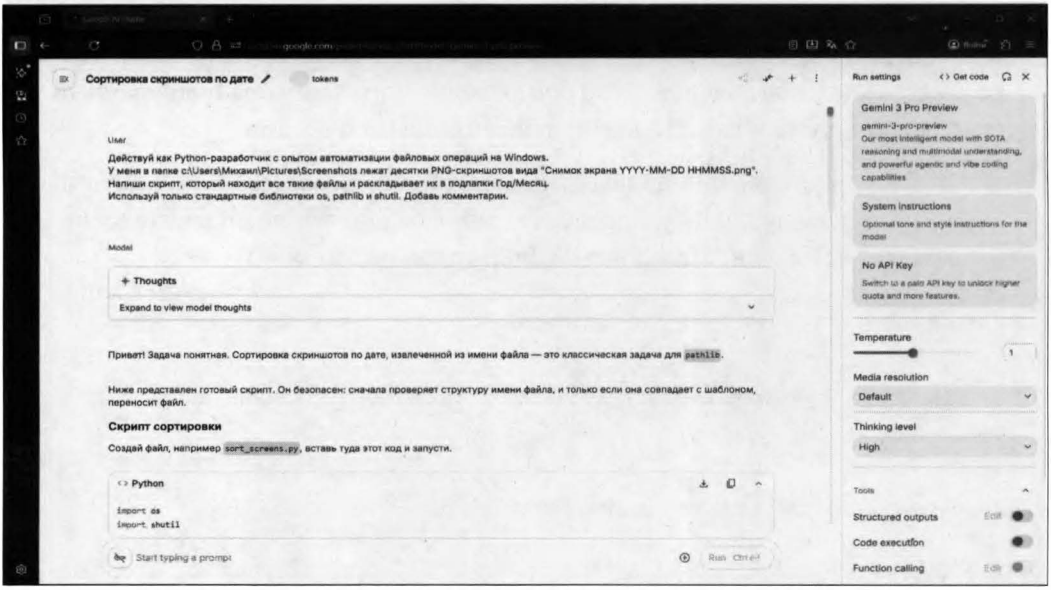


Рис. 10.4. ***Первый черновик скрипта для сортировки скриншотов***

В этот момент мы совершаем ключевой сдвиг парадигмы: мы не корректируем код вручную, а корректируем задачу.

Смотрим на код, который прислал ИИ, проверяем логику: скрипт проходит по файлам, парсит дату, создаёт подпапки, переносит PNG-файлы.

Этот стиль работы и есть вайбкодинг: быстрая, гибкая разработка через разговор. Чаще всего он приводит к такому эффекту, который раньше был доступен только опытным разработчикам: можно начинать создавать инструменты быстрее, чем успеваем о них мечтать.

10.4. Отладка и объяснение: если код не работает

Представим: у вас есть проект – возможно, написанный вами же больше года назад, а может, и скачанный с GitHub. Он состоит из десятков файлов, и вы уже и не вспомните, как там всё устроено. Вы знаете, что код работает, но не понимаете – как именно.

Для этого можно собрать всё содержимое проекта в один текстовый файл и передать его модели с большим контекстным окном. Это особенно актуально для современных LLM: некоторые из них принимают до 1 миллиона токенов. Для справки: 1 токен – это примерно 0,75 символа на русском языке, то есть миллион токенов ≈ 750 000 знаков. Этого хватит на полноценный небольшой проект – даже со всеми комментариями и логами.

Чтобы упростить сбор данных, мы подготовим небольшой bash-скрипт. Он обходит текущую папку, определяет тип каждого файла по расширению и сохраняет всё в один Markdown-файл. Вот как он выглядит:

```
#!/bin/bash

# Скрипт для преобразования всех файлов (включая вложенные папки) в
текстовый вид

# Функция для определения языка файла по расширению
get_language() {
    local file="$1"
    local ext="${file##*.}"

    case "$ext" in
        py) echo "python" ;;
```

```
        js) echo "javascript" ;;
        sh|bash) echo "bash" ;;
        yaml|yml) echo "yaml" ;;
        json) echo "json" ;;
        xml) echo "xml" ;;
        html|htm) echo "html" ;;
        sql) echo "sql" ;;
        md|markdown) echo "markdown" ;;
        txt|log) echo "text" ;;
        *) echo "text" ;;
    esac
}

# Проверяем, есть ли файлы в текущей директории
if [ ! "$(ls -A . 2>/dev/null)" ]; then
    echo "В текущей папке нет файлов для обработки."
    exit 1
fi

# Создаем или очищаем выходной файл
output_file="all_files_content.md"
> "$output_file"

# Добавляем заголовок с текущей датой и временем
current_time=$(date '+%Y-%m-%d, %H:%M')
echo "Ниже мои файлы по состоянию на $current_time :" >> "$output_file"
echo >> "$output_file"

echo "Начинаю преобразование файлов в текстовый вид..."
echo "Результат будет сохранен в файл: $output_file"
echo

# Счетчик обработанных файлов
count=0

# Обходим все файлы рекурсивно, используя find
while IFS= read -r -d '' file; do
    # Пропускаем сам выходной файл
```

```
    if [ "$file" = "./$output_file" ]; then
        continue
    fi

    # Пропускаем сам скрипт
    if [[ "$(basename "$file")" == "+files_to_text.sh" ]] || [[
"$(basename "$file")" == "+files_to_text.txt" ]]; then
        continue
    fi

    # Проверяем, можно ли прочитать файл
    if [ ! -r "$file" ]; then
        echo "Предупреждение: Файл '$file' недоступен для чтения.
Пропускаю..."
        continue
    fi

    # Проверяем, является ли файл текстовым
    if file "$file" | grep -qE 'text|empty|JSON|XML|HTML|script'; then
        # Записываем заголовок с относительным путем
        echo "Файл $file:" >> "$output_file"

        # Пустая строка после имени файла
        echo >> "$output_file"

        # Определяем язык для подсветки синтаксиса
        lang=$(get_language "$file")
        echo "\'\'\'$lang" >> "$output_file"

        # Записываем содержимое файла
        cat "$file" >> "$output_file"

        # Закрываем блок кода с новой строки
        echo '' >> "$output_file"
        echo ''''' >> "$output_file"

        # Добавляем пустую строку для разделения
        echo >> "$output_file"
```

```
        # Увеличиваем счетчик
        ((count++))

        echo "Обработан файл: $file (язык: $lang)"
    else
        echo "Пропущен бинарный файл: $file"
    fi
done < <(find . -type f -print0)

echo
if [ $count -eq 0 ]; then
    echo "Не найдено файлов для обработки."
    rm -f "$output_file"
else
    echo "Завершено! Обработано файлов: $count"
    echo "Результат сохранен в файл: $output_file"
fi
```

Запускаем его в корне проекта – и через несколько секунд получаем файл *all_files_content.md*, в котором лежит вся структура проекта в читаемом виде.

Каждый файл отмечен заголовком, его содержимое оформлено как блок кода.

Теперь мы можем загрузить этот файл в модель с большим контекстным окном (например, Qwen3-MAX или Gemini 3.5 Pro) и задать один из следующих запросов:

- "Объясни, как работает этот проект, шаг за шагом".
- "Почему у меня выскакивает ошибка `KeyError: 'temperature'` при запуске *weather_bot.py*?"
- "Как добавить поддержку уведомлений в Telegram в этот парсер?"

Модель, имея полный контекст, не гадает, а анализирует реальный код. Она видит, где определены переменные, как они передаются между функциями, где лежат конфиги. Это как если бы вы сели рядом с опытным коллегой и показали ему весь проект сразу.

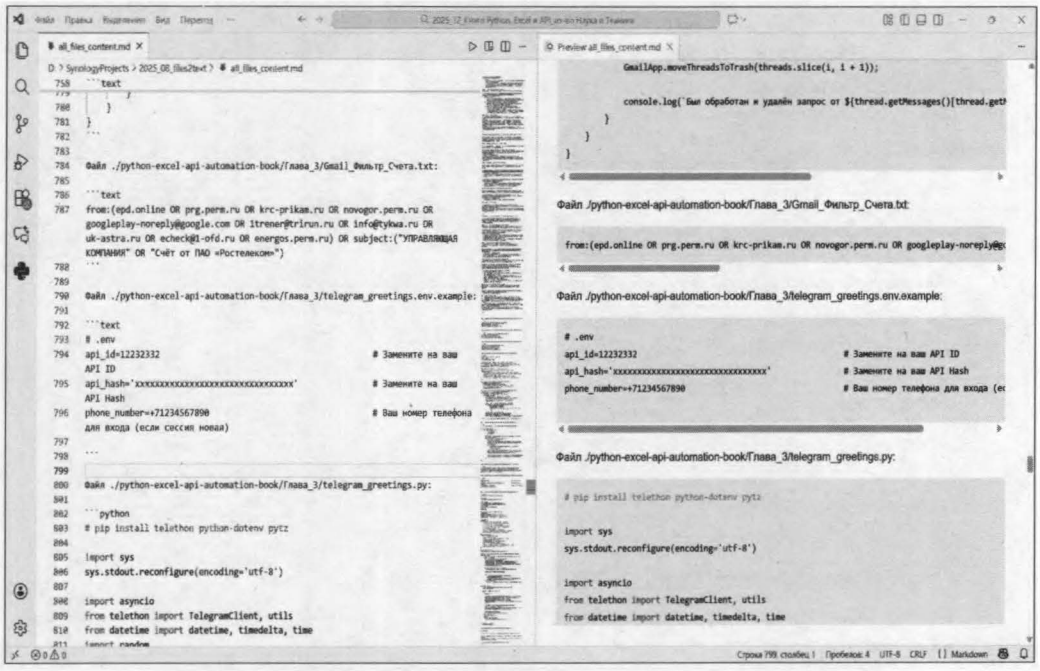


Рис. 10.5. ***Структура проекта, собранная в один Markdown-файл***

Конечно, не все модели поддерживают такой объём. Но даже если вы работаете с ChatGPT (где контекст короче), вы всё равно можете использовать этот подход частями: сначала загрузить *main.py*, потом *utils.py*, потом *config.yaml* – и постепенно выстраивать понимание.

А теперь ещё один важный момент: мы не просто просим объяснить – мы просим объяснять. Например: "Объясни логику скрипта, как будто я бухгалтер, который только начал учить Python. Без жаргона, с примерами из Excel".

ИИ адаптирует объяснение под ваш уровень. Это превращает сложный технический код в понятную историю с логикой, а не в набор команд.

Более того, если код работает, но вы не понимаете почему, ИИ может пройтись по нему построчно:

"Комментируй каждую строку в *main.py*, объясняя, зачем она нужна и что произойдёт, если её убрать".

Вы получаете не просто рабочий инструмент, а интерактивный учебник, написанный под ваш конкретный проект. Так вы учите Python не абстрактно, а на своём коде, решая свои задачи.

В этом и есть суть новой автоматизации: можно не боятся сложности, потому что у нас есть союзник. И этот союзник работает 24/7, не требует кофе и готов объяснить регулярку восьмой раз подряд.

10.5. Безопасность и галлюцинации: где ИИ врет, а где опасен

Это – один из самых важных разделов во всей главе. Мы уже привыкли использовать ИИ как сверхбыстрого помощника-разработчика, но вместе с этим в игру входят новые риски. И наша задача – научиться работать так, чтобы не подставить ни себя, ни данные, ни компанию, ни клиентов.

Правило №1: никогда не отправляем в чат реальные API-ключи, пароли и токены!

Мы можем работать с банковскими API, облачными сервисами, Telegram-ботами, домашней автоматизацией. Все эти системы используют секретные ключи.

Но ИИ-модели не предназначены для хранения секретов, мы не можем проверить, как именно обрабатываются данные на стороне сервиса. Поэтому базовое правило: ничего секретного в чат не уходит.

Мы всегда делаем замену:

```
# Плохо - никогда так не делаем
TINKOFF_TOKEN = "t.iasdklj32h4kjh23kjh234"

# Хорошо - так делаем всегда
TINKOFF_TOKEN = "TOKEN_PLACEHOLDER"
```

На всякий случай мы можем добавить себе привычку: прежде чем отправлять код в чат, пробежаться глазами и заменить всё, что выглядит как

пароль, OAuth-токен, секретная строка, ID приложения, приватная ссылка. Такой подход автоматически защищает от любых утечек, даже случайных.

Галлюцинации: когда ИИ выдумывает то, чего не существует

Пожалуй, самое раздражающее качество больших языковых моделей – способность уверенно придумывать библиотеки, функции, параметры, которых не было и нет.

Мы спрашиваем:

"Сделай запрос к API Тинькофф Инвестиций"

ИИ отвечает:

```
from tinkoff_magic_api import Client
client = Client(token=TINKOFF_TOKEN)
print(client.get_positions())
```

Проверяем, существует ли библиотека или метод.

Алгоритм простой:

1. Если код не запускается – смотрим на ошибку.
2. Если импорт не проходит – проверяем pip:
3. `pip install tinkoff_magic_api`

   Если не находится – значит, ИИ придумал.
4. Сверяемся с официальной документацией API или библиотеки.

Это занимает меньше минуты, но избавляет нас от попыток "починить" код, которого в природе не существует.

Чтобы минимизировать галлюцинации можно использовать правильные формулировки:

"Используй только стандартные библиотеки Python".

> "Используй только **pandas** и **requests**".
>
> "Не придумывай новые библиотеки, используй официальную документацию API Тинькофф брокер".

Такие ограничения – мощный инструмент. Мы словно ставим ИИ рамки, и он охотно в них работает.

Устаревшие данные: модели помнят прошлое, но не знают будущего

Большинство ИИ-моделей обучены на данных на определённую дату. Даже если она свежая, библиотеки обновляются быстрее.

И вот мы получаем код:

```
df = pd.read_csv("file.csv", error_bad_lines=False)
```

А в реальности этот параметр уже удалён. Pandas выбрасывает ошибку – и мы удивляемся: "Почему ИИ написал устаревший код?"

Ответ прост: он опирается на старую версию документации.

Чтобы работать безопасно, мы вводим правило:

Мы всегда проверяем код на своей версии библиотек

Можно прямо в промпте указать версии:

> `Работаем на pandas 2.2.2. Пиши код под неё, избегая устаревших параметров.`

Ещё лучше – написать:

> `Вот моя ошибка. Исправь код под актуальную версию библиотеки.`

ИИ очень хорошо исправляет несовместимость версий, если мы ему показываем конкретный пример.

В этом разделе мы выстраиваем важный навык: мы работаем с ИИ как с мощным инструментом, но понимаем ограничения.

Мы управляем процессом: фильтруем, уточняем, ставим рамки, проверяем актуальность. И в результате получаем не просто код, а безопасный, современный и работающий код – который действительно можно использовать в реальных проектах.

Заключение

Мы в конце не только этой главы, но и всей книги, а также, возможно, и определённого этапа вашей жизни – который, надеюсь, позволит по-другому взглянуть на цифровую рутину в своих действиях.

В этой главе мы разбирались как превратить ИИ из модной игрушки в цифрового помощника. Учились писать промпты по формуле "Роль + Контекст + Задача + Ограничения" и поняли, что можно доверять модели, но вот проверять её это именно наша обязанность.

Но самая главная мысль, которую я бы хотел донести выходит за рамки написания кода.

Автоматизация – это не про Python и не про Excel, и даже не про ChatGPT.

Автоматизация – это, прежде всего, мышление.

Ведь инструменты всегда будут меняться. Вчера это макросы VBA в Excel, сегодня Python, а завтра нейросетевой агент, который по вашей голосовой команде напишет целое приложение.

А ещё синтаксис меняется, но вот ваш навык смотреть на рутинную задачу и видеть в ней алгоритм – останется с вами навсегда.

Я считаю, что вы научились главному – не мириться с тем что так делали всегда, что так принято, вы научились видеть возможности автоматизации и делать свою работу лучше, быстрее и без ошибок.

ИИ теперь как экзоскелет. Он не делает работу за вас, но позволяет увеличить скорость и точность. А ещё снимает барьер входа в сложные технологии и освобождает время для жизни.

Используйте эту силу не чтобы переделать больше работы, а чтобы освободить время для творчества и самой жизни, ведь время — это единственный ресурс, который нельзя накопить в базе данных.

Но его можно сберечь, если вовремя сказать компьютеру: "**Действуй как программист. Автоматизируй это!**".

Удачи в ваших проектах!

P.S.

Проекты, которые создавались в ходе книги, доступны в репозитории по ссылке ниже:

https://github.com/empenoso/python-excel-api-automation-book

Также материалы из репозитория доступны на виртуальном диске, который можно скачать на сайте издательства в разделе "Материалы к книгам".



Список использованных источников информации

Список использованных источников информации:

Введение

1. Шардин М. Авторский сайт. Электронный ресурс. URL: https://shardin.name/ (дата обращения: 18.01.2026).
2. Smart-Lab. Статья «Google Таблицы против Excel: неочевидное преимущество для инвестора и трейдера». Электронный ресурс. URL: https://smart-lab.ru/mobile/topic/1219831/ (дата обращения: 18.01.2026).

Глава 1

1. Т—Ж. Статья «Как автоматически скачивать котировки в Excel». Электронный ресурс. URL: https://t-j.ru/guide/excel-kotirovki/ (дата обращения: 18.01.2026).
2. Хабр. Статья «Расчет стоимости оборудования через Google Apps Script». Электронный ресурс. URL: https://habr.com/ru/articles/837616/ (дата обращения: 18.01.2026).
3. Google. Google Sheets API Documentation. Электронный ресурс. URL: https://developers.google.com/sheets/api (дата обращения: 18.01.2026).

Глава 2

1. Python Software Foundation. pathlib — Object-oriented filesystem paths. Электронный ресурс. URL: https://docs.python.org/3/library/pathlib.html (дата обращения: 18.01.2026).

2. Google. Обзор Google Apps Script. Электронный ресурс. URL: https://developers.google.com/apps-script?hl=ru (дата обращения: 18.01.2026).

3. Хабр. Статья «Как я научил бухгалтерию составлять договора дарения со скоростью 1 договор в 4 секунды». Электронный ресурс. URL: https://habr.com/ru/articles/843488/ (дата обращения: 18.01.2026).

Глава 3

1. Хабр. Статья «Автоответчик на запросы доступа к файлам в гугл почте». Электронный ресурс. URL: https://habr.com/ru/articles/739168/ (дата обращения: 18.01.2026).

2. Хабр. Статья «Как бесплатно разослать всем членам ТСЖ персональное сообщение о завтрашней уборке снега через WhatsApp». Электронный ресурс. URL: https://habr.com/ru/articles/876216/ (дата обращения: 18.01.2026).

3. Хабр. Статья «Как автоматизировать свою переписку с девушкой в Telegram при помощи Python». Электронный ресурс. URL: https://habr.com/p/874412/ (дата обращения: 18.01.2026).

4. Хабр. Статья «Как поддерживать видимость социальной жизни, если ты лентяй». Электронный ресурс. URL: https://habr.com/ru/articles/871966/ (дата обращения: 18.01.2026).

5. Telegram Bot API. Электронный ресурс. URL: https://core.telegram.org/bots/api (дата обращения: 18.01.2026).

Глава 4

1. Home Assistant. Официальная документация. Электронный ресурс. URL: https://www.home-assistant.io/installation/ (дата обращения: 18.01.2026).

2. Хабр. Статья «Как я переделал освещение в подъезде на два контура и сэкономил 200 т.р. не делая ремонт, а используя умный дом». Электронный ресурс. URL: https://habr.com/ru/articles/855864 (дата обращения: 18.01.2026).

3. Хабр. Статья «Как заставить рулонные шторы поумнеть - дешевый способ без 220В». Электронный ресурс. URL: https://habr.com/ru/articles/811681/ (дата обращения: 18.01.2026).

4. Хабр. Статья «Как вернуть себе потерянные (или украденные из-за санкций?) умные функции стиральной машины Bosch». Электронный ресурс. URL: https://habr.com/ru/articles/783066/ (дата обращения: 18.01.2026).

5. Хабр. Статья «Самая дешевая заводская панель управления умным домом, которая отлично работает». Электронный ресурс. URL: https://habr.com/ru/articles/777376/ (дата обращения: 18.01.2026).

6. GitHub. Репозиторий «doorcam-face-report». Электронный ресурс. URL: https://github.com/empenoso/doorcam-face-report (дата обращения: 18.01.2026).

7. Хабр. Статья «Экологичное расставание с Алисой: строим полностью локальный и приватный голосовой ассистент». Электронный ресурс. URL: https://habr.com/ru/articles/930340/ (дата обращения: 18.01.2026).

8. Т—Ж. Статья «Что такое Home Assistant — как настроить умный дом без подключения к облаку». Электронный ресурс. URL: https://t-j.ru/guide/local-smart-home/ (дата обращения: 18.01.2026).

9. Хабр. Статья «Как обновлять информацию о фильмах и сериалах на медиацентре Kodi бесплатно и на русском языке». Электронный ресурс. URL: https://habr.com/ru/articles/773620/ (дата обращения: 18.01.2026).

10. Хабр. Статья «Анализ лиц с домофона: dlib+CUDA на Ubuntu — готовый скрипт». Электронный ресурс. URL: https://habr.com/ru/articles/935098/ (дата обращения: 18.01.2026).

11. Хабр. Статья «Паранойя безопасности против здравого смысла: чиним Home Assistant, который окирпичился без интернета». Электронный ресурс. URL: https://habr.com/ru/articles/983106/ (дата обращения: 18.01.2026).

Глава 5

1. Хабр. Статья «Отображение возраста контактов в Google Календаре». Электронный ресурс. URL: https://habr.com/p/969242/ (дата обращения: 18.01.2026).

2. Хабр. Статья «Вычисление отсрочки старта приготовления каши на завтра». Электронный ресурс. URL: https://habr.com/ru/articles/833648/ (дата обращения: 18.01.2026).

3. Хабр. Статья «Поиск ближайших любительских соревнований». Электронный ресурс. URL: https://habr.com/ru/articles/825508/ (дата обращения: 18.01.2026).

4. Хабр. Статья «Как сканировать двусторонние документы на одностороннем сканере». Электронный ресурс. URL: https://habr.com/p/721846/ (дата обращения: 18.01.2026).

5. GitHub. Репозиторий « empenoso/Double-sided-pages». Электронный ресурс. URL: https://github.com/empenoso/Double-sided-pages (дата обращения: 18.01.2026).

6. Хабр. Статья «Как сканировать многостраничные двухсторонние документы, если под рукой только обычный сканер с автоподачей». Электронный ресурс. URL: https://habr.com/ru/articles/721846/ (дата обращения: 18.01.2026).

7. Хабр. Статья «Когда менять шины: как Python и открытые API помогут избежать ошибок». Электронный ресурс. URL: https://habr.com/ru/articles/898174/ (дата обращения: 18.01.2026).

8. Смартлаб. Статья «Жену не бесит, бывшая не звонит: как я автоматизировал родительство после развода». Электронный ресурс. URL: https://smart-lab.ru/blog/1174812.php (дата обращения: 18.01.2026).

9. Хабр. Статья «Как убрать пустые оборотные страницы из PDF». Электронный ресурс. URL: https://habr.com/ru/articles/733754/ (дата обращения: 18.01.2026).

Глава 6

1. Книга. Шагабутдинов Ренат Магия таблиц. 100+ приемов ускорения работы в Excel (и немного в Google Таблицах) – МИФ, 2025 г.

Глава 7

1. GitHub. Репозиторий «empenoso/moex2excel_BID_OFFER». Электронный ресурс. URL: https://github.com/empenoso/moex2excel_BID_OFFER/ (дата обращения: 18.01.2026).

2. GitHub. Репозиторий « empenoso/moex-bond-search-and-analysis». Электронный ресурс. URL: https://github.com/empenoso/moex-bond-search-and-analysis (дата обращения: 18.01.2026).

3. Хабр. Статья «Почему Google Таблицы лучше Excel для частного инвестора». Электронный ресурс. URL: https://habr.com/ru/articles/957676/ (дата обращения: 18.01.2026).

4. Т—Ж. Статья «Как я создал собственную гугл-таблицу для учета капитала». Электронный ресурс. URL: https://t-j.ru/investment-report/ (дата обращения: 18.01.2026).

5. Хабр. Статья «Как «взломать» сайт Мосбиржи и получать лучшие BID и OFFER в Excel». Электронный ресурс. URL: https://habr.com/ru/articles/927238/ (дата обращения: 18.01.2026).

6. Хабр. Статья «Автоматизируем учёт облигаций: как избавиться от рутины». Электронный ресурс. URL: https://habr.com/ru/articles/882608/ (дата обращения: 18.01.2026).

7. Московская Биржа. Информационно-статистический сервер (ИСС / ISS). Электронный ресурс. URL: https://www.moex.com/a2193 (дата обращения: 18.01.2026).

8. Книга Тимофей Мартынов Механизм трейдинга. Как построить бизнес на бирже – Эксмо, 2022 г.

9. Книга УК Арсагера Заметки в инвестировании, 6-е издание – Виктория, 2024 г.

Глава 8

1. Хабр. Статья «Как я улучшил свой пользовательский опыт взаимодействия с фитнес клубом World Class найдя их API». Электронный ресурс. URL: https://habr.com/ru/articles/838322/ (дата обращения: 18.01.2026).

2. GitHub. Репозиторий «LynxReport». Электронный ресурс. URL: https://github.com/empenoso/LynxReport (дата обращения: 18.01.2026).

3. Хабр. Статья «Удаляем незваных подписчиков из своего Telegram-канала». Электронный ресурс. URL: https://habr.com/ru/articles/933128/] (https://habr.com/ru/articles/933128/) (дата обращения: 18.01.2026).

4. Хабр. Статья «Как я научился оценивать популярность статей через парсинг показателей сайтов». Электронный ресурс. URL: https://habr.com/ru/articles/871234/ (дата обращения: 18.01.2026).

5. Хабр. Статья «Как сделать своё онлайн портфолио и продвинуть его на первое место в поиске без бюджета». Электронный ресурс. URL: https://habr.com/ru/articles/742472/ (дата обращения: 18.01.2026).

6. Книга Бухарев Р.С. Хакинг на Python – Наука и Техника, 2025 г.

Глава 9

1. Смартлаб. Статья «Отвечаю на критику: бэктест линейной регрессии (из S&C 2007 года) на фьючерсах MOEX в 2025. Код и результаты!». Электронный ресурс. URL: https://smart-lab.ru/mobile/topic/1153389/ (дата обращения: 18.01.2026).

2. GitHub. Репозиторий «empenoso/SilverFir-TradingBot_backtesting». Электронный ресурс. URL: https://github.com/empenoso/SilverFir-TradingBot_backtesting (дата обращения: 18.01.2026).

3. GitHub. Репозиторий «empenoso/backtrader-quickstart-template». Электронный ресурс. URL: https://github.com/empenoso/backtrader-quickstart-template (дата обращения: 18.01.2026).

4. GitHub. Репозиторий «empenoso/llm-stock-market-predictor». Электронный ресурс. URL: https://github.com/empenoso/llm-stock-market-predictor (дата обращения: 18.01.2026).

5. Смартлаб. Статья «Как я решил вернуться в алготрейдинг после 10 летнего перерыва». Электронный ресурс. URL: https://smart-lab.ru/mobile/topic/1068045/ (дата обращения: 18.01.2026).

6. Смартлаб. Статья «Робот, который живёт в стене: мой опыт автоматизации торговли на Python». Электронный ресурс. URL: https://smart-lab.ru/mobile/topic/1156550/ (дата обращения: 18.01.2026).

7. Смартлаб. Статья «Прокачай свой TradingView: введение в мир Pine Script». Электронный ресурс. URL: https://smart-lab.ru/blog/1141546.php (дата обращения: 18.01.2026).

8. Смартлаб. Статья «Научить искусственный интеллект думать как трейдер – это утопия или начало нового подхода?». Электронный ресурс. URL: https://smart-lab.ru/blog/1216966.php (дата обращения: 18.01.2026).

9. Книга Д. Бакалов Программируем финансы на Python: криптовалюта, биржа, торговые и Телеграм боты – Наука и Техника, 2025 г.

10. Книга Стефан Янсен Машинное обучение для алгоритмической торговли на финансовых рынках. Практикум – BHV-СПб, 2020.

11. Книга Маркос Лопез де Прадо Машинное обучение: алгоритмы для бизнеса – ИД «Питер», 2019.

12. Книга Zihao Zhang, Stefan Zohren Deep Learning in Quantitative Trading – Cambridge University Press, 2025.

Глава 10

1. Google. Брошюра Prompting Guide 101 – сборник промтов, отобранных Google на основе анализа самых успешных запросов пользователей. Электронный ресурс. URL: https://services.google.com/fh/files/misc/gemini-for-google-workspace-prompting-guide-101.pdf (дата обращения: 18.01.2026).

https://ru.freepik.com/icon/data-transformation_17365209

Издательство «Наука и Техника» выпускает книги более 25 лет!

Уважаемые авторы!

Приглашаем к сотрудничеству по созданию книг
по **IT-технологиям, электронике, электротехнике, медицине, педагогике.**

Наши преимущества:

- являемся одним из ведущих технических издательств страны;
- выпускаем книги большими тиражами, что положительно влияет на гонорар авторов;
- регулярно переиздаем тиражи, автоматически выплачивая гонорар за *каждый* тираж;
- применяем индивидуальный подход в работе с каждым автором;
- работаем профессионально: от корректуры до авторских дизайн-проектов;
- проводим политику доступной цены;
- имеем собственные каналы сбыта: от федеральных сетей, крупнейших книжных магазинов РФ, ведущих маркетплейсов OZON, Wildberries, Яндекс-Маркет и др. до ведущих библиотек вузов, ссузов.

Ждем Ваши предложения:

- **тел. (812) 412-70-26**
- **эл. почта: nitmail@nit.com.ru**

Будем рады сотрудничеству!

Для заказа книг:

➢ **интернет-магазин: nit.com.ru**
- более 3000 пунктов выдачи на территории РФ, доставка 3–5 дней
- более 300 пунктов выдачи в Санкт-Петербурге и Москве, доставка 1–2 дня
- тел. (812) 412-70-26
- эл. почта nitmail@nit.com.ru

➢ **магазин издательства:** г. Санкт-Петербург, пр. Обуховской обороны, д. 107
- метро Елизаровская, 200 м за ДК им. Крупской
- ежедневно с 10.00 до 18.00
- справки и заказ: тел. (812) 412-70-26

➢ **книжные сети и магазины**

«Читай-город» - сеть магазинов	тел. +7 (495) 424-84-44
«Буквоед» - сеть магазинов	тел. +7 (812) 601-0-601
Московский дом книги – сеть магазинов	тел. +7 (495) 789-35-91
ТД «БиблиоГлобус»	тел. +7 (495) 781-19-12
«Амиталь» — сеть магазинов	тел. +7 (473) 223-00-02
Дом книги, г. Екатеринбург	тел. +7 (343) 289-40-45
Дом книги, г. Нижний Новгород	тел. +7 (831) 246-22-92
Приморский торговый Дом книги	тел. +7 (423) 263-10-54

➢ **маркетплейсы OZON, Wildberries, Яндекс-Маркет, Myshop и др.**

Михаил Шардин

Excel, Python и API:

Автоматизация данных и управление офисом, домом, финансами…

+ виртуальный диск с с готовыми скриптами и примерами

Группа подготовки издания:

Зав. редакцией компьютерной литературы: *Е.В. Финков*

Редакторы: *О.С. Петрунич, Н.В. Жерлов*

Корректор: *А.В. Громова*

Изображение на обложке использовано с ресурсов freepik.com, vecteezy.com

Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме без письменного разрешения владельцев авторских прав.

Издательство не несет ответственности за возможный ущерб, причиненный в ходе использования материалов данной книги, а также за доступность материалов, ссылки на которые вы можете найти в этой книге. На момент подготовки книги к изданию все ссылки на Интернет-ресурсы были действующими. Информация, содержащаяся в данной книге, получена из источников, рассматриваемых издательством как надежные. Тем не менее, имея в виду возможные человеческие или технические ошибки, издательство не может гарантировать абсолютную точность и полноту приводимых сведений и не несет ответственности за возможные ошибки, связанные с использованием книги.

12+

ООО "Издательство Наука и Техника"

ОГРН 1217800116247, ИНН 7811763020, КПП 781101001

192029, г. Санкт-Петербург, пр. Обуховской обороны, д. 107, лит. Б, пом. 1-Н

Подписано в печать 11.02.2026. Формат 70x100 1/16.

Бумага офсетная. Печать офсетная. Объем 27 п.л.

Тираж 2500. Заказ 16883.

Отпечатано с готового оригинал-макета

ООО «Принт-М», 142300, М.О., г.Чехов, ул. Полиграфистов, д.1



Михаил Шардин

кандидат технических наук, IT-разработчик и частный инвестор с опытом работы на финансовых рынках почти 20 лет.

Спикер многих конференций, автор более 300 публикаций с 2003 года (Хабр, Т—Ж, Смартлаб и другие ресурсы) о современных IT-инструментах и технологиях автоматизации, помогающих экономить время и бороться с цифровой неэффективностью и рутиной.

Книга объясняет, как совмещать вместе код, таблицы и нейросети так, чтобы они усиливали работу друг друга: генерировали сотни документов за пару минут, создавали автоматические отчёты, делали Вам умный дом без облака, проводили анализ личных расходов, мониторили цены на инвестиции и многое другое.

На конкретных примерах будет показано, какие именно повседневные задачи (например, создание однотипных отчётов, ручной поиск данных, бесконечные напоминания и т.п.) имеет смысл автоматизировать и как это делать с помощью уже знакомых инструментов: **Excel**'я или **Google Таблиц**, при этом дополняя их простыми скриптами на **Python** и небольшой работой с **API**.

Главной особенностью книги является её прикладной характер, поэтому все примеры, которые приведены в книге, взяты из реальной жизни – это офисная работа, домашние задачи, сложные семейные процессы, а также хобби и финансы. В книге нет обобщённых задач – они все конкретные, их можно взять из книги и адаптировать для себя.

издательство НАУКА и ТЕХНИКА Санкт-Петербург





Наши книги — ваши инвестиции в будущее

ISBN 978-5-6049658-5-6

9 785604 965856

nit.com.ru
nitbooks
nitbooks

Для заказа книг:
(812) 412-70-26
nitmail@nit.com.ru

НиТ издательство