Excel с Python и R

раскройте потенциал расширенной обработки и визуализации данных

СТИВЕН САНДЕРСОН
ДЭВИД КУН

Extending Excel with Python and R

Unlock the potential of analytics languages for advanced data manipulation and visualization

Steven Sanderson

David Kun

‹packt›

Excel с Python и R

раскройте потенциал расширенной обработки
и визуализации данных

Стивен Сандерсон

Дэвид Кун

SPRINT book 2025

Стивен Сандерсон, Дэвид Кун

Excel с Python и R: раскройте потенциал расширенной обработки и визуализации данных

Перевел с английского С. Черников

Научный редактор Денис Квист

ББК 32.973.23-018.2
УДК 004.67

Сандерсон Стивен, Кун Дэвид

С 18 Excel с Python и R: раскройте потенциал расширенной обработки и визуализации данных. — Астана: «Спринт Бук», 2025. — 320 с.: ил.

ISBN 978-601-08-4839-9

Анализ и визуализация данных имеют большое значение, именно они позволяют принимать обоснованные решения. Но в Excel достаточно много ограничений, которые превращают вашу жизнь в ад. «Excel с Python и R» меняет правила. Стивен Сандерсон — автор пакетов healthyverse для R, Дэвид Кун — соучредитель Functional Analytics, компании, создавшей ownR (платформа для разработки решений на R, Python и других языках обработки данных).

Интеграция Python и R с Excel изменит ваш подход к анализу данных с использованием электронных таблиц. Вы сможете автоматизировать задачи статистического анализа и создавать мощные визуализации, научитесь выполнять разведочный анализ данных и анализ временных рядов и даже интегрировать различные API для максимальной эффективности. И новички, и эксперты найдут в этой книге все необходимое, чтобы раскрыть весь потенциал Excel и поднять навыки анализа данных на новый уровень.

К концу книги вы освоите приемы импортирования данных из Excel, манипулирования ими в R или Python, сможете решать задачи анализа данных в выбранном вами фреймворке и возвращать результаты обратно в Excel.

16+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

.

ISBN 978-601-08-4839-9

Изготовлено в России. Изготовитель: ТОО «Спринт Бук». Место нахождения и фактический адрес: 010000, Казахстан, город Астана, район Алматы, Проспект Ракымжан Кошкарбаев, дом 10/1, н. п. 18.

Дата изготовления: 03.2025. Наименование: книжная продукция. Срок годности: не ограничен.

Подписано в печать 30.01.25. Формат 70×100/16. Бумага офсетная. Усл. п. л. 25,800. Тираж 700. Заказ 0000.

Краткое содержание

Оглавление

Часть III
Разведочный, статистический анализ данных и анализ временных рядов

Часть IV
Вызов кода R и Python из Excel

Часть V
Анализ и визуализация данных в Excel с помощью R и Python на конкретном примере

Моей любимой супруге. Спасибо за то, что каждый вечер укладывала детей спать, чтобы я мог работать над книгой. Я бы не смог ее написать, не имея этого свободного времени. Моим детям: папа вас любит. Моей маме. Спасибо за то, что считала хорошей любую мою идею.

Стивен Сандерсон

Моей семье, которая освещала каждый шаг этого путешествия своей любовью и неизменной поддержкой.

Дэвид Кун

Об авторах

Стивен Сандерсон, MPH, занимается приложениями в отделе расчетов с пациентами в Медицинском центре Университета Стони-Брук. Получил степень бакалавра экономики и MPH (Master of Public Health) в Университете Стони-Брук. Он работает в сфере здравоохранения уже почти 20 лет. Разработал и поддерживает набор пакетов healthyverse, написанных на языке R. Интересуется всем, что связано с социальной экономикой и экономикой труда, а недавно вернулся к игре на гитаре, надеясь на то, что его дети последуют его примеру и у них появится совместное хобби.

Я хочу поблагодарить издательство Packt за предоставленную возможность и моего соавтора Дэвида. Кроме того, хочу выразить благодарность своей семье, так как написание книги заняло довольно много времени.

Дэвид Кун — математик и актуарий (специалист по страховой математике, матстатистике и теории вероятностей), работает на стыке количественных методов анализа данных и информационно-коммуникационных технологий. Соучредитель и директор компании Functional Analytics, а также создатель платформы ownR Infinity. Использует ownR в повседневной работе для анализа данных. Проекты Дэвида связаны с анализом временных рядов для прогнозирования спроса, использованием компьютерного зрения для автоматизации проектирования и визуализацией данных.

Я сердечно благодарю свою спутницу Аню и своих детей за поддержку, которую они оказывали мне на протяжении всего процесса написания моей первой книги.

О научных редакторах

Хесус Мартин де ла Сьерра Сильва — R-разработчик с инженерным образованием, участвовавший в крупных ИТ-проектах, занимающихся преобразованием необработанных данных в обоснованные решения, которые можно применять на практике. Он помогает принимать решения на основе статистического подхода, исследуя взаимосвязи и закономерности в данных и создавая информативные визуализации. Кроме того, Хесус специализируется на создании удобных для пользователей приложений, которые легко интегрируются с такими сложными вычислительными методами, как интеллектуальный анализ процессов, прогнозирование, сетевой анализ и предсказательные модели машинного обучения. На протяжении многих лет Хесус призывает коллег перейти от использования электронных таблиц к применению мощных возможностей языка R для анализа данных.

Дэвид Наполи имеет степень магистра в области аэрокосмической техники и является докторантом в области исследований медицинских услуг и биостатистики. Обладает 25-летним опытом работы с данными и количественной разработки, в том числе следующими навыками: создание и обслуживание репозиториев данных, управление ими; применение методологий расчета поправки на риск; определение РПНУ (резерва произошедших, но незаявленных убытков); внедрение инноваций в аналитические платформы; расширенная визуализация данных; статистическое моделирование и оценка; анализ выживаемости. Дэвид применял свои глубокие знания в ходе работы на различных должностях, в том числе будучи директором по стратегической аналитике и заслуженным преподавателем по визуализации и расширенной аналитике данных.

Мехмет Синан Ийсой — биостатистик с многолетним опытом работы в сфере медицинской статистики. Имеет степени бакалавра и магистра в области математики и статистики. За свою карьеру успел поработать в качестве математика, программиста, системного администратора и специалиста по биостатистике. Неизменно демонстрирует приверженность к работе с данными и программированию. С энтузиазмом использует широкий спектр технологий, преимущественно состоящих из компонентов с открытым исходным кодом. Обладает глубокими знаниями и обширным опытом, которые на протяжении своей карьеры применял на разных

должностях в различных учреждениях. Увлеченный статистикой, Синан и сегодня продолжает вносить свой вклад в научную литературу и развитие общественных знаний, работая в Университете им. Неджметтина Эрбакана.

Шейн Алекс Хосе имеет степень магистра в области статистики. Его страсть к программированию возникла благодаря преподаванию Python, однако, по его словам, он гораздо увереннее чувствует себя при работе с R. Удивительно, но Шейн обожает отлаживать код и считает это занятием буквально катарсическим! В настоящее время работает программистом-аналитиком в команде аналитиков данных EvoEnergy, где совершенствует модели, а также создает и тестирует R-пакеты и дашборды, используемые внутри компании. Имеет опыт работы в различных отраслях и восхищается разнообразием данных в них, а также тем влиянием, которое эти данные способны оказывать (особенно при неправильном управлении). Шейн стремится стать хорошим специалистом в нескольких областях, чтобы способствовать выявлению изъянов и разрабатывать уникальные решения, позволяющие их устранить.

Предисловие

Добро пожаловать! В этой книге мы опишем способы, позволяющие дополнить функционал программы Excel мощными возможностями Python и R, а также предоставим подробное руководство о том, как с помощью этих языков работать с данными, анализировать и визуализировать их, а также выполнять многие другие действия.

Присоединяйтесь к нашему исследованию области пересечения Excel, R и Python, чтобы научиться успешно ориентироваться в мире больших данных.

Кому адресована эта книга

Книга рассчитана на пользователей R и/или Python среднего или более высокого уровня подготовки, имеющих некоторый опыт анализа данных, а также знакомых с основами работы в Excel.

Структура издания

Глава 1 «Чтение данных из электронных таблиц Excel» посвящена импорту данных из Excel в среду R/Python. Вы импортируете свой первый лист Excel в R, разберетесь в тонкостях работы с файлами Excel, а затем выполните импорт данных из Excel в Python.

В главе 2 «Запись данных в электронные таблицы Excel» речь пойдет о важности эффективного донесения до пользователей Excel результатов анализа данных, проведенного с помощью R/Python. Вы узнаете о том, как создавать таблицы Excel из среды R/Python и экспортировать в них результаты анализа.

В главе 3 «Запуск кода VBA из среды R и Python» рассказывается о том, как после записи результатов в итоговый лист Excel добавить макросы и функции VBA, чтобы расширить возможности конечных пользователей результатов анализа.

В главе 4 «Дальнейшая автоматизация: планирование задач и электронной рассылки» рассказывается о таком R-пакете, как RDCOMClient, который работает

с программами Outlook и Blastula и может помочь автоматизировать процесс анализа в R и отправить отчеты по электронной почте. В Python для той же цели используется пакет `smtplib`.

В главе 5 «Форматирование листа Excel» мы поговорим о том, как с помощью пакетов создавать листы и таблицы Excel, содержащие отформатированные данные, а также составлять эстетичные отчеты.

В главе 6 «Вставка графиков ggplot2/matplotlib» мы обсудим способы создания графиков с помощью библиотек `ggplot2` и `matplotlib`. Вдобавок мы рассмотрим существующие в `ggplot2` темы оформления, которые можно использовать для создания красивых графиков в R/Python и их вставки в Excel.

В главе 7 «Сводные таблицы» вы узнаете о том, как, используя R и Python, создавать сводные таблицы в Excel и управлять ими непосредственно из среды R/Python, обеспечивая при этом беспрепятственное взаимодействие с Excel.

В главе 8 «Разведочный анализ данных с помощью R и Python» рассказывается о том, как извлечь данные из файла Excel и выполнить их *разведочный анализ* (Exploratory Data Analysis, EDA), используя такие инструменты, как R-пакет `{skimr}` и пакеты Python `pandas` и `ppscore`.

В главе 9 «Статистический анализ: линейная и логистическая регрессия» вы научитесь выполнять простой статистический анализ данных Excel с помощью линейной и логистической регрессии в среде R и Python.

В главе 10 «Анализ временны́х рядов: статистика, графики и прогнозирование» описываются способы выполнения простого анализа временны́х рядов с помощью R-пакета `forecast`, а также пакета `kats` и нейросети с *долгой краткосрочной памятью* (long short-term memory (LSTM)) в Python.

Глава 11 «Локальный вызов кода R/Python из Excel напрямую или через API» посвящена способам локального вызова кода R и Python из Excel. В ней также рассматриваются инструменты с открытым исходным кодом для вызова локально установленной среды R/Python из Excel с помощью нейросети BERT и пакета `xlwings`, а также решений с открытым исходным кодом и коммерческих API-решений.

В главе 12 «Анализ и визуализация данных в Excel с помощью R и Python на конкретном примере» описан пример выполнения визуализации данных и машинного обучения в Excel с помощью R или Python.

Как извлечь максимальную пользу из книги

Чтобы лучше усвоить материал этой книги, необходимо владеть языком R или Python (или обоими) хотя бы на среднем уровне, а также обладать навыками обработки данных и их анализа с помощью таких библиотек, как `pandas`, `NumPy` и `tidyverse`. Вдобавок мы исходим из того, что читатели имеют такие основные

навыки работы с Excel, как навигация по электронным таблицам и выполнение простых операций с данными. Кроме того, хорошо иметь базовое понимание статистических концепций и методов визуализации данных — это будет полезно при изучении представленных в книге примеров и выполнении упражнений.

Программное/аппаратное обеспечение, рассматриваемое в книге	Требования к операционной системе
R	ОС Windows (для изучения материала, связанного с VBA), macOS или Linux (для изучения материала, не связанного с VBA)
Python 3.11	—
Excel (в том числе VBA)	—

В каждой главе вы найдете инструкции по установке соответствующих пакетов и инструментов.

Если вы используете электронную версию этой книги, то советуем вам набирать код самостоятельно или использовать код из репозитория GitHub (ссылку на него вы найдете в следующем разделе). Это поможет вам избежать возможных ошибок, связанных с копированием и вставкой кода.

ДИСКЛЕЙМЕР

Авторы признают: чтобы улучшить стиль изложения материала книги и сделать работу с ней более приятной, они использовали передовые системы искусственного интеллекта, такие как ChatGPT. Важно отметить, что сам материал был подготовлен авторами и отредактирован командой специалистов издательства.

Загрузка файлов с примерами кода

Код из примеров, приведенных в этой книге, опубликован на GitHub по адресу https://github.com/PacktPublishing/Extending-Excel-with-Python-and-R.

Условные обозначения

В книге используются следующие условные обозначения.

Фрагменты кода в тексте, названия таблиц базы данных, файлов, а также расширения файлов, пути к ним и данные, введенные пользователем, выделяются моноширинным шрифтом. Например: «Пакет `styledtables` можно установить только с GitHub с помощью пакета `devtools`».

Блок кода оформляется следующим образом:

```
install.packages("devtools")
# Установите версию разработки с GitHub
devtools::install_github(
'R-package/styledTables',
build_vignettes = TRUE
)
```

Любой ввод или вывод командной строки оформляется следующим образом:

```
python -m pip install pywin32==306
```

Курсивом выделяются новые термины, важные понятия.

Шрифтом без засечек выделяются URL, названия папок и каталогов, а также слова, которые вы видите на экране, в частности названия элементов меню или диалоговых окон, упомянутые в тексте. Например: «Прежде чем запускать этот код, вы можете проверить наличие макроса в файле `iris_data.xlsm`, выбрав пункт меню Разработчик ▸ Макрос (или Visual Basic)».

СОВЕТЫ И ПРИМЕЧАНИЯ

Оформляются так.

От издательства

Мы выражаем огромную благодарность компании Orion soft за помощь в работе над русскоязычным изданием книги и их вклад в повышение качества переводной литературы.

Ваши замечания, предложения, вопросы отправляйте по адресу comp@sprintbook.kz (издательство «SprintBook», компьютерная редакция).

Мы будем рады узнать ваше мнение!

О научном редакторе русскоязычного издания

Денис Квист — ведущий инженер-эксперт по виртуализации в компании Orion soft с более чем 24-летним опытом работы в ИТ. Его профессиональный путь начался с должности системного администратора и программиста в Национальном архиве Республики Карелия, где он стоял у истоков цифровизации архивной отрасли.

За свою карьеру Денис занимал ведущие позиции в различных организациях, включая гостиницу ParkInn, Республиканский перинатальный центр, Республиканский медицинский информационно-аналитический центр. Более двухсот студентов под его руководством прошли обучение по основам Linux-систем на платформе GeekBrains. Сегодня он применяет знания как в личных проектах (обслуживает и развивает три FM-радиостанции), так и в профессиональной среде: активно участвует в разработке и поддержке системы виртуализации zVirt.

Часть I

Основы: чтение и запись файлов Excel на R и Python

Эта часть книги является вводной и посвящена основам работы с файлами Excel в среде R и Python. В ней рассматриваются такие важные операции, как чтение и запись таблиц Excel с помощью популярных библиотек R и Python. Кроме того, здесь описаны способы, с помощью которых вы сможете автоматизировать решение задач и совершенствовать рабочие процессы в Excel, используя такие инструменты, как `RDCOMClient`, `blastula`, `schedule` и `smtplib` (позволяющие осуществлять запланированный запуск сценариев и отправку электронных писем), а также `readxl`, `openxlsx`, `xlsx`, `pandas` и `openpyxl`. Вдобавок мы поговорим о выполнении кода VBA.

1 Чтение данных из электронных таблиц Excel

В огромном мире анализа данных Excel — ваш верный помощник, упрощающий процесс вычислений, организации и представления информации. Благодаря интуитивно понятному интерфейсу и широкому распространению Excel получил статус основного инструмента в мире бизнеса. Однако объем данных и их сложность непрерывно растут, из-за чего возможности Excel становятся все более ограниченными. Решить эту проблему призвано сближение миров Excel, R и Python. В книге вы увидите, насколько эффективна синергия этих языков программирования и Excel, расширите возможности этой программы и научитесь легко решать задачи, связанные с данными. Мы подробно обсудим способы интеграции Excel с R и Python, которые позволяют раскрыть потенциал этой программы, извлекать ценные сведения, автоматизировать процессы и использовать все возможности анализа данных.

Пакет Microsoft Excel появился на рынке в 1985 году и до сих пор остается популярным программным обеспечением (ПО) для работы с электронными таблицами. Изначально приложение было известно под названием MultiPlan. Microsoft Excel и базы данных в целом имеют некоторые общие черты в плане организации и управления данными, хотя и служат разным целям. Excel — программа для работы с электронными таблицами, которая позволяет пользователям хранить данные в табличном формате и выполнять с ними различные действия. Таблица состоит из строк и столбцов, а каждая ее ячейка может содержать текст, числа или формулы. Аналогичным образом база данных представляет собой структурированную коллекцию данных, хранящихся в таблицах, состоящих из строк и столбцов.

И Excel, и базы данных позволяют хранить и извлекать данные. В Excel можно вводить данные, выполнять вычисления, создавать диаграммы и графики. Точно так же базы данных позволяют хранить большие объемы структурированных данных и управлять ими, выполняя запросы, сортировку и фильтрацию. Кроме того, Excel и базы данных поддерживают концепцию отношений. В Excel можно связывать ячейки или диапазоны ячеек на разных листах, устанавливая связи между

данными. Базы данных используют отношения для связывания таблиц на основе общих полей, что позволяет извлекать связанные данные из нескольких таблиц.

Цель этой главы — познакомить вас со способами чтения файлов Excel в среде R и выполнения некоторых операций над ними. В частности, мы рассмотрим следующие темы:

- обработка данных Excel с помощью пакетов R;
- чтение файлов Excel в среду R;
- чтение нескольких листов Excel с помощью readxl и пользовательской функции;
- пакеты Python для работы с Excel;
- открытие листа Excel из среды Python и чтение данных;
- чтение нескольких листов с помощью Python (openpyxl и пользовательских функций).

Технические требования

Во время написания книги мы использовали следующее программное обеспечение:

- R 4.2.1;
- версия RStudio 2023.03.1+446 Cherry Blossom для Windows.

Изучить материал этой главы вы сможете, установив следующие пакеты:

- `readxl`;
- `openxlsx`;
- `xlsx`.

Для выполнения кода Python, приведенного в этой главе, мы будем использовать:

- Python 3.11;
- `pandas`;
- `openpyxl`;
- Excel-файл `iris.xlsx`, доступный в репозитории GitHub для этой книги.

Описание процесса настройки среды Python выходит за рамки данной книги, но сделать это несложно. Необходимые пакеты можно установить, выполнив следующие команды:

```
python -m pip install pandas==2.0.1
python -m pip install openpyxl==3.1.2
```

Обратите внимание, что эти команды следует запускать из терминала, а не из сценария Python. Запустите их из папки, в которой находится файл `requirements.txt`, или укажите полный путь к нему.

В репозитории GitHub к этой книге также содержится файл `requirements.txt`, который вы можете использовать для установки всех зависимостей. Для этого выполните следующую команду:

```
python -m pip install -r requirements.txt
```

Она устанавливает все пакеты, которые будут использоваться в текущей главе, избавляя вас от необходимости устанавливать их по одному. Вдобавок она гарантирует, что дерево зависимостей будет полностью совпадать с тем, которое использовали мы, авторы данной книги.

В качестве альтернативы при использовании блокнотов Jupyter можно применить следующие команды:

```
%pip install pandas==2.0.1
%pip install openpyxl==3.1.2
```

Все примеры кода из книги представлены на GitHub по адресу https://github.com/PacktPublishing/Extending-Excel-with-Python-and-R. Примеры из каждой главы содержатся в соответствующей папке: примеры из текущей главы можно найти в папке Chapter1.

ПРИМЕЧАНИЕ

Технические требования к Python перечислены в файле requirements.txt, доступном в репозитории GitHub по адресу https://github.com/PacktPublishing/Extending-Excel-with-Python-and-R/blob/main/requirements.txt. Если вы установите эти зависимости, то вам будет проще писать код и легче читать книгу. Обязательно установите их все, прежде чем приступать к выполнению упражнений.

Обработка данных Excel с помощью пакетов R

На ресурсах CRAN и GitHub доступно несколько пакетов, позволяющих читать файлы Excel и выполнять с ними различные действия. В этом разделе мы рассмотрим пакеты `readxl`, `openxlsx` и `xlsx`. В них есть функции для чтения файлов Excel:

- `readxl::read_excel()`;
- `openxlsx::read.xlsx()`;
- `xlsx::read.xlsx()`.

Каждая функция предусматривает свой набор параметров и соглашений, которые необходимо соблюдать. Пакет `readxl` — часть коллекции пакетов `tidyverse`, поэтому он придерживается ее соглашений и после чтения файла возвращает объект `tibble` (тиббл) — современную версию `data.frame` языка R, которая представляет собой своего рода электронную таблицу в среде R. Данный объект служит строительным блоком при выполнении большинства видов анализа. Что касается пакетов `openxlsx` и `xlsx`, то они оба возвращают базовый объект R `data.frame`, причем `xlsx` может возвращать еще и объект `list`. Возможно, вам интересно, как это связано с обработкой реального файла Excel. Начнем вот с чего: чтобы получить возможность работать с данными в среде R, их необходимо сначала считать. В указанных выше пакетах есть различные функции для работы с файлами Excel, а также для чтения данных такими способами, которые позволяют проводить их дальнейший анализ. Имейте в виду, что пакет `xlsx` требует установки Java.

Обсудив пакеты R для работы с Excel, поговорим об эффективном способе считывания файлов Excel в среду R, который предоставляет еще больше возможностей для анализа данных и работы с ними.

Чтение файлов Excel в среду R

В этом разделе мы будем считывать данные из Excel с помощью различных библиотек R. Это необходимо сделать до выполнения каких-либо действий с данными, содержащимися в листах файлов Excel, или до их анализа.

Как уже было сказано в разделе «Технические требования» выше, для считывания данных в среду R мы будем использовать пакеты `readxl`, `openxlsx` и `xlsx`.

Установка и загрузка библиотек

Мы будем использовать библиотеки `openxlsx`, `xlsx`, `readxl` и `readxlsb`. Если вы еще не установили и не загрузили их, выполните следующий блок кода:

```
pkgs <- c("openxlsx", "xlsx", "readxl")
install.packages(pkgs, dependencies = TRUE)
lapply(pkgs, library, character.only = TRUE)
```

Функция `lapply()` в языке R — это универсальный инструмент для применения функции к каждому элементу списка, вектора данных или датафрейма (`DataFrame`). Она принимает два аргумента: `x` и `FUN`, где `x` — это список, а `FUN` — функция, которая применяется к объекту `x`.

Теперь, когда библиотеки установлены, можем приступить к работе. Для этого мы считаем электронную таблицу, созданную на основе встроенного в R набора

данных `Iris`. Мы прочитаем файл с помощью трех разных библиотек, а затем создадим пользовательскую функцию для работы с библиотекой `readxl`, которая будет считывать все листы файла Excel. Мы назовем эту функцию `read_excel_sheets()`.

Первой библиотекой, которой мы воспользуемся для открытия файла Excel, будет `openxlsx`. Чтобы прочитать нужный файл Excel, вы можете запустить код из папки `chapter1` репозитория GitHub под названием `ch1_create_iris_dataset.R`. Процесс считывания файла в среду R показан на рис. 1.1.

```
> # Use openxlsx
> openxlsx::read.xlsx(f_path) |> head(5)
  sepal_length sepal_width petal_length petal_width species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
> openxlsx::read.xlsx(f_path, sheet = "iris") |> head(5)
  sepal_length sepal_width petal_length petal_width species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
```

Рис. 1.1. Использование пакета openxlsx для чтения файла Excel

Обратите внимание на переменную `f_path`. Она содержит путь к месту сохранения набора данных Iris в виде файла Excel, например `C:/User/UserName/Documents/iris_data.xlsx`.

В этом примере предполагается, что для создания файла Excel вы использовали файл `ch1_create_iris_datase.R`. В действительности вы можете прочитать любой нужный вам файл Excel.

Теперь мы выполним ту же операцию, но уже с помощью библиотеки `xlsx`. На рис. 1.2 показан тот же метод чтения, что и в случае с пакетом `openxlsx`.

Наконец, воспользуемся библиотекой `readxl`, которая является частью коллекции пакетов `tidyverse` (рис. 1.3).

```
> # Use xlsx
> xlsx::read.xlsx(file = f_path, sheetIndex = 1) |> head(5)
  sepal_length sepal_width petal_length petal_width species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
> xlsx::read.xlsx(file = f_path, sheetName = "iris") |> head(5)
  sepal_length sepal_width petal_length petal_width species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
```

Рис. 1.2. Использование библиотеки xlsx и функции read.xlsx() для открытия созданного нами файла Excel

```
> # Use readxl
> readxl::read_excel(f_path) |> head(5)
# A tibble: 5 × 5
  sepal_length sepal_width petal_length petal_width species
         <dbl>       <dbl>        <dbl>       <dbl> <chr>
1          5.1         3.5          1.4         0.2 setosa
2          4.9         3            1.4         0.2 setosa
3          4.7         3.2          1.3         0.2 setosa
4          4.6         3.1          1.5         0.2 setosa
5          5           3.6          1.4         0.2 setosa
> readxl::read_excel(f_path, "iris") |> head(5)
# A tibble: 5 × 5
  sepal_length sepal_width petal_length petal_width species
         <dbl>       <dbl>        <dbl>       <dbl> <chr>
1          5.1         3.5          1.4         0.2 setosa
2          4.9         3            1.4         0.2 setosa
3          4.7         3.2          1.3         0.2 setosa
4          4.6         3.1          1.5         0.2 setosa
5          5           3.6          1.4         0.2 setosa
```

Рис. 1.3. Использование библиотеки readxl и функции read_excel() для считывания файла Excel в память

В этом разделе вы научились считывать файлы Excel в среду R с помощью нескольких различных пакетов. Эти пакеты позволяют выполнять множество других операций, однако в данном случае нас интересовало именно чтение файлов. Итак, вы знаете, как использовать функции `readxl::read_excel()`, `xlsx::read.xlsx()` и `openxlsx::read.xlsx()`.

Теперь мы можем двигаться дальше и поговорить об эффективном извлечении данных из нескольких листов файла Excel.

Чтение нескольких листов с помощью readxl и пользовательской функции

При работе в Excel мы часто имеем дело с рабочими книгами, включающими несколько листов. Они могут содержать представленные в определенном формате статистические данные за разные месяцы или какие-то другие периоды. По тем или иным причинам мы можем захотеть прочитать все листы, содержащиеся в файле, не вызывая при этом функцию чтения для каждого отдельного листа. Можно использовать R, чтобы выполнить эту операцию в цикле с помощью пакета `purrr`.

Создадим пользовательскую функцию. Для этого загрузим функцию `readxl`, если она еще не загружена. Однако если библиотека установлена, но вы не хотите загружать ее в память, можете вызвать функцию `excel_sheets()` с помощью `readxl::excel_sheets()` (рис. 1.4).

Этот новый код можно разделить так:

```
read_excel_sheets <- function(filename, single_tbl) {
```

Эта строка определяет функцию `read_excel_sheets`, принимающую два аргумента: `filename` (имя файла Excel, который нужно прочитать) и `single_tbl` (логическое значение, указывающее на то, что именно должна возвращать функция: одну таблицу или список таблиц).

Далее следует такая строка:

```
sheets <- readxl::excel_sheets(filename)
```

Здесь пакет `readxl` используется для извлечения имен всех листов из файла Excel с именем, определяемым параметром `filename`. Имена листов сохраняются в переменной `sheets`.

```
read_excel_sheets <- function(filename, single_tbl = FALSE) {
  sheets <- readxl::excel_sheets(filename)

  if (single_tbl){
    x <- purrr::map_df(sheets, readxl::read_excel, path = filename)
  } else {
    x <- purrr::map(sheets, ~ readxl::read_excel(filename, sheet = .x))
    purrr::set_names(x, sheets)
  }

  x
}
```

Рис. 1.4. Создание функции read_excel_sheets() для одновременного считывания всех листов из файла Excel

Следующая строка выглядит так:

```
if (single_tbl) {
```

Она запускает инструкцию `if`, которая проверяет значение аргумента `single_tbl`.

Далее следует строка:

```
x <- purrr::map_df(sheets, read_excel, path = filename)
```

Если значением `single_tbl` является `TRUE`, то вызывается функция `map_df` пакета `purrr` для перебора имен листов, хранящихся в переменной `sheets`, и чтения соответствующих листов с помощью функции `read_excel` из пакета `readxl`. Полученные объекты `DataFrame` объединяются в таблицу, которая присваивается переменной `x`.

Следующая строка выглядит так:

```
} else {
```

Это начало блока `else` инструкции `if`. Если значением `single_tbl` является `FALSE`, то выполняется код, содержащийся в этом блоке.

Затем идет такая строка:

```
x <- purrr::map(sheets, ~ readxl::read_excel(filename, sheet = .x))
```

Здесь функция `map` пакета `purrr` используется для перебора имен листов, хранящихся в переменной `sheets`. Для чтения каждого листа файла Excel, имя которого определяется параметром `filename`, вызывается функция `read_excel` из пакета `readxl`. Полученные объекты `DataFrame` сохраняются в списке, который присваивается переменной `x`.

Далее следует строка:

```
purrr::set_names(x, sheets)
```

В ней используется функция `set_names` из пакета `purrr` для присвоения элементам списка `x` имен листов, хранящихся в переменной `sheets`.

Предпоследняя строка выглядит так:

```
x
```

Она возвращает из функции значение `x`, которое будет представлять собой либо одну таблицу (`data.frame`), если значением `single_tbl` является `TRUE`, либо список таблиц (`data.frame`), если значением `single_tbl` является `FALSE`.

Итак, функция `read_excel_sheets` принимает имя файла Excel и логическое значение, указывающее на то, что именно необходимо возвратить: одну таблицу или их список. Функция использует пакет `readxl` для извлечения имен листов из файла Excel, а затем считывает соответствующие листы либо в одну таблицу (если значением `single_tbl` является `TRUE`), либо в их список (если значением `single_tbl` является `FALSE`). Полученные данные возвращаются в качестве результата работы функции. Чтобы увидеть, как работает эта схема, рассмотрим следующий пример.

У нас есть электронная таблица с четырьмя вкладками — по одной для каждого вида ирисов из набора данных Iris, и еще один лист под названием `iris`, который содержит весь набор данных.

Как показано на рис. 1.5, функция `read_excel_sheets()` прочитала все четыре листа файла Excel. Кроме того, мы видим, что эта функция импортировала листы в виде объекта-списка и присвоила каждому его элементу имя соответствующей вкладки в файле Excel. Важно отметить, что для корректной работы данной функции все листы должны иметь одинаковые имена столбцов и структуру.

В этом разделе мы показали процесс написания функции, которая может прочитать все листы в любом файле Excel и возвратить их в виде списка элементов, имена которых соответствуют названиям вкладок в исходном файле.

Теперь, когда вы научились считывать листы Excel в среду R, посмотрим, как делать то же самое с помощью языка Python.

```
> read_excel_sheets(f, T)
# A tibble: 300 × 5
   sepal_length sepal_width petal_length petal_width species
          <dbl>       <dbl>        <dbl>       <dbl> <chr>
 1          5.1         3.5          1.4         0.2 setosa
 2          4.9         3            1.4         0.2 setosa

> read_excel_sheets(f, F)
[[1]]
# A tibble: 50 × 5
   sepal_length sepal_width petal_length petal_width species
          <dbl>       <dbl>        <dbl>       <dbl> <chr>
 1          5.1         3.5          1.4         0.2 setosa
 2          4.9         3            1.4         0.2 setosa

[[2]]
# A tibble: 50 × 5
   sepal_length sepal_width petal_length petal_width species
          <dbl>       <dbl>        <dbl>       <dbl> <chr>
 1          7           3.2          4.7         1.4 versicolor
 2          6.4         3.2          4.5         1.5 versicolor

[[3]]
# A tibble: 50 × 5
   sepal_length sepal_width petal_length petal_width species
          <dbl>       <dbl>        <dbl>       <dbl> <chr>
 1          6.3         3.3          6           2.5 virginica
 2          5.8         2.7          5.1         1.9 virginica

[[4]]
# A tibble: 150 × 5
   sepal_length sepal_width petal_length petal_width species
          <dbl>       <dbl>        <dbl>       <dbl> <chr>
 1          5.1         3.5          1.4         0.2 setosa
 2          4.9         3            1.4         0.2 setosa
```

Рис. 1.5. Файл Excel, прочитанный с помощью функции read_excel_sheets()

Пакеты Python для работы с Excel

Один из ключевых аспектов работы с файлами Excel в Python — наличие подходящего набора пакетов, обеспечивающих необходимую функциональность. В этом разделе мы обсудим пакеты Python, наиболее часто используемые при работе с Excel, и расскажем об их преимуществах и особенностях.

Пакеты Python для работы с данными Excel

Для работы с файлами Excel в Python существует несколько пакетов, позволяющих извлекать данные из файлов Excel, выполнять с ними различные действия и записывать их обратно в файлы Excel. Рассмотрим несколько популярных пакетов Python.

- `pandas` — мощная библиотека для работы с данными, которая может читать файлы Excel с помощью функции `read_excel`. Используя эту библиотеку, вы с помощью объекта `DataFrame` можете обрабатывать данные, представленные в табличной форме, а это упрощает их анализ. Пакет `pandas` эффективно обрабатывает большие наборы данных и предоставляет различные способы их фильтрации, преобразования и агрегирования.
- `openpyxl` — широко распространенная библиотека, специально предназначенная для работы с файлами Excel. Она предоставляет полный набор функций для чтения и записи таблиц Excel, содержит поддержку различных форматов файлов Excel и обеспечивает совместимость с различными версиями этого приложения. Кроме того, `openpyxl` позволяет осуществлять точный контроль над структурой и содержимым файлов Excel и выполнять такие задачи, как обращение к отдельным ячейкам, создание новых рабочих листов и применение форматирования.
- `xlrd` и `xlwt` — старые библиотеки, которые до сих пор используются для чтения и записи файлов Excel, особенно в таких устаревших форматах, как `.xls`. Пакет `xlrd` позволяет считывать данные из файлов Excel, а `xlwt` — записывать их в файлы Excel. Эти библиотеки легковесны и просты в использовании, но им не хватает некоторых расширенных возможностей, предоставляемых пакетами `pandas` и `openpyxl`.

Факторы, которые следует учесть при выборе пакета

При выборе пакета Python для работы с файлами Excel необходимо учитывать специфические требования вашего проекта. Вот несколько факторов, которые следует иметь в виду.

- **Функциональность.** Оцените возможности пакета и убедитесь в том, что он отвечает вашим потребностям в плане чтения файлов Excel. Подумайте, нужны ли вам расширенные функции для работы с данными или будет достаточно относительно простого пакета.
- **Производительность.** Если вы работаете с большими наборами данных или нуждаетесь в их эффективной обработке, то благодаря использованию таких пакетов, как `pandas`, содержащих оптимизированные алгоритмы, вы можете получить значительные преимущества с точки зрения производительности.

- **Совместимость.** Проверьте совместимость пакета с различными форматами и версиями файлов Excel. Убедитесь, что он поддерживает конкретный формат, с которым вы работаете, — это позволит избежать проблем с совместимостью.
- **Сложность освоения.** Оцените сложность освоения каждого пакета. Некоторые пакеты, например `pandas`, предоставляют более широкий набор функций, но их изучение может потребовать дополнительного времени и усилий.

Каждый пакет Python предлагает уникальные возможности для чтения таблиц Excel и имеет свои сильные и слабые стороны. Например, если вам необходимо считывать большие объемы данных и выполнять с ними различные действия, то `pandas` может оказаться более предпочтительным выбором. Но если вы стремитесь к точному контролю над файлом Excel, то вам, скорее всего, больше подойдет `openpyxl`.

В следующих разделах мы подробно рассмотрим способы применения этих пакетов для чтения и извлечения данных из файлов Excel с помощью Python.

Открытие листа Excel из среды Python и чтение данных

При работе с файлами Excel в Python часто возникает необходимость открыть определенный лист и считать данные в среду Python, чтобы впоследствии выполнить их анализ. Это можно сделать с помощью таких популярных библиотек, как `pandas` и `openpyxl`, речь о которых шла в предыдущем разделе.

Использование пакета pandas

Как мы уже сказали, `pandas` — мощная библиотека, которая упрощает процесс работы со структурированными данными, в том числе с электронными таблицами Excel. Чтобы прочитать лист Excel с помощью `pandas`, можно задействовать функцию `read_excel`. Рассмотрим пример использования файла `iris_data.xlsx`, содержащего лист `setosa`:

```
import pandas as pd
# Чтение файла Excel
df = pd.read_excel('iris_data.xlsx', sheet_name='setosa')

# Отображение нескольких первых строк объекта DataFrame
print(df.head())
```

Вам нужно будет запустить этот код, указав в качестве рабочего каталога Python то место, где находится файл Excel, либо задать полный путь к файлу в команде `read_excel()` (рис. 1.6).

```
>>> import pandas as pd
>>> # Read the Excel file
>>>
>>> df = pd.read_excel('iris_data.xlsx', sheet_name='setosa')
>>> # Display the first few rows of the DataFrame
>>>
>>> print(df.head())
   sepal_length  sepal_width  petal_length  petal_width species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa
>>>
```

Рис. 1.6. Использование пакета pandas для чтения файла Excel

В предыдущем фрагменте кода мы импортировали библиотеку `pandas` и использовали функцию `read_excel` для чтения листа `setosa` из файла `iris_data.xlsx`. Полученные данные хранятся в объекте `pandas DataFrame`, который обеспечивает табличное представление данных. Вызвав функцию `head()` на объекте `DataFrame`, мы отобразили первые несколько строк таблицы, чтобы осуществить ее предварительный просмотр.

Использование пакета openpyxl

Напомним, что `openpyxl` — это мощная библиотека для работы с файлами Excel, позволяющая осуществлять более точный контроль над отдельными ячейками и листами. Чтобы открыть лист Excel и получить доступ к его данным с помощью `openpyxl`, можно использовать функцию `load_workbook`. Обратите внимание, что этот пакет не может работать с файлами `.xls`, он совместим только с более современными версиями `.xlsx` и `.xlsm`.

В качестве примера рассмотрим файл `iris_data.xlsx`, содержащий лист `versicolor`:

```
import openpyxl
import pandas as pd

# Загрузка рабочей книги
wb = openpyxl.load_workbook('iris_data.xlsx')

# Выбор листа
sheet = wb['versicolor']

# Извлечение значений (в том числе заголовка)
sheet_data_raw = sheet.values
```

```
# Выделение заголовков в переменную
header = next(sheet_data_raw)[0:]

# Создание объекта DataFrame на основе второй
# и последующих строк данных с использованием заголовков
# в качестве названий столбцов
sheet_data = pd.DataFrame(sheet_data_raw, columns=header)
print(sheet_data.head())
```

Результат выполнения приведенного выше кода выглядит так (рис. 1.7).

```
>>> import openpyxl
>>> import pandas as pd
>>> # Load the workbook
>>>
>>> wb = openpyxl.load_workbook('iris_data.xlsx')
>>> # Select the sheet
>>>
>>> sheet = wb['versicolor']
>>> # Extract the values (including header)
>>>
>>> sheet_data_raw = sheet.values
>>> # Separate the headers into a variable
>>>
>>> header = next(sheet_data_raw)[0:]
>>> # Create a DataFrame based on the second and subsequent lines of data with the header as column names
>>>
>>> sheet_data = pd.DataFrame(sheet_data_raw, columns=header)
>>> print(sheet_data.head())
   sepal_length  sepal_width  petal_length  petal_width     species
0           7.0          3.2           4.7          1.4  versicolor
1           6.4          3.2           4.5          1.5  versicolor
2           6.9          3.1           4.9          1.5  versicolor
3           5.5          2.3           4.0          1.3  versicolor
4           6.5          2.8           4.6          1.5  versicolor
>>>
```

Рис. 1.7. Использование пакета openpyxl для чтения файла Excel

В этом фрагменте кода мы импортируем функцию `load_workbook` из библиотеки `openpyxl`. Затем загружаем рабочую книгу, указав имя файла `iris_data.xlsx`. Далее мы выбираем нужный лист, обращаясь к нему по имени, — в нашем случае это `versicolor`. После этого считываем необработанные данные, используя свойство `values` загруженного объекта-листа. Это генератор, доступ к которому можно получить, используя цикл `for` либо путем преобразования объекта-листа в список или объект `DataFrame`. В данном примере мы преобразовали его в `pandas DataFrame`, поскольку именно с этим форматом нам будет удобнее всего работать в дальнейшем.

Пакеты `pandas` и `openpyxl` предоставляют весьма ценные функции для работы с файлами Excel в среде Python. Библиотека `pandas` упрощает обработку данных благодаря своей структуре `DataFrame`, а `openpyxl` обеспечивает более точный контроль над отдельными ячейками и листами. Вы можете выбрать библиотеку, которая лучше всего отвечает вашим потребностям.

Освоив методы открытия листов Excel и считывания данных в среду Python, вы сможете извлекать ценные сведения из своих файлов Excel, выполнять различные преобразования данных и подготавливать их для дальнейшего анализа или визуализации. Эти навыки необходимы всем, кто стремится использовать возможности Python и Excel в своих рабочих процессах.

Чтение нескольких листов с помощью Python (openpyxl и пользовательских функций)

Многие файлы Excel состоят из нескольких листов, содержащих различные наборы данных. Считывание данных из нескольких листов и объединение их в общую структуру данных может быть очень полезным умением при их анализе и обработке. В данном разделе мы обсудим способ решения этой задачи с помощью библиотеки `openpyxl` и пользовательской функции.

Важность чтения нескольких листов

При работе со сложными файлами Excel нередко возникают ситуации, когда связанные данные оказываются распределенными по разным листам. Например, один лист может содержать данные о продажах, второй — информацию о клиентах, а третий — сведения об остатках товаров. Если считать данные с нескольких листов и объединить их, то можно получить целостную картину и провести всесторонний анализ.

Рассмотрим основные этапы процесса чтения нескольких листов.

1. **Загрузка рабочей книги.** Перед получением доступа к листам нужно загрузить рабочую книгу Excel с помощью функции `load_workbook` из пакета `openpyxl`.
2. **Получение имен листов.** Мы можем получить имена всех листов рабочей книги, используя атрибут `sheetnames`. Это позволит нам определить листы, которые необходимо прочитать.
3. **Считывание данных с каждого листа.** Перебирая имена листов, мы можем получить доступ к каждому из них по отдельности и прочитать содержащиеся в них данные. Пакет `openpyxl` предоставляет такие методы, как `iter_rows` или `iter_cols`, которые позволяют выполнить обход ячеек каждого листа и получить нужные данные.
4. **Сохранение данных.** Чтобы объединить данные с нескольких листов, мы можем использовать подходящую структуру данных, например объект `pandas DataFrame` или список Python. В процессе чтения данных с каждого листа мы объединяем их в общую структуру данных, выполняя конкатенацию или слияние.

- Если данные на всех листах имеют одинаковый формат (как в примере этой главы), то мы можем просто объединить наборы данных, выполнив конкатенацию.
- Если же наборы данных имеют разную структуру, поскольку описывают разные показатели (например, как мы уже сказали, один лист содержит данные о товарах, второй — информацию о клиентах, а третий — сведения о продажах товаров клиентам), то мы можем выполнить слияние этих наборов на основе уникальных идентификаторов, чтобы получить полный набор.

Использование пакета openpyxl для получения доступа к листам

Напомним, что `openpyxl` — мощная библиотека, позволяющая взаимодействовать с файлами Excel с помощью Python. Она предоставляет широкий спектр функций, в том числе возможность получения доступа к нескольким листам сразу и выполнения с ними различных действий. Прежде чем погружаться в детали, разберемся, почему `openpyxl` часто выбирается для решения этой задачи.

Одно из главных преимуществ пакета `openpyxl` — его способность работать с файлами Excel различных форматов, таких как `.xlsx` и `.xlsm`. Благодаря этому вы можете работать с различными версиями файлов Excel, не испытывая проблем с совместимостью. Кроме того, `openpyxl` предоставляет простой и интуитивно понятный интерфейс, позволяющий получать доступ к содержащимся в листе данным, что упрощает нахождение нужной информации.

Чтение данных с каждого листа

Чтобы считать несколько листов, нужно загрузить рабочую книгу Excel с помощью функции `load_workbook` из библиотеки `openpyxl`. Эта функция принимает путь к файлу и возвращает объект рабочей книги, представляющий весь файл Excel.

Загрузив рабочую книгу, мы можем получить названия всех листов, присутствующих в файле Excel, с помощью атрибута `sheetnames`. Затем можем перебрать эти названия, чтобы считать данные с каждого листа по отдельности.

Получение данных из листа с помощью пакета openpyxl

Пакет `openpyxl` предоставляет различные методы для получения доступа к данным, содержащимся в листе.

Два наиболее часто используемых метода — `iter_rows` и `iter_cols`. Они позволяют перебирать строки или столбцы листа и извлекать значения, содержащиеся в ячейках.

Рассмотрим пример использования метода `iter_rows`:

```
# Предполагается, что мы работаем с первым листом
sheet = wb['versicolor']

# Перебор строк и вывод на экран необработанных значений
for row in sheet.iter_rows(min_row=1, max_row=5, values_only=True):
    print(row)
```

Метод `iter_cols` можно использовать следующим образом:

```
# Перебор столбцов и вывод на экран необработанных значений
for column in sheet.iter_cols(min_col=1, max_col=5, values_only=True):
    print(column)
```

При использовании методов `iter_rows` или `iter_cols` мы можем указать, хотим ли получить значения ячеек в необработанном или отформатированном виде. Необработанные значения предоставляют фактические данные, хранящиеся в ячейках, в то время как отформатированные значения содержат любое форматирование, примененное к ячейкам, например форматирование даты или числа.

Перебирая строки или столбцы листа, мы можем получить нужные данные и сохранить их в подходящей структуре данных. Один из популярных вариантов — использование объектов `pandas DataFrame`, которые обеспечивают табличное представление данных и предлагают удобные методы их обработки и анализа.

Альтернативное решение — использование атрибута `values` объекта-листа. Оно предоставляет генератор для всех значений в листе (подобно тому, как это делают `iter_rows` и `iter_cols` для строк и столбцов соответственно). Хотя генераторы нельзя использовать для получения непосредственного доступа к данным, их можно задействовать в циклах `for` для перебора значений. Кроме того, функция `DataFrame` библиотеки `pandas` позволяет напрямую преобразовать подходящий объект-генератор в `DataFrame`.

Объединение данных из нескольких листов

В процессе считывания данных с каждого из листов мы можем сохранить их в списке или словаре в зависимости от наших потребностей. Получив данные со всех листов, можем объединить их в общую структуру данных. Этот шаг очень важен, поскольку впоследствии их можно подвергать анализу и обработке.

Чтобы объединить данные, мы можем использовать объект `pandas DataFrame`. Создавая отдельные `DataFrame` для данных каждого листа, а затем объединяя в общий `DataFrame` путем конкатенации или слияния, мы можем получить полный набор данных, содержащий всю информацию из нескольких листов.

Пользовательская функция для чтения нескольких листов

Чтобы упростить процесс чтения нескольких листов и объединения данных, мы можем создать пользовательские функции, отвечающие нашим потребностям. Эти функции инкапсулируют необходимые действия и позволяют повторно использовать код.

В нашем примере мы определяем функцию `read_multiple_sheets`, которая принимает в качестве входных данных путь к файлу. Внутри нее мы загружаем рабочую книгу, используя `load_workbook`, и перебираем названия листов, полученные с помощью атрибута `sheets`.

Мы обращаемся к каждому листу через объект рабочей книги и получаем данные с помощью пользовательской функции `read_single_sheet`. Затем сохраняем полученные данные в списке. Наконец, объединяем данные со всех листов в объект `pandas DataFrame`, используя соответствующий метод конкатенации из библиотеки `pandas`.

С помощью этих пользовательских функций мы можем легко считать несколько листов из файла Excel и получить объединенный и готовый к анализу набор данных. Такая функция представляет собой многоразовое и эффективное решение, позволяющее экономить время и силы при работе со сложными файлами Excel.

Настройка кода

Приведенный пример — отправная точка, которую вы можете изменить в соответствии со своими потребностями. Ниже мы указали несколько нюансов, касающихся настройки кода.

- **Фильтрация столбцов.** Если вам нужны определенные столбцы из каждого листа, то вы можете изменить код, чтобы извлечь только нужные столбцы на этапе получения данных. Для этого можно применить метод `iter_cols` вместо атрибута `values` и использовать отфильтрованный список в цикле `for` или отфильтровать полученный (-е) объект (-ы) `pandas DataFrame`.
- **Обработка отсутствующих значений.** Если какие-то данные в листах отсутствуют, то вы можете использовать соответствующие методы обработки, например заполнить пропущенные значения или исключить неполные строки.
- **Применение преобразований.** В зависимости от характера данных вам может потребоваться применить преобразования к объединенному набору данных или произвести некоторые вычисления. Пользовательскую функцию можно расширить для учета подобных преобразований.

Помните: цель настройки кода — обеспечить его соответствие вашим потребностям в плане обработки данных.

Используя возможности `openpyxl` и создавая пользовательские функции, вы можете эффективно считывать сразу несколько листов из файлов Excel, объединять данные и подготавливать их к дальнейшему анализу. Эта возможность позволит вам извлекать ценные сведения из сложных файлов Excel и использовать весь потенциал ваших данных.

Теперь рассмотрим пример, демонстрирующий этот процесс:

```
from openpyxl import load_workbook
import pandas as pd

def read_single_sheet(workbook, sheet_name):
    # Загрузка листа из рабочей книги
    sheet = workbook[sheet_name]

    # Считывание необработанных данных, в том числе заголовков
    sheet_data_raw = sheet.values

    # Выделение заголовков в переменную
    columns = next(sheet_data_raw)[0:]

    # Создание объекта DataFrame на основе второй и последующих
    # строк данных с использованием заголовков в качестве
    # названий столбцов и его возврат
    return pd.DataFrame(sheet_data_raw, columns=columns)

def read_multiple_sheets(file_path):

    # Загрузка рабочей книги
    workbook = load_workbook(file_path)

    # Получение списка названий всех листов в рабочей книге
    sheet_names = workbook.sheetnames

    # Перебор названий листов, загрузка данных для каждого
    # из них и их конкатенация в один объект DataFrame
    return pd.concat([read_single_sheet(workbook=workbook, sheet_name=sheet_name)
      for sheet_name in sheet_names], ignore_index=True)

# Определение пути к файлу и названий листов
file_path = 'iris_data.xlsx' # adjust the path as needed

# Считывание данных с нескольких листов
consolidated_data = read_multiple_sheets(file_path)

# Вывод объединенных данных
print(consolidated_data.head())
```

Результаты выполнения этого кода показаны на рис. 1.8.

```
>>> import openpyxl
>>> import pandas as pd
>>> # Load the workbook
>>>
>>> wb = openpyxl.load_workbook('iris_data.xlsx')
>>> # Select the sheet
>>>
>>> sheet = wb['versicolor']
>>> # Extract the values (including header)
>>>
>>> sheet_data_raw = sheet.values
>>> # Separate the headers into a variable
>>>
>>> header = next(sheet_data_raw)[0:]
>>> # Create a DataFrame based on the second and subsequent lines of data with the header as column names
>>>
>>> sheet_data = pd.DataFrame(sheet_data_raw, columns=header)
>>> print(sheet_data.head())
   sepal_length  sepal_width  petal_length  petal_width     species
0           7.0          3.2           4.7          1.4  versicolor
1           6.4          3.2           4.5          1.5  versicolor
2           6.9          3.1           4.9          1.5  versicolor
3           5.5          2.3           4.0          1.3  versicolor
4           6.5          2.8           4.6          1.5  versicolor
>>> from openpyxl import load_workbook
>>> import pandas as pd
>>> def read_single_sheet(workbook, sheet_name):
...         # Load the sheet from the workbook
...         sheet = workbook[sheet_name]
...         # Read out the raaw data including headers
...         sheet_data_raw = sheet.values
...         # Separate the headers into a variable
...         columns = next(sheet_data_raw)[0:]
...         # Create a DataFrame based on the second and subsequent lines of data with the header as column names and return it
...         return pd.DataFrame(sheet_data_raw, columns=columns)
...
>>> def read_multiple_sheets(file_path):
...     # Load the workbook
...     workbook = load_workbook(file_path)
...     # Get a list of all sheet names in the workbook
...     sheet_names = workbook.sheetnames
...     # Cycle through the sheet names, load the data for each and concatenate them into a single DataFrame
...     return pd.concat([read_single_sheet(workbook=workbook, sheet_name=sheet_name) for sheet_name in sheet_names], ignore_index=True)
... # Define the file path and sheet names
...
>>> file_path = 'iris_data.xlsx'
>>> # Read the data from multiple sheets
>>>
>>> consolidated_data = read_multiple_sheets(file_path)
>>> # Display the consolidated data
>>>
>>> print(consolidated_data.head())
   sepal_length  sepal_width  petal_length  petal_width species   1   2   3
0           5.1          3.5           1.4          0.2  setosa NaN NaN NaN
1           4.9          3.0           1.4          0.2  setosa NaN NaN NaN
2           4.7          3.2           1.3          0.2  setosa NaN NaN NaN
3           4.6          3.1           1.5          0.2  setosa NaN NaN NaN
4           5.0          3.6           1.4          0.2  setosa NaN NaN NaN
>>>
```

Рис. 1.8. Использование пакета openxlsx для чтения файла Excel

В приведенном выше коде мы определили две функции:

- `read_single_sheet` считывает данные с одного листа в объект `pandas DataFrame`;
- `read_multiple_sheets` считывает и объединяет данные со всех листов рабочей книги путем конкатенации.

В функции `read_multiple_sheets` мы загружаем рабочую книгу с помощью `load_workbook` и перебираем названия листов. Мы получаем данные из каждого листа с помощью вспомогательной функции `read_single_sheet`, которая считывает данные с листа и создает для них объект `pandas DataFrame`, причем в качестве названий столбцов указываются заголовки. Наконец, используем `pd.concat` для объединения всех объектов `DataFrame`.

С помощью этих пользовательских функций мы можем легко считать несколько листов из файла Excel и получить объединенный набор данных. Это позволяет выполнять различные действия с этими данными, в том числе анализировать и визуализировать их.

Понимание методов эффективной работы с несколькими листами повышает вашу способность взаимодействовать со сложными файлами Excel и извлекать ценные данные из различных наборов.

Резюме

В этой главе мы обсудили процесс импорта данных из таблиц Excel в среду программирования. Мы рассмотрели функциональные возможности таких библиотек R, как `readxl`, `xlsx` и `openxlsx`, предоставляющих эффективные решения, которые позволяют извлекать данные и выполнять с ними различные действия. Кроме того, мы представили пользовательскую функцию `read_excel_sheets`, с помощью которой можно упростить процесс извлечения данных из нескольких листов, содержащихся в файлах Excel. Что касается Python, то мы обсудили необходимые для работы с Excel пакеты `pandas` и `openpyxl` и продемонстрировали их возможности на практических примерах. К этому моменту вы должны иметь твердое представление об этих инструментах и их функциях, с помощью которых можно обрабатывать данные Excel и выполнять их анализ.

В следующей главе мы обсудим способы записи данных в электронные таблицы Excel.

2 Запись данных в электронные таблицы Excel

Это может показаться странным, но специалисты по работе с данными нередко используют Excel для анализа и представления данных. Экспорт данных из среды R или Python в Excel может быть полезен по нескольким причинам и позволяет пользователям задействовать возможности сразу двух платформ. Excel — широко распространенная программа для работы с электронными таблицами, известная своим удобным интерфейсом, а R и Python — мощные языки статистического программирования. Экспортируя данные из R и Python в Excel, пользователи могут с помощью уже знакомых им универсальных функций Excel выполнять дальнейший анализ информации, визуализировать и распространять данные.

Excel предоставляет множество инструментов, таких как сводные таблицы, диаграммы и условное форматирование, которые позволяют пользователям исследовать и представлять данные более интерактивным и интуитивно понятным способом. Эти функции дают возможность быстро исследовать данные, выявлять тенденции и создавать отчеты и презентации высокого профессионального уровня.

Кроме того, экспорт данных в Excel может облегчить сотрудничество с коллегами или заинтересованными лицами, незнакомыми с R и Python или статистическим программированием. Excel — широко известный и доступный инструмент, который используется в различных отраслях для анализа данных и составления отчетов. Экспортируя данные в Excel, пользователи могут поделиться ими с теми, кто предпочитает работать с электронными таблицами, что облегчает сотрудничество и обмен знаниями.

Еще одна причина для экспорта данных из R и Python в Excel заключается в возможности воспользоваться обширной экосистемой надстроек и расширений Excel. Эта программа предлагает множество специализированных инструментов, которые могут пригодиться в процессе анализа данных, например Power Query, Power Pivot и Solver. Эти инструменты предоставляют дополнительные функции для очистки данных, выполнения сложных расчетов и оптимизации, которые могут быть недоступны или не слишком удобны в R и Python. Экспорт данных в Excel позволяет пользователям применить эти инструменты и воспользоваться всеми возможностями обширной экосистемы Excel.

Таким образом, экспорт данных из R и Python в Excel позволяет использовать удобный интерфейс этой программы, мощные возможности обработки данных, их визуализации и анализа, а также обеспечить эффективное взаимодействие с широким кругом пользователей.

В этой главе мы рассмотрим следующие темы:

- пакеты для записи данных в файлы Excel;
- создание листов Excel и выполнение с ними различных действий с помощью Python;
- простой экспорт данных в Excel с помощью `pandas`;
- использование пакета `openpyxl` для обработки данных в Excel;
- выбор между `openpyxl` и `pandas`;
- прочие альтернативы.

Технические требования

В этой главе мы будем использовать встроенный набор данных Iris, который очень хорошо подходит для демонстрационных целей.

Файлы с кодом из примеров главы доступны по адресу https://github.com/PacktPublishing/Extending-Excel-with-Python-and-R/tree/main/Chapter2.

Пакеты для записи данных в файлы Excel

В этом разделе мы рассмотрим несколько библиотек, которые можно использовать для записи объектов `data.frame/tibble` в файлы Excel, в частности `writexl`, `openxlsx` и `xlsx`.

Далее мы перечислим все пакеты, укажем, где можно найти документацию по функциям, позволяющим записывать данные в Excel, и рассмотрим параметры этих функций.

Пакет writexl

Пакет `writexl` предоставлен консорциумом rOpenSci и доступен по адресу https://docs.ropensci.org/writexl/reference/write_xlsx.html.

Для работы этой библиотеки не требуется ни Java, ни Excel.

Запись данных в Excel выполняется с помощью функции `write_xlsx()`. Пройдемся по ее параметрам и разберем ее псевдовызов.

Сначала рассмотрим сам вызов функции `write_xlsx()`:

```
write_xlsx(
    x,
    path = tempfile(fileext = ".xlsx"),
    col_names = TRUE,
    format_headers = TRUE,
    use_zip64 = FALSE
)
```

Теперь рассмотрим каждый из параметров в этом коде:

- `x` — это объект `DataFrame` или именованный список таких объектов, которые представляют собой листы в файле `.xlsx`;
- `path` — имя файла, в который будут записываться данные. Здесь можно ввести что-то вроде `tempfile(fileext = .xlsx")`;
- `col_names` — флаг, указывающий на необходимость записи названий столбцов в верхней части файла;
- `format_headers` — параметр, который центрирует названия столбцов (`col_names`) в файле `.xlsx` и выделяет их полужирным начертанием;
- `use_zip64` — использует формат ZIP64 (https://en.wikipedia.org/wiki/Zip_(file_format)#ZIP64) для обеспечения поддержки файлов `.xlsx` размером более 4 Гбайт, читать которые могут не все платформы.

Вот простой пример использования этой функции:

```
write_xlsx(list(mysheet = iris), path = "./iris_data_written.xlsx")
```

Пакет openxlsx

Пакет `openxlsx` можно найти по адресу https://ycphs.github.io/openxlsx/; для записи данных в Excel мы будем использовать функцию `write.xlsx()`. Начнем с того, что рассмотрим полный вызов этой функции и ее параметры:

```
write.xlsx(
    x,
    file,
    asTable = FALSE,
    overwrite = TRUE,
    …
)
```

Теперь пройдемся по параметрам функции:

- `x` — это объект `DataFrame` или (именованный) список объектов, которые могут быть обработаны функцией `writeData()` или `writeDataTable()` для записи в файл;

- `file` — путь, указывающий место для сохранения файла `.xlsx`;
- `asTable` — если значением данного параметра является `TRUE`, то для записи `x` в файл будет использоваться функция `writeDataTable()`, а не `writeData()` (значением по умолчанию является `FALSE`);
- `overwrite` — перезаписывает существующий файл (значением данного параметра по умолчанию является `TRUE`);
- `...` — дополнительные аргументы, передаваемые функции `buildWorkbook();`. Чтобы получить дополнительные сведения, вы можете ввести код `?openxlsx::buildWorkbook` в консоли R.

Теперь рассмотрим пример кода, позволяющего записать набор данных Iris в файл:

```
openxlsx::write.xlsx(
    x = iris,
    File = paste0(getwd(), "/iris.xlsx"
)
```

Далее рассмотрим последний пакет — `xlsx`.

Пакет xlsx

Пакет `xlsx` можно найти по адресу https://github.com/colearendt/xlsx; для записи данных в Excel мы будем использовать функцию `write.xlsx()`. Обратите внимание: эта функция имеет то же имя, что и функция из библиотеки `openxlsx`. Когда функции, содержащиеся в двух или более разных пакетах, имеют одинаковые имена, может образоваться конфликт пространств имен. Избежать этого несложно, но могут возникнуть некоторые неудобства. Можно написать `xlsx::write.xlsx()`. Рассмотрим полный вызов этой функции и параметры, которые ей передаются:

```
write.xlsx(
    x,
    file,
    sheetName = "Sheet1",
    col.names = TRUE,
    row.names = TRUE,
    append = FALSE,
    showNA = TRUE,
    password = NULL
)
```

Теперь пройдемся по параметрам:

- `x` — объект `DataFrame`, подлежащий записи в рабочую книгу;
- `file` — путь к выходному файлу;

- `sheetName` — строка символов, содержащая название листа;
- `col.names` — логическое значение, указывающее на то, должны ли названия столбцов объекта `x` быть записаны в файл вместе с этим объектом;
- `row.names` — логическое значение, указывающее на то, должны ли названия строк объекта `x` быть записаны в файл вместе с этим объектом;
- `append` — логическое значение, указывающее на то, следует ли добавить `x` в конец существующего файла. Если этим значением является `TRUE`, то файл считывается с диска. В противном случае создается новый файл;
- `showNA` — логическое значение. Если равно `FALSE`, то значения `NA` будут сохранены в виде пустых ячеек;
- `password` — строка, содержащая пароль.

Простой вызов этой функции будет иметь следующий вид:

```
xlsx::write.xlsx(x = iris,
File = paste0(getwd(), "/iris.xlsx"
)
```

Теперь, когда мы рассмотрели три разные функции, важно оценить, сколько времени занимает запись на диск и насколько большим оказывается выходной файл при использовании каждой из них. Для проверки скорости записи мы будем использовать библиотеку `rbenchmark`. Кроме того, мы задействуем библиотеку `dplyr`, чтобы упорядочить результаты по скорости записи. После этого мы определим выходной файл, имеющий наименьший размер.

Код для выполнения этой задачи представлен на рис. 2.1.

Этот код R используется для сравнения производительности трех различных пакетов, `writexl`, `openxlsx` и `xlsx`, при записи объектов `DataFrame` в файлы Excel. Ниже описаны выполняемые этим кодом операции.

1. Сначала происходит загрузка нескольких библиотек (`rbenchmark`, `xlsx`, `writexl`, `openxlsx` и `dplyr`) с помощью функции `library()`. Эти библиотеки предоставляют функции, которые будут использоваться далее в коде. Если их у вас нет, то установите их с помощью команды наподобие `install.packages("package")`.
2. Переменной `n` присваивается значение `5`. Она определяет количество повторных запусков кода для измерения производительности каждого пакета.
3. Функция `benchmark()` вызывается для сравнения производительности трех пакетов в плане записи данных в Excel. Она принимает несколько аргументов.
   - Первый аргумент, `writexl`, — имя, которое присваивается первому тесту. Внутри фигурных скобок вызывается функция `write_xlsx()` из пакета `writexl` для записи набора данных `iris` во временный файл Excel.

```r
library(rbenchmark)
library(xlsx)
library(writexl)
library(openxlsx)
library(dplyr)

n <- 5

benchmark(
  "writexl" = {
    writexl::write_xlsx(iris, tempfile())
  },
  "openxlsx" = {
    openxlsx::write.xlsx(iris, tempfile())
  },
  "xlsx" = {
    xlsx::write.xlsx(iris, paste0(tempfile(),".xlsx"))
  },
  replications = n,
  columns = c(
    "test","replications","elapsed","relative","user.self","sys.self")
) |>
  arrange(relative)
```

Рис. 2.1. Сравнение производительности разных функций при записи файлов

- Второй аргумент, `openxlsx`, — имя, которое присваивается второму тесту. Внутри фигурных скобок вызывается функция `write.xlsx()` из пакета `openxlsx` для записи набора данных `iris` во временный файл Excel.
- Третий аргумент, `xlsx`, — имя, которое присваивается третьему тесту. Внутри фигурных скобок вызывается функция `write.xlsx()` из пакета `xlsx` для записи набора данных `iris` во временный файл Excel с расширением `.xlsx`.
- Аргумент `replications` имеет значение `n`, указывающее количество повторений каждого теста.
- Аргумент `columns` задает столбцы, которые необходимо добавить в вывод. В данном случае это названия тестов, количество повторений, прошедшее время, относительная производительность, а также пользовательское и системное время.

4. Полученные результаты сравнительного анализа (бенчмаркинга) передаются (`|>`) в функцию `arrange()` из пакета `dplyr`, которая используется для сортировки результатов в порядке возрастания значений производительности.

   В нашем примере были получены следующие результаты:

```
  test      replications elapsed relative  user.self  sys.self
1 writexl      5     0.03   1.000      0.01        0.00
2 openxlsx     5     0.32   10.667     0.01        0.00
3 xlsx         5     0.88   29.333     0.81        0.01
```

   Итак, приведенный выше код загружает необходимые библиотеки, проводит сравнительный анализ производительности трех различных пакетов (`writexl`, `openxlsx` и `xlsx`) в плане записи данных в Excel и сортирует результаты в порядке возрастания. Цель заключается в сравнении эффективности этих пакетов при записи набора данных `iris` в файл Excel. Важно отметить, что в данном случае на результаты влияет множество факторов, в том числе используемая операционная система. Теперь посмотрим, как соотносятся размеры итоговых файлов (рис. 2.2).

```
writexl::write_xlsx(iris, tmp1 <- tempfile())
file.info(tmp1)$size
[1] 8497

openxlsx::write.xlsx(iris, tmp2 <- tempfile())
file.info(tmp2)$size
[1] 9628

xlsx::write.xlsx(iris, tmp3 <- paste0(tempfile(),".xlsx"))
file.info(tmp3)$size
[1] 7904
```

Рис. 2.2. Сравнение размеров итоговых файлов

Этот код на языке R, используя различные пакеты для записи набора данных `iris` в файлы Excel, выполняет ряд задач (описаны ниже), а затем извлекает размеры полученных файлов в байтах.

- При вызове функции `write_xlsx()` из пакета `writexl` ей передаются два аргумента: набор данных `iris` и путь к временному файлу, сгенерированному с помощью функции `tempfile()`. Функция `write_xlsx()` записывает набор данных `iris` во временный файл Excel.

- Функции `file.info()` в качестве аргумента передается путь к временному файлу (`tmp1`). Она извлекает информацию о файле, в том числе его размер. Для извлечения размера файла используется атрибут `$size`.
- При вызове функции `write.xlsx()` из пакета `openxlsx` ей передаются два аргумента: набор данных `iris` и путь к еще одному временному файлу, сгенерированному с помощью функции `tempfile()`. Функция `write.xlsx()` записывает набор данных `iris` во временный файл Excel.
- Как было указано во *втором пункте данного списка*, для получения информации о размере файла вызывается функция `file.info()`, которой в качестве аргумента передается путь к временному файлу (`tmp2`).
- При вызове функции `write.xlsx()` из пакета `xlsx` ей передаются два аргумента: набор данных `iris` и путь к временному файлу, сгенерированному с помощью комбинации функций `tempfile()` и `paste0()`. Функция `write.xlsx()` записывает набор данных `iris` во временный файл Excel с расширением `.xlsx`. В случае с пакетом `xlsx` мы используем функцию `paste0()` и указываем расширение файла, так как в данной функции это не делается по умолчанию, поэтому пользователь должен быть внимательным и правильно указывать расширение.
- Для получения размера файла снова вызывается функция `file.info()`, принимающая в качестве аргумента путь к временному файлу (`tmp3`).

Итак, приведенный выше код использует различные пакеты (`writexl`, `openxlsx` и `xlsx`) для записи набора данных `iris` в три разных файла Excel. Затем он определяет размеры этих файлов с помощью функции `file.info()`. Его цель заключается в том, чтобы сравнить размеры полученных файлов Excel при использовании этих разных пакетов. Как и в предыдущем примере, на размеры этих файлов может повлиять множество факторов, обсуждение которых выходит за рамки этой книги, в частности использование различных систем и конфигураций.

Развернутый обзор и выводы

В предыдущем подразделе вы научились записывать объекты `data.frame` в файл Excel с помощью трех разных пакетов. Мы выяснили, что эти три пакета различаются по скорости записи данных в файл и по размеру выходного файла. Представленный выше сравнительный анализ может оказаться полезным, если мы стремимся к максимизации скорости записи и/или к минимизации размера файла.

Если речь идет о скорости, то сравнительный анализ в R проводится по нескольким причинам.

- **Точность.** Анализ можно использовать для точного измерения скорости работы различных функций. Это важно, так как вы можете выбрать самую быструю функцию для решения стоящей перед вами задачи.
- **Последовательность.** Результаты сравнительного анализа можно использовать для последовательного измерения скорости работы различных функций с течением времени, что позволяет выявлять изменения в их производительности.
- **Надежность.** Результаты сравнительного анализа можно использовать для надежного измерения скорости работы различных функций на разных платформах. Это позволяет убедиться в точности и воспроизводимости результатов подобных анализов.

Помимо всего прочего, сравнительный анализ можно использовать для выявления узких мест в вашем коде. Благодаря этому вы сможете повысить производительность кода, оптимизировав те его фрагменты, на выполнение которых уходит больше всего времени.

Ниже перечислены некоторые из наиболее популярных пакетов R, предназначенных для проведения сравнительного анализа:

- `microbenchmark` предоставляет простые и удобные функции для проведения сравнительного анализа кода на языке R;
- в `rbenchmark` есть более полный набор функций для проведения сравнительного анализа, чем в `microbenchmark`;
- `rbenchmark2` является ответвлением пакета `rbenchmark` и предоставляет дополнительные возможности, в частности, позволяет провести сравнительный анализ нескольких ядер.

При выборе пакета для проведения сравнительного анализа важно учитывать свои потребности и доступные функции. Например, если вам нужно проанализировать большой объем кода, то лучше выбрать пакет, поддерживающий *параллельный бенчмаркинг*.

Выбрав пакет для бенчмаркинга, вы можете использовать его для сравнения скорости работы различных функций. Для этого вам нужно создать объект `benchmark`, содержащий подлежащие сравнению функции, а затем использовать его для запуска этих функций и измерения времени их выполнения.

По результатам бенчмаркинга можно выявить самую быструю функцию, предназначенную для решения вашей задачи. Затем вы можете использовать эту информацию для повышения производительности вашего кода.

Итак, в этом разделе вы узнали не только о том, как записывать данные в файл Excel, но и о том, как делать это с помощью различных библиотек R. Помимо изучения различных методов достижения одной и той же цели, вы увидели, что реализации пакетов различаются размерами выходного файла и скоростью записи данных в Excel. Далее мы выполним аналогичное упражнение с использованием Python.

Создание листов Excel и работа с ними с помощью Python

Задача экспорта данных в Excel довольно часто возникает при анализе данных и подготовке отчетов. Интерфейс программы облегчает визуализацию, анализ и распространение данных.

В следующих разделах мы рассмотрим способы решения таких задач, как создание новых рабочих книг, добавление листов в существующие книги, удаление листов и обработка данных внутри рабочей книги Excel. Язык Python предусматривает несколько библиотек, которые позволяют решать эти задачи простым и эффективным способом. Однако прежде чем переходить к ним, поговорим о том, зачем вообще может понадобиться экспорт данных в Excel.

Зачем экспортировать данные в Excel

Экспорт данных в Excel дает несколько преимуществ. Начнем с того, что Excel предоставляет удобную среду для изучения и визуализации данных, благодаря которой пользователи могут легко сортировать, фильтровать и анализировать их. Кроме того, обширные возможности форматирования Excel позволяют создавать профессиональные отчеты или делиться данными с заинтересованными лицами, далекими от программирования. Excel также поддерживает различные формулы и функции, с помощью которых вы можете легко выполнить расчеты, используя экспортированные данные.

Простой экспорт данных в Excel с помощью pandas

Как мы уже говорили, `pandas` — популярная библиотека Python для работы с данными, которая предоставляет мощные инструменты их анализа. Кроме того, она предлагает удобные функции для экспорта данных в Excel. `pandas` позволяет без труда преобразовать данные в листы или рабочие книги Excel.

Пакет pandas предусматривает метод DataFrame.to_excel(), позволяющий экспортировать данные в файл Excel с помощью нескольких строк кода. Рассмотрим пример:

```
import pandas as pd

# Создание объекта DataFrame с образцом данных
data = {
    'Name': ['John', 'Jane', 'Mike'],
    'Age': [25, 30, 35],
    'City': ['New York', 'London', 'Sydney']
}
df = pd.DataFrame(data)

# Экспорт объекта DataFrame в файл
df.to_excel('data.xlsx', index=False)
```

Этот код ничего не возвращает, но у него есть побочный эффект: он создает файл data.xlsx, содержащий экспортированные данные (рис. 2.3).

```
Python 3.11.3 (tags/v3.11.3:f3909b8, Apr  4 2023, 23:49:59) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import pandas as pd
>>> # Create a DataFrame with sample data
>>>
>>> data = {
...     'Name': ['John', 'Jane', 'Mike'],
...     'Age': [25, 30, 35],
...     'City': ['New York', 'London', 'Sydney']
... }
>>>
>>> df = pd.DataFrame(data)
>>> # Export the DataFrame to an Excel file
>>>
>>> df.to_excel('data.xlsx', index=False)
>>>
```

Рис. 2.3. Экспорт данных в Excel с помощью pandas

Пакет pandas хорошо справляется с простым экспортом данных, однако вам может потребоваться больший контроль над рабочей книгой Excel. Следующие несколько подразделов посвящены более сложным способам обработки данных Excel с помощью пакета openpyxl. Мы рассмотрим возможности данного пакета, начиная с таких основ, как создание новой рабочей книги путем добавления и удаления листов, и заканчивая обработкой имеющихся на листе данных.

Использование пакета openpyxl для манипулирования данными в Excel

В этом разделе мы рассмотрим пакет `openpyxl`, который позволяет осуществлять более точный контроль над процессом записи данных в Excel.

Создание новой рабочей книги

Прежде чем начать работать с листами Excel в среде Python, нужно создать новую рабочую книгу. Пакет `openpyxl` предоставляет интуитивно понятный API для создания, изменения и сохранения рабочих книг Excel. Следующий фрагмент кода демонстрирует процесс создания новой рабочей книги:

```
import openpyxl

# Создание новой рабочей книги
workbook = openpyxl.Workbook()
```

Как и в предыдущем примере, этот фрагмент кода ничего не возвращает, но у него есть побочный эффект: он создает рабочую книгу (рис. 2.4).

```
Python 3.11.3 (tags/v3.11.3:f3909b8, Apr  4 2023, 23:49:59) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import openpyxl
>>> # Create a new workbook
>>>
>>> workbook = openpyxl.Workbook()
>>>
```

Рис. 2.4. Создание рабочей книги с помощью пакета openpyxl

Добавление листов в рабочую книгу

Теперь, когда у нас есть рабочая книга, мы можем добавить в нее листы, чтобы распределять данные по разделам или категориям. Пакет `openpyxl` предусматривает простой метод `create_sheet()`, который позволяет добавлять листы в рабочую книгу. Рассмотрим пример:

```
import openpyxl

# Создание новой рабочей книги
workbook = openpyxl.Workbook()

# Добавление нового листа
workbook.create_sheet(title="Sheet2")

# Сохранение изменений
workbook.save("example.xlsx")
```

В результате мы получаем объект рабочего листа `openpyxl`, который можем использовать в дальнейшем. Затем мы можем сохранить полученную рабочую книгу и тоже использовать ее позже при необходимости.

В предыдущем примере мы пытаемся сохранить рабочую книгу, с которой работаем. Если она открыта в Excel, то эта попытка окажется неудачной и приведет к получению непонятного сообщения об ошибке, связанной с COM-системами. Убедитесь в том, что закрыли экземпляр Excel, прежде чем сохранять свою работу в среде Python! Это предупреждение будет актуально для большинства примеров из книги, поэтому помните о нем при чтении последующих глав.

Удаление листа

Иногда нам может понадобиться удалить лист из рабочей книги. Метод `remove()` из пакета `openpyxl` позволяет удалить лист по его имени. В примере ниже показан процесс такого удаления. Обратите внимание, что мы не собираемся сохранять результат, поэтому сохраненная ранее версия файла останется неизменной.

```
import openpyxl

# Загрузка существующей рабочей книги
workbook = openpyxl.load_workbook("example.xlsx")

# Удаление листа
sheet_name = "Sheet2"
sheet = workbook[sheet_name]
workbook.remove(sheet)
```

Как и в предыдущих случаях, у кода есть побочный эффект (удаление листа), но сам он не возвращает никакого значения (рис. 2.5).

```
Python 3.11.3 (tags/v3.11.3:f3909b8, Apr  4 2023, 23:49:59) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import openpyxl
>>> # Load an existing workbook
>>>
>>> workbook = openpyxl.load_workbook("example.xlsx")
>>> # Delete a sheet
>>>
>>> sheet_name = "Sheet2"
>>> sheet = workbook[sheet_name]
>>> workbook.remove(sheet)
>>>
```

Рис. 2.5. Удаление листа с помощью пакета openpyxl

В этом примере мы использовали созданный ранее лист. Метод `load_workbook()` из пакета `openpyxl` применяется для загрузки существующей рабочей книги, после чего мы удаляем лист по имени, используя метод `remove()`.

Выполнение различных действий с существующей рабочей книгой

Такие библиотеки Python, как `openpyxl`, позволяют выполнять различные действия с имеющимися рабочими книгами. С их помощью мы можем изменять значения в ячейках, применять форматирование, вставлять формулы и не только. Рассмотрим пример обновления значения ячейки в существующей рабочей книге:

```
import openpyxl

# Загрузка существующей рабочей книги
workbook = openpyxl.load_workbook("example.xlsx")

# Добавление нового листа
workbook.create_sheet(title="Sheet1")

# Выбор листа
sheet_name = "Sheet1"

sheet = workbook[sheet_name]

# Обновление значения в ячейке
sheet["A1"] = "Hello, World!"

# Сохранение изменений
workbook.save("example.xlsx")
```

Этот фрагмент кода позволяет напрямую изменить значение ячейки в листе Excel (рис. 2.6).

```
Python 3.11.3 (tags/v3.11.3:f3909b8, Apr  4 2023, 23:49:59) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import openpyxl
>>> # Load an existing workbook
>>>
>>> workbook = openpyxl.load_workbook("example.xlsx")
>>> # Add a new sheet
>>>
>>> workbook.create_sheet(title="Sheet1")
<Worksheet "Sheet1">
>>> # Select a sheet
>>>
>>> sheet_name = "Sheet1"
>>> sheet = workbook[sheet_name]
>>> # Update a cell value
>>>
>>> sheet["A1"] = "Hello, World!"
>>> # Save the changes
>>>
>>> workbook.save("example.xlsx")
>>>
```

Рис. 2.6. Обновление значения ячейки с помощью пакета openpyxl

Результат получился именно таким, которого мы и ожидали (помните, что изменение, связанное с удалением листа, не было сохранено) (рис. 2.7).

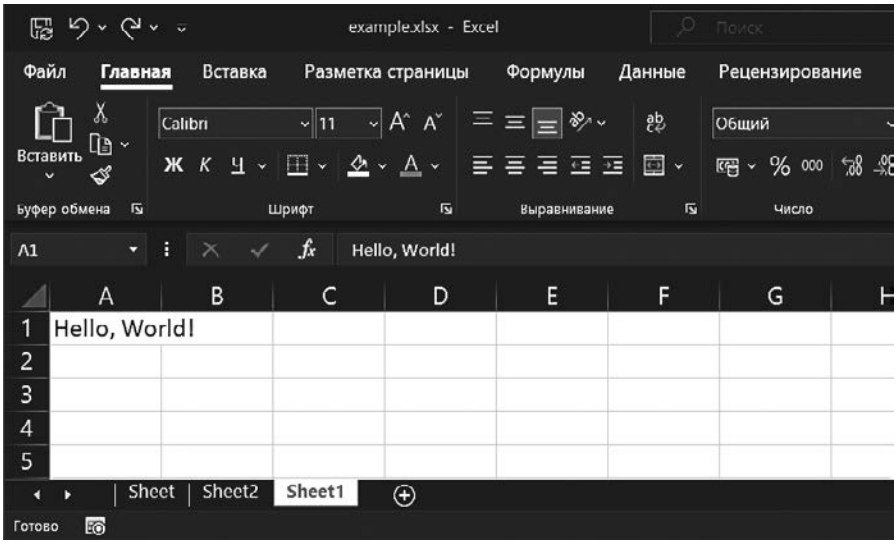


Рис. 2.7. Результат наших усилий — фраза Hello, World! в ячейке A1 листа Sheet1

Выбор между openpyxl и pandas

Оба пакета отлично подходят для экспорта данных в Excel. Пакет `openpyxl` предназначен специально для работы с файлами Excel и предоставляет широкие возможности по созданию, изменению и сохранению рабочих книг. Пакет `pandas`, в свою очередь, предлагает высокоуровневый интерфейс для работы с данными и удобные методы их экспорта в Excel, которые идеально подходят для тех случаев, когда вам не требуется ничего, кроме дампа данных.

Если вы хотите осуществлять более точный контроль над структурой файла Excel, например добавлять форматирование, формулы или диаграммы, то рекомендуем выбрать пакет `openpyxl`, который позволяет работать непосредственно с базовыми объектами Excel и тем самым дает больше возможностей. Если же вы в основном занимаетесь обработкой данных и хотите просто экспортировать объекты `DataFrames` в Excel, не используя специфические функции этой программы, то выбирайте `pandas`. Этот пакет абстрагирует некоторые низкоуровневые детали и предоставляет более простой интерфейс для экспорта данных. Такой простой абстрактный способ обработки листов и управления ими также можно найти в R-пакетах `openxlsx` и `xlsx`, предоставляющих собственные реализации подобных функций.

Прочие альтернативы

Помимо `pandas` и `openpyxl`, существуют и другие библиотеки для экспорта данных в Excel из среды Python. Самые популярные альтернативы — `XlsxWriter`, `xlrd` и `xlwt`. Эти библиотеки предлагают различные функции и возможности, выбор которых зависит от ваших конкретных потребностей. Например, `XlsxWriter` делает акцент на производительности и поддерживает расширенные функции Excel, тогда как `xlrd` и `xlwt` предоставляют функции для чтения и записи файлов Excel в старых форматах (`.xls`).

Резюме

В этой главе мы рассмотрели процессы записи данных в Excel с помощью различных библиотек R и Python и проведения сравнительного анализа их производительности. Кроме того, мы обсудили экспорт данных в Excel и разобрались, как это сделать с помощью библиотеки `pandas`. Затем мы описали процесс создания листов Excel и их обработки с помощью пакета `openpyxl`, показали, чем различаются `openpyxl` или `pandas`, а также упомянули прочие альтернативы. Используя возможности этих библиотек, вы сможете легко экспортировать данные из среды Python в Excel, обеспечивая их эффективный анализ, упрощая создание отчетов и совместную работу.

В следующей главе вы узнаете о том, как запустить код VBA из среды R и Python.

3 Запуск кода VBA из среды R и Python

Интеграция различных языков программирования позволяет получить дополнительные возможности и оптимизировать рабочие процессы. При работе с файлами Excel для автоматизации задач часто используется язык *VBA (Visual Basic for Applications)*. Однако в некоторых ситуациях вы можете захотеть запустить код VBA из среды R или Python, чтобы с помощью сильных сторон этих языков выполнять обработку, анализ и визуализацию данных.

Выполнение кода VBA, содержащегося в файле Excel, с помощью R или Python обеспечивает гибкий подход к использованию существующих макросов VBA и расширению функциональных возможностей Excel. Такая интеграция позволяет специалистам по работе с данными, аналитикам и разработчикам задействовать файлы Excel в своих рабочих процессах, сочетая преимущества VBA с аналитическими возможностями языков R или Python.

Запуская код VBA из среды R или Python, вы можете автоматизировать сложные процессы, выполнять различные операции над данными, генерировать отчеты и взаимодействовать с функциями Excel на программном уровне. Эта возможность позволяет пользователям работать с большими наборами данных, применять передовые методы их обработки и получать индивидуализированные результаты.

Для выполнения кода VBA из файла Excel можно использовать несколько библиотек и пакетов R и Python. Эти инструменты предоставляют API и функции для взаимодействия с Excel и выполнения макросов VBA непосредственно из сценариев, избавляя вас от необходимости вмешиваться в данный процесс.

В этой главе мы поговорим о различных подходах к выполнению кода VBA из файла Excel с помощью R и Python. Мы рассмотрим практические примеры, в которых показывается, как интегрировать эти языки с Excel и использовать совокупную мощь этих платформ для автоматизации решения задач и улучшения рабочих процессов, связанных с анализом данных.

Сочетание VBA с R или Python позволяет пользователям вывести свои Excel-проекты на новый уровень, повысив эффективность, точность и производительность.

Далее мы подробно обсудим эту комбинацию инструментов и широкий спектр ее применения.

В этой главе мы рассмотрим следующие темы:

- установка и описание R-пакета `RDCOMClient`;
- выполнение кода VBA с помощью `RDCOMClient`;
- интеграция Python с VBA с помощью `pywin32`;
- автоматизация задач в Excel.

Технические требования

Для изучения материалов главы вам нужно будет установить следующие библиотеки:

- `RDCOMClient` для R;
- `pywin32` для Python.

Файлы с кодом из примеров этой главы доступны по адресу https://github.com/PacktPublishing/Extending-Excel-with-Python-and-R/tree/main/Chapter%203.

Установка и описание R-пакета RDCOMClient

`RDCOMClient` — R-пакет, который служит мостом между R и *объектной моделью компонентов* (COM, component object model) Microsoft, позволяя пользователям взаимодействовать с COM-объектами из среды R. С помощью `RDCOMClient` можно использовать возможности таких COM-приложений, как Microsoft Excel, Word, PowerPoint и Outlook, для автоматизации задач, обработки данных и интеграции языка R с другими программными системами.

Прежде чем приступить к изучению `RDCOMClient`, необходимо разобраться с концепцией COM-объектов. COM — двоичный стандарт, позволяющий различным программным компонентам взаимодействовать друг с другом с помощью разных языков программирования и платформ. В контексте `RDCOMClient` под COM-объектами понимаются специфические объекты, предоставляемые приложениями на базе COM, доступ к которым можно осуществлять программно, как и выполнение с ними различных действий.

Пакет `RDCOMClient` предоставляет набор функций и методов для взаимодействия с COM-объектами, облегчая автоматизацию задач и извлечение данных из COM-приложений. Рассмотрим некоторые из ключевых возможностей этого пакета.

- **Создание объектов и подключение к ним.** Благодаря `RDCOMClient` пользователи могут создавать COM-объекты и подключаться к ним, устанавливая канал

связи между средой R и целевым приложением. Например, вы можете создать объект приложения Excel и получить доступ к его функциям непосредственно из среды R.

- **Получение доступа к методам и свойствам.** После подключения к COM-объекту `RDCOMClient` может вызывать методы, а также получать или изменять свойства объекта. Эта функциональность позволяет автоматизировать сложные задачи, обрабатывать данные и настраивать поведение приложения.
- **Работа с коллекциями и рабочими листами.** С помощью `RDCOMClient` вы можете получать доступ к таким коллекциям внутри COM-объектов, как рабочие книги и листы или диапазоны Excel. Благодаря этой возможности вы можете извлекать, обрабатывать и анализировать данные непосредственно из среды R, используя всю мощь встроенных функций Excel.
- **Обработка событий.** Пакет `RDCOMClient` поддерживает обработку событий, позволяя пользователям реагировать на события, вызываемые COM-объектами. Так, вы можете написать код R, который будет выполняться каждый раз, когда в Excel происходит определенное событие, например изменение значения ячейки или активация рабочего листа.
- **Обработка ошибок и управление памятью.** Пакет `RDCOMClient` предусматривает механизмы для обработки ошибок, которые могут возникнуть при взаимодействии с COM-объектами, обеспечивая надежность вашего кода. Вдобавок он управляет процессами выделения и очистки памяти, предотвращая ее утечки и оптимизируя использование ресурсов.

Универсальность пакета `RDCOMClient` открывает широкие возможности для его применения. Ниже описаны несколько вариантов.

- **Автоматизация работы в Excel.** `RDCOMClient` позволяет пользователям автоматизировать решение таких повторяющихся задач Excel, как извлечение данных, форматирование, создание диаграмм и отчетов. В результате может существенно повыситься производительность и точность рабочих процессов, связанных с анализом данных.
- **Работа с документами Word.** С помощью `RDCOMClient` можно программно создавать, изменять и извлекать содержимое из документов Word, что позволяет автоматизировать процессы создания документов, форматирования и интеграции данных.
- **Интеграция с Outlook.** Пакет `RDCOMClient` допускает интеграцию с Outlook, позволяя пользователям автоматизировать управление электронной почтой, календарное планирование, синхронизацию контактов и другие процессы.
- **Презентации PowerPoint.** Пользователи могут использовать `RDCOMClient` для динамического создания и изменения презентаций PowerPoint, автоматизируя создание слайдов, форматирование и встраивание диаграмм или таблиц, исходя из результатов анализа данных.

Теперь, когда мы выяснили, на что способен пакет `RDCOMClient`, рассмотрим процесс его установки.

Установка RDCOMClient

Для установки R-пакета, например библиотеки `dplyr`, мы обычно набираем в командной строке команду наподобие `install.packages("dplyr")`.

В случае с библиотекой `RDCOMClient` этот процесс будет несколько иным. Как правило, при получении пакетов из репозитория RStudio по умолчанию используется глобальный (CDN) репозиторий RStudio, однако при установке данного пакета мы дадим специальные инструкции.

> **ПРИМЕЧАНИЕ**
>
> Пакет RDCOMClient доступен только для ОС Windows.

Команда установки будет выглядеть так:

```
install.packages(
"RDCOMClient",
repos = "http://www.omegahat.net/R",
type = "win.binary"
)
```

Теперь, когда мы установили `RDCOMClient`, можно приступить к работе с этой библиотекой. В следующем разделе мы рассмотрим несколько примеров ее использования. Если способ ее установки, описанный выше, кажется вам не очень удобным, то попробуйте следующий вариант:

```
Install.packages("devtools")
Library(devtools)
Install_github("omegahat/RDCOMClient")
```

Выполнение кода VBA с помощью RDCOMClient

Для решения этой задачи нам понадобится новая рабочая книга, которую мы назовем `mult_by_rand_ch3`.

На первом листе (`Sheet1`) мы создадим два столбца с названиями `Record` и `Value`, в которых будут содержаться простые числа от 1 до 10. После этого нужно будет создать простой сценарий VBA, который мы будем запускать из библиотеки `RDCOMClient`.

Напишем макрос, который будет умножать значения из столбца `Value` на случайное число с помощью функции `RAND()`.

Обсудим процесс создания этого макроса и принцип его работы. Для начала рассмотрим следующий код VBA:

```
Sub MultiplyByRandom()
    Dim rng As Range
    Dim cell As Range
    ' Установка нужного диапазона ячеек на листе Sheet1
    Set rng = Sheets("Sheet1").Range("B2:B11")
    ' Перебор ячеек заданного диапазона
    For Each cell In rng
        ' Умножение значения ячейки на RAND() и сохранение результата
        ' в соседней ячейке
        cell.Offset(0, 1).Value = cell.Value * Rnd()
    Next cell
End Sub
```

Для создания макроса необходимо перейти на вкладку Разработчик или нажать Alt+F11, чтобы открыть редактор Visual Basic.

Вставьте новый модуль, выбрав пункт меню Вставка ▸ Модуль. После этого вы можете ввести указанный выше код в окно редактора, а затем закрыть его.

Чтобы внести некоторую ясность, разберемся в том, что делает каждая из строк кода макроса.

- `Sub MultiplyByRandom()` определяет начало подпрограммы VBA с именем `MultiplyByRandom`.
- В `Dim rng As Range` и `Dim cell As Range` объявляются две переменные `Range` с именами `rng` и `cell`, которые будут использоваться для хранения диапазонов и отдельных ячеек соответственно.
- `Set rng = Sheets("Sheet1").Range("B2:B11")` настраивает переменную `rng` так, чтобы она ссылалась на диапазон ячеек с `B2` по `B11` листа `Sheet1`, в которых хранятся числа.
- `For Each cell In rng` запускает цикл, перебирающий все ячейки в диапазоне `rng`. В ходе каждой итерации текущая ячейка присваивается переменной `cell`.
- `cell.Offset(0, 1).Value = cell.Value * Rnd()` умножает значение в текущей ячейке на случайное число, сгенерированное с помощью функции `Rnd()`, и сохраняет результат в соседней ячейке, полученной с помощью метода `Offset`, который сдвигает ссылку на один столбец вправо (`0` обозначает строки, а `1` — столбцы).

- `Next cell` обозначает конец текущей итерации цикла и переход к следующей ячейке диапазона, обеспечивая повторение этого процесса до тех пор, пока все ячейки не будут обработаны.
- `End Sub` завершает работу подпрограммы `MultiplyByRandom`.

Чтобы запустить данный макрос, мы можем написать код на языке R. Мы сделаем это с помощью библиотеки `RDCOMClient` и используем метод `$Run()` из созданного нами объекта Excel `Workbook`.

Далее приводится описание каждой строки кода R.

Эта строка загружает библиотеку `RDCOMClient`, которая позволяет взаимодействовать с такими приложениями Microsoft Office, как Excel:

```
# Загрузка библиотеки
library(RDCOMClient)
```

В следующих строках определяются переменные, хранящие путь к файлу и имя рабочей книги Excel. Приведенные здесь значения работают только у нас, авторов книги, так что вам следует обновить их с учетом особенностей вашего проекта. Для этого вы можете использовать команду наподобие `paste0(getwd(), "/")`. Переменные `f_path`, `f_chapter` и `f_name` задают путь к каталогу, имя подкаталога и имя файла соответственно. Функция `paste0()` используется для конкатенации этих переменных в целях создания полного пути к файлу.

```
# Указание пути к файлу
f_path <- "C:/Users/steve/Documents/GitHub/Extending-Excel-with- Python-
and-R/"
f_chapter <- "chapter3/"
f_name <- "mult_by_rand_ch3.xlsm"
f <- paste0(f_path, f_chapter, f_name)
```

Следующие строки кода создают экземпляр приложения Excel с помощью функции `COMCreate()`. Данное приложение представляет переменная `xl_app`. Затем с помощью метода `Open()` свойства `Workbooks()` приложения Excel открывается указанная рабочая книга (`f`). Наконец, фрагмент `xl_app[['Visible']] <- TRUE` делает приложение Excel видимым.

```
# Создание экземпляра приложения Excel
xl_app <- COMCreate("Excel.Application")
xl_wkbk <- xl_app$Workbooks()$Open(f)
xl_app[['Visible']] <- TRUE
```

Здесь переменной `macro_name` присваивается имя макроса `MultiplyByRandom`, который будет выполняться в Excel:

```
macro_name <- "MultiplyByRandom"
```

Эта строка выполняет макрос `MultiplyByRandom` в приложении Excel. Для запуска указанного макроса используется метод `Run()` приложения Excel.

```
# Запуск макроса
xl_app$Run(macro_name)
```

Следующие команды сохраняют рабочую книгу и закрывают ее с помощью метода `close()` объекта `xl_wkbk`. Аргумент `TRUE` указывает на то, что изменения должны быть сохранены перед закрытием. Наконец, для закрытия приложения Excel (`xl_app`) используется его метод `Quit()`:

```
# Сохранение и закрытие приложения
xl_wkbk$close(TRUE); xl_app$Quit()
```

Итак, данный код открывает рабочую книгу Excel с помощью `RDCOMClient`, выполняет макрос `MultiplyByRandom` в книге, сохраняет изменения и закрывает книгу и приложение Excel.

Теперь посмотрим, как этот процесс работает в Python.

Интеграция VBA с Python с помощью pywin32

В этом разделе вы научитесь запускать код VBA из среды Python и осуществлять плавную интеграцию этих двух языков. Кроме того, мы рассмотрим огромные возможности, которые она предоставляет в плане автоматизации задач в Excel, расширения функциональности и использования этого приложения в рабочих процессах Python.

Мы расскажем о том, зачем запускать код VBA из Python, как настроить необходимую для этого среду в ОС Windows, а также изложим процесс написания и выполнения кода VBA. Приступим.

Зачем запускать код VBA из среды Python

Прежде чем обсуждать детали, разберемся, зачем в принципе может понадобиться запускать код VBA из среды Python.

Программа Excel, обладая огромным набором функций и возможностей, служит важнейшим инструментом анализа данных, создания отчетов и автоматизации задач. Однако встроенных функций Excel может оказаться недостаточно для выполнения сложных вычислений или операций с данными. Именно в таких случаях можно прибегнуть к интеграции Python и VBA.

Python предоставляет богатую экосистему, которая позволяет выполнять анализ данных и осуществлять машинное обучение. Такие библиотеки, как `pandas`, `NumPy`

и `SciPy`, предлагают эффективные инструменты обработки данных, статистического анализа и моделирования. Благодаря гибкости библиотек Python вы можете расширить возможности Excel и упростить решение сложных задач, связанных с анализом данных.

Интеграция Python с VBA дает возможность задействовать возможности обоих языков программирования. Python предоставляет надежную и универсальную среду для анализа данных, а с помощью VBA можно автоматизировать конкретные задачи Excel и получить доступ к расширенным функциям этой программы. Такая синергия позволяет дополнить возможности Excel функциями библиотек Python, эффективно обрабатывать большие наборы данных и легко выполнять сложные вычисления и преобразования данных.

Преимущества выполнения кода VBA из среды Python не ограничиваются анализом данных. Вы можете использовать возможности обширной экосистемы Python для решения таких задач, как веб-скрейпинг (извлечение информации со страниц веб-ресурсов), обработка текста, машинное обучение и интеграция с внешними API. Комбинируя возможности Python и VBA, вы можете создавать динамичные и эффективные рабочие процессы, выходящие за рамки Excel.

Кроме того, интеграция Python с VBA предоставляет возможности для совместной работы и обмена кодом. Популярность Python среди специалистов по работе с данными, аналитиков и разработчиков способствует формированию обширного сообщества и накоплению общих знаний. Интеграция Python с Excel, осуществляемая с помощью VBA, позволяет преодолеть разрыв между этими двумя сферами, помогая аналитикам данных, разработчикам и опытным пользователям Excel сотрудничать и обмениваться опытом.

Таким образом, благодаря интеграции Python с VBA вы можете раскрыть весь потенциал Excel, использовать сильные стороны обоих языков программирования и вывести на новый уровень свои навыки анализа данных, создания отчетов и автоматизации.

Настройка среды

Для успешного выполнения кода VBA из Python нужно настроить среду, установив требуемые зависимости и сконфигурировав необходимые соединения. В этом подразделе мы рассмотрим три простых этапа процесса настройки.

Установка библиотеки pywin32

Библиотека `pywin32` служит мостом между Python, Windows API и COM-объектами, позволяя Python взаимодействовать с объектной моделью Excel и выполнять код VBA.

Чтобы установить `pywin32`, можно воспользоваться менеджером пакетов, например `pip`, выполнив следующую команду в командной строке или терминале:

```
python -m pip install pywin32==306
```

Результат выполнения этого кода выглядит так (рис. 3.1).

```
PS C:\Users\david_1q5aypk\Extending-Excel-with-Python-and-R> &
c:/Users/david_1q5aypk/Extending-Excel-with-Python-and-R/bookvenv/Scripts/Activate.ps1
(bookvenv) PS C:\Users\david_1q5aypk\Extending-Excel-with-Python-and-R> pip install pywin32
Collecting pywin32
  Downloading pywin32-306-cp311-cp311-win_amd64.whl (9.2 MB)
     ---------------------------------------- 9.2/9.2 MB 22.6 MB/s eta 0:00:00
Installing collected packages: pywin32
Successfully installed pywin32-306
(bookvenv) PS C:\Users\david_1q5aypk\Extending-Excel-with-Python-and-R>
```

Рис. 3.1. Установка pywin32 в ОС Windows с помощью команды pip

Данный код устанавливает пакет `pywin32` и его зависимости, позволяя Python взаимодействовать с Excel.

Установка соединения с Excel

После инсталляции `pywin32` мы можем установить соединение с Excel, которое позволит на программном уровне получать доступ к рабочим книгам, листам, диапазонам ячеек и функциям Excel из среды Python.

Для установки соединения мы можем воспользоваться модулем `win32com.client` из пакета `pywin32`. Вот пример того, как это можно сделать:

```
import win32com.client as win32

excel_app = win32.Dispatch("Excel.Application")
```

Если среда настроена правильно, то этот код ничего не возвратит (рис. 3.2). Но если код возвращает ошибку `com_error`, обратитесь к подразделу «Обработка ошибок, возникающих при настройке среды» данной главы.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\Users\david_1q5aypk\Extending-Excel-with-Python-and-R> & c:/Users/david_1q5aypk/Extending-Excel-wit
(bookvenv) PS C:\Users\david_1q5aypk\Extending-Excel-with-Python-and-R> & c:/Users/david_1q5aypk/Extendin
Python 3.11.4 (tags/v3.11.4:d2340ef, Jun  7 2023, 05:45:37) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import win32com.client as win32
>>> excel_app = win32.Dispatch("Excel.Application")
>>> vba_interface = excel_app.VBE
>>>
```

Рис. 3.2. Проверка соединения с Excel

В предыдущем фрагменте кода мы импортировали модуль `win32com.client` и создали новый экземпляр приложения Excel с помощью метода `win32.Dispatch`. В результате между Python и Excel было установлено соединение, позволяющее нам взаимодействовать с объектами Excel и выполнять код VBA.

Создание интерфейса для взаимодействия с кодом VBA

Теперь, когда соединение установлено, можно создать интерфейс, который позволит выполнять код VBA из среды Python. С помощью этого интерфейса мы сможем отправлять команды, вызывать функции VBA и обращаться к VBA-макросам. Создать интерфейс можно с помощью объекта `excel_app`, который мы получили на предыдущем этапе:

```
vba_interface = excel_app.VBE
```

Если вы получите ошибку Programmatic access to Visual Basic Project is not trusted (Программный доступ к проекту Visual Basic не является доверенным), то поищите решение здесь: https://stackoverflow.com/questions/17033526/programmatic-access-to-visual-basic-project-is-not-trusted-from-iis.

В приведенном выше коде мы получаем доступ к *редактору VBA* (VBA editor, VBE) с помощью объекта `excel_app`. Это позволяет использовать функции VBA, запускать код на этом языке, обрабатывать модули и взаимодействовать с объектами Excel из среды Python.

Пройдя эти три этапа, вы получите среду для беспрепятственного выполнения VBA-кода из Python.

В следующих разделах мы подробно рассмотрим процесс выполнения кода VBA и взаимодействия с объектами Excel, а также различные варианты использования комбинации Python и VBA.

Обработка ошибок, возникающих при настройке среды

Существует вероятность, что выполнение последней строки кода приведет к возникновению ошибки Project is not trusted (Проект не является доверенным). Как следует из самого текста ошибки, такая ситуация связана с тем, что VBA не относится к доверенным ресурсам в настройках безопасности Excel. Чтобы получить возможность осуществлять программный доступ к VBA, необходимо изменить соответствующие настройки безопасности.

> **ПРИМЕЧАНИЕ**
>
> Это изменение чревато последствиями для безопасности, обсуждение которых выходит за рамки данной книги, поэтому изменяйте настройки только в том случае, если готовы пойти на соответствующий риск.

Чтобы изменить настройки безопасности, вам придется создать новый ключ в реестре и добавить к нему новое свойство, запустив оболочку PowerShell от имени администратора. Выполните следующий код:

```
New-Item -path HKLM:\Software\Microsoft\Office\16.0\Excel\ -Name "Security"
Set-ItemProperty -path HKLM:\Software\Microsoft\Office\16.0\Excel\Security
-Name "AccessVBOM" -Value 1
```

После этого снова запустите код Python, чтобы убедиться в правильности настройки среды.

Написание и выполнение кода VBA

Настроив среду, мы можем приступать к написанию кода VBA и его запуску из Python. В этом разделе мы обсудим различные подходы и приемы, позволяющие взаимодействовать с Excel, запускать VBA-макросы и возвращать результаты в среду Python.

Для начала рассмотрим ключевые аспекты процесса написания и выполнения кода VBA.

Использование модуля win32com.client

Модуль `win32com.client` из библиотеки `pywin32` предлагает удобный способ создания COM-интерфейса и взаимодействия с Excel из среды Python. С его помощью можно получать доступ к объектам Excel, открывать рабочие книги, совершать различные действия с листами и выполнять VBA-макросы.

В примере ниже показан процесс открытия рабочей книги Excel и выполнения макроса VBA с помощью `win32com.client`. Прежде чем запускать этот код, вы можете проверить наличие макроса в файле `iris_data.xlsm`, выбрав пункт меню Разработчик ▸ Макрос (или Visual Basic).

```
import win32com.client as win32
import os

excel_app = win32.Dispatch("Excel.Application")

path = os.getcwd().replace('\'','\\') + '\\'

workbook = excel_app.Workbooks.Open(path+"iris_data.xlsm")
excel_app.Run("examplePythonVBA")
workbook.Close(SaveChanges=True)
excel_app.Quit()
```

В данном случае мы используем библиотеку `os` и характерный для ОС Windows разделитель, чтобы гарантировать корректность используемого абсолютного пути к рабочему каталогу. Как уже говорилось в предыдущих главах, вам необходимо

запустить код Python из папки, в которой находится файл (то есть из рабочего каталога), либо указать полный путь к нему.

Данный код ничего не возвращает, поскольку результат его работы виден только в Excel (рис. 3.3).

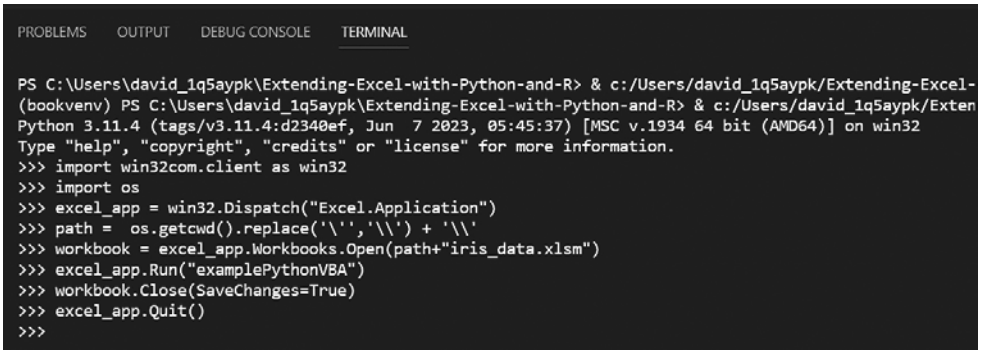


Рис. 3.3. Запуск VBA-макроса из среды Python

В приведенном выше фрагменте кода мы создаем экземпляр приложения Excel с помощью `win32.Dispatch` и открываем рабочую книгу, используя метод `Workbooks.Open`. Затем мы выполняем VBA-макрос `examplePythonVBA` с помощью `excel_app.Run`. Наконец, мы закрываем рабочую книгу, не сохраняя изменений, и выходим из приложения Excel.

Макрос просто создает новый лист с коротким сообщением, содержащимся в ячейке.

Вы можете открыть рабочую книгу `.xlsm` после выполнения этого кода, чтобы убедиться в том, что макрос сработал.

Взаимодействие с объектами Excel

С помощью модуля `win32com.client` вы можете получить доступ к различным объектам Excel, таким как рабочие листы, диапазоны и диаграммы, и на программном уровне выполнять с ними различные действия. Например, вы можете записывать данные в определенный диапазон ячеек, форматировать ячейки, создавать диаграммы и выполнять вычисления. В примере ниже показан способ записи данных в рабочий лист Excel с помощью Python:

```
import win32com.client as win32
import os

excel_app = win32.Dispatch("Excel.Application")
path = os.getcwd().replace('\'','\\') + '\\'
```

```
workbook = excel_app.Workbooks.Open(path+"iris_data.xlsm")
worksheet = workbook.Worksheets("Sheet1")

data = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
for row_index, row_data in enumerate(data, start=1):
for col_index, value in enumerate(row_data, start=1):
worksheet.Cells(row_index, col_index).Value = value

workbook.Close(SaveChanges=True)
excel_app.Quit()
```

В случае успешного выполнения этот код ничего не возвращает (рис. 3.4).

Рис. 3.4. Результат взаимодействия с ячейками

В данном коде мы открываем рабочую книгу Excel, обращаемся к рабочему листу `Sheet1`, перебираем значения в объекте `data` и записываем их в соответствующие ячейки с помощью их *свойства*.

Возвращение результатов в среду Python

Результаты выполнения кода VBA в Excel можно возвратить в среду Python, чтобы впоследствии их можно было анализировать или обрабатывать. Один из способов сделать это — использовать объектную модель Excel для получения доступа к определенным значениям или диапазонам и их извлечения в переменные Python.

Извлечь данные из рабочего листа Excel в список Python можно так:

```
import win32com.client as win32
import os

excel_app = win32.Dispatch("Excel.Application")
path = os.getcwd().replace('\'','\\') + '\\'

workbook = excel_app.Workbooks.Open(path+"iris_data.xlsm")
worksheet = workbook.Worksheets("Sheet1")
```

```
# Получение доступа к нескольким ячейкам с использованием нотации Range
range_of_cells = worksheet.Range('A1:C3')

# Считывание значений из диапазона ячеек
values = range_of_cells.Value

workbook.Close(SaveChanges=False)
excel_app.Quit()

print(values)
```

Результатом выполнения этого кода является кортеж кортежей (рис. 3.5):

```
((1.0, 2.0, 3.0), (4.0, 5.0, 6.0), (7.0, 8.0, 9.0))
```

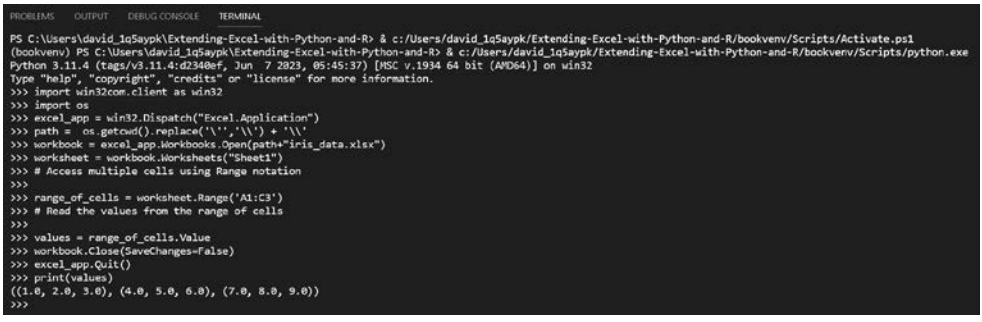


Рис. 3.5. Извлечение данных из Excel

В приведенном выше фрагменте кода мы определяем диапазон ячеек Excel, извлекаем из них значения с помощью свойства `Value` и сохраняем их в списке Python в целях дальнейшей обработки или анализа.

Использование модуля `win32com.client` и объектной модели Excel позволяет эффективно писать и выполнять код VBA из среды Python.

Автоматизация задач в Excel

В этом разделе мы рассмотрим практические примеры автоматизации распространенных операций Excel с помощью кода VBA, запускаемого из среды Python. Интегрируя Python с VBA, вы сможете оптимизировать рабочие процессы анализа данных и значительно повысить свою продуктивность.

- **Выполнение различных действий с данными.** Интеграция Python с VBA позволяет автоматизировать выполнение таких операций над данными в Excel, как

сортировка, фильтрация записей, объединение наборов данных и выполнение их сложных преобразований. Например, используя Python, можно получать данные из внешних источников, обрабатывать их с помощью таких библиотек, как `pandas` или `NumPy`, а затем заполнять рабочий лист Excel преобразованными данными, используя код VBA. Такая интеграция позволяет автоматизировать выполнение повторяющихся операций над данными и обеспечить их согласованность в разных источниках.

- **Форматирование.** Автоматизация задач форматирования в Excel позволяет сэкономить много времени и сил. С помощью Python и VBA вы можете определить правила форматирования и применить их к определенным ячейкам, диапазонам или целым рабочим листам. Это касается таких параметров, как стили шрифтов, границы ячеек, цвет фона и форматирование чисел. Сочетая гибкие инструменты Python и возможности форматирования VBA, вы сможете легко создавать динамичные и визуально приятные отчеты или дашборды Excel.
- **Создание диаграмм.** Функции Excel для создания диаграмм можно эффективно использовать, автоматизировав процесс создания графиков с помощью Python. Вы можете извлекать данные из различных источников, выполнять необходимые вычисления или агрегацию в среде Python, а затем динамически генерировать диаграммы с помощью кода VBA. Благодаря такой автоматизации вы сможете создавать интерактивные визуализации данных на основе анализа, проведенного в Python, экономя время и получая больший контроль над процессом создания диаграмм.

  Поскольку эта тема довольно обширна и важна, мы подробно обсудим ее в отдельной главе.
- **Сложные расчеты.** Программа Excel хорошо известна благодаря мощным встроенным функциям и формулам. Комбинируя Python и VBA, можно еще больше расширить возможности Excel в плане выполнения вычислений. Вы можете использовать библиотеки Python для сложных математических или статистических расчетов и интегрировать полученные результаты в Excel с помощью VBA. Такая интеграция позволяет выполнять сложные вычисления, симуляции или прогнозное моделирование в привычной среде Excel.

  Автоматизация задач в Excel, осуществляемая за счет интеграции Python и VBA, позволит вам сэкономить время, устранить ошибки, допускаемые при выполнении соответствующих операций вручную, и повысить эффективность процессов анализа данных. Приведенные примеры кода и пояснения служат отправной точкой для дальнейшего изучения обширных возможностей автоматизации. Экспериментируйте с различными сценариями, адаптируйте код под собственные нужды и раскройте весь потенциал Python и VBA, связанный с автоматизацией задач в Excel.

Плюсы и минусы запуска кода VBA из среды Python

В этом подразделе мы рассмотрим плюсы и минусы запуска кода VBA из среды Python. Понимая преимущества и ограничения данного подхода, вы сможете принимать взвешенные решения, выбирая подходящий инструмент для стоящей перед вами задачи.

Начнем с положительных сторон этого подхода.

- **Гибкость и мощь.** Комбинируя Python и VBA, вы можете воспользоваться преимуществами обоих языков программирования. Python предлагает обширную экосистему библиотек и инструментов для анализа данных, научных вычислений и автоматизации. VBA, в свою очередь, предоставляет широкие функциональные возможности, позволяющие использовать встроенные в Excel функции, формулы и макросы. Благодаря такому сочетанию можно решать сложные задачи и эффективно автоматизировать операции, выполняемые в Excel.
- **Интеграция с внешними источниками данных.** Python отлично подходит для подключения к внешним источникам данных, в частности к базам данных, API и веб-ресурсам. Запуская код VBA из среды Python, вы можете легко объединить эти внешние источники данных с Excel. Python позволяет получать данные, выполнять вычисления и преобразования, а затем обновлять рабочую книгу Excel с помощью кода VBA. Благодаря такой интеграции вы можете использовать всю мощь Python для обработки данных и их анализа, задействуя при этом функции Excel для визуализации и создания отчетов.
- **Автоматизация и эффективность.** Выполнение кода VBA из среды Python позволяет автоматизировать повторяющиеся задачи в Excel, что способствует повышению эффективности вашей деятельности. Вы можете оптимизировать свои рабочие процессы, автоматизировав импорт/экспорт данных, их очистку, форматирование и создание отчетов. Благодаря такой автоматизации можно устранять ошибки, допускаемые при выполнение этих операций вручную, снижать трудозатраты и высвобождать время для более важных задач, связанных с анализом и принятием решений.

Теперь перейдем к аспектам данного подхода, нуждающимся в улучшении.

- **Совместимость и зависимость от платформы.** Выполнение кода VBA из среды Python в основном поддерживается в системах на базе ОС Windows. Если вы используете другую операционную систему, например macOS или Linux, то можете столкнуться с проблемами совместимости. Кроме того, совместимость с различными версиями приложений Excel или Office может быть разной, что следует иметь в виду при распространении своих решений, предусматривающих интеграцию Python и VBA.

- **Сложность освоения и требования к уровню навыков.** Успешное выполнение кода VBA из среды Python требует определенного уровня владения обоими языками. Вы должны понимать синтаксис и возможности VBA, чтобы выполнять автоматизацию задач в Excel. В свою очередь, владение синтаксисом Python и понимание возможностей этого языка позволит взаимодействовать с Excel и выполнять VBA-код. Возможно, вам придется пройти некое обучение и получить определенный опыт, особенно если вы совершенно не знакомы с каким-то из этих языков.
- **Сопровождение и обновления.** Любая подобная интеграция предполагает необходимость сопровождения и обновления. Если в экосистеме Excel или Python произойдут изменения или обновления, то вам может потребоваться соответствующим образом адаптировать свой код. Кроме того, обеспечение совместимости и функциональности различных версий Excel и Python может требовать периодического обновления и тестирования.

Несмотря на вышеперечисленные нюансы, выполнение кода VBA из среды Python позволяет автоматизировать задачи в Excel, использовать внешние источники данных и налаживать эффективные рабочие процессы их анализа. Задействуя сильные стороны Python и VBA, вы можете раскрыть весь потенциал Excel в своих проектах Python, повысить свою продуктивность и расширить возможности взаимодействия с данными.

Резюме

В этой главе вы узнали о том, как интегрировать R и Python с VBA с помощью пакета `RDCOMClient` и библиотеки `pywin32` соответственно. Вы научились запускать VBA-код из Python, настраивать необходимую для этого среду и автоматизировать задачи в Excel. Кроме того, мы обсудили плюсы и минусы подобной интеграции.

Эти знания помогут вам усовершенствовать свои навыки автоматизации, которые пригодятся вам при работе в Excel.

В следующей главе мы перейдем к более сложным темам, в основе которых лежат знания, полученные вами на данный момент.

4 Дальнейшая автоматизация: планирование задач и электронной рассылки

Иногда нам приходится тратить бессчетное количество часов на создание, форматирование и отправку таблиц Excel по электронной почте. Однако в этих временны́х затратах нет необходимости! Такие языки программирования, как R и Python, позволяют автоматизировать эти рутинные задачи и экономить драгоценное время.

Представьте, что вам нужно поместить данные о продажах в профессионально оформленную электронную таблицу и отправить ее нескольким адресатам по электронной почте. Вместо того чтобы вручную вводить и форматировать эти данные, вы можете использовать возможности R или Python, которые позволяют упростить рабочий процесс.

В языке R есть пакет `taskscheduleR` для планирования задач и автоматического запуска сценариев в указанное время. С его помощью вы можете наладить повторяющийся процесс создания таблиц Excel и их отправки, в котором не требуется ваше участие. Для работы с электронной почтой можно использовать пакеты `RDCOMClient`, `Windows365R` или `blastula`, которые позволяют напрямую взаимодействовать с программой Outlook и сервисом Gmail. Это означает, что вы можете без особых усилий составлять и отправлять электронные письма с прикрепленными таблицами, изменяя их содержимое и список адресатов по мере необходимости.

Язык Python позволяет достичь такого же уровня автоматизации. В частности, вы можете использовать библиотеку `pywin32` для рассылки красиво оформленных писем через сервис Gmail. Она легко интегрируется в ваш код, позволяя прикреплять таблицы Excel и легко настраивать содержимое письма и список адресатов.

В этой главе мы рассмотрим следующие темы:

- установка и описание библиотеки `taskscheduleR`;
- использование пакетов `RDCOMClient`, `Windows365R` и `blastula` для работы с электронной почтой в Outlook или Gmail;
- запланированный запуск сценариев Python;
- электронные уведомления и автоматизация с помощью Python.

Технические требования

Файлы с кодом из примеров этой главы доступны по адресу https://github.com/PacktPublishing/Extending-Excel-with-Python-and-R/tree/main/Chapter%204.

Для изучения данной главы вам нужно установить следующие пакеты R и Python:

- `Blastula`;
- `Windows365R`;
- `schedule==1.2.0`;
- `apscheduler==3.10.1`.

Установка библиотеки taskscheduleR и знакомство с ней

R-пакет `taskscheduleR` позволяет запланировать выполнение R-сценариев или процессов с помощью планировщика заданий Windows. Это означает, что вы можете автоматически запускать R-процессы в определенный момент времени непосредственно из среды R. По сути, данный пакет представляет собой обертку вокруг `Schtasks.exe` — инструмента командной строки, который позволяет создавать, удалять, запрашивать, изменять, запускать и завершать выполнение запланированных задач на локальном или удаленном компьютере. Чтобы использовать пакет `taskscheduleR`, его нужно установить из репозитория CRAN. После установки пакета вам будут доступны следующие функции для запланированного запуска R-сценариев:

- `taskscheduler_create()` — создает новую запланированную задачу;
- `taskscheduler_remove()` — удаляет существующую запланированную задачу. В версии 1.8 для этого используется функция `taskscheduler_delete()`;
- `taskscheduler_get()` — получает информацию о существующей запланированной задаче.

Функция `taskscheduler_create()` принимает несколько аргументов, в том числе имя запланированной задачи, скрипт R, который требуется запустить, и время его запуска. Например, следующий код создает запланированную задачу, которая запускает R-сценарий `my_scheduled_excel_script.R` каждый день в 10:00:

```
taskscheduler_create(
  name = "My Scheduled Excel Task",
  rscript = "my_scheduled_excel_script.R",
  schedule = "DAILY",
  start_time = "10:00:00"
)
```

Установив пакеты `miniUI` и `shiny`, вы можете создавать такие задачи, используя графический интерфейс и обходясь без написания кода. Вот как можно установить эти пакеты:

```
# Установка пакета для планирования задач
install.packages("taskscheduleR")

# Установка пакетов, позволяющих использовать графический интерфейс
install.packages('miniUI')
install.packages('shiny')
```

Теперь обсудим процесс создания сценариев.

Создание сценариев

В первую очередь нужно создать сценарий, который можно запустить с помощью планировщика заданий Windows. Чтобы не усложнять пример, в этом сценарии мы просто отобразим слово `Hello` и текущие дату и время, полученные с помощью функции R `Sys.time()`. После этого мы создадим несколько заданий, запускающих этот сценарий.

Код сценария `hello_world.R` выглядит так:

```
library("tcltk")
tkmessageBox(
  title='Message',
  message = paste0("Hello, it is: ", Sys.time()),
  type = "ok"
)
```

Первая строка кода, `library("tcltk")`, загружает пакет `tcltk`, который предоставляет функции для создания и взаимодействия с *графическими интерфейсами пользователя* (Graphical User Interface, GUI) в среде R.

Вторая строка, `tkmessageBox()`, создает всплывающее окно, которое отображает сообщение и позволяет пользователю выполнить какое-либо действие, например нажать кнопку.

Функция `tkmessageBox()` принимает следующие аргументы:

- `title` — заголовок окна сообщения;
- `message` — сообщение, которое будет отображаться в окне;
- `type` — тип окна сообщения. Возможными значениями данного аргумента являются: `ok`, `okcancel`, `yesno`, `yesnocancel`, `retrycancel` и `abortretryignore`.

В данном случае мы выбрали тип `ok`. Это означает, что для закрытия окна сообщения пользователь может нажать кнопку `OK`.

Третья строка кода, `paste0("Hello, it is: ", Sys.time())`, создает строку, которая будет отображаться в окне сообщения. Она содержит информацию о текущем времени, полученную с помощью функции `Sys.time()`.

В результате запуска этого кода появится окно сообщения с указанием текущего времени и кнопкой OK, которую пользователь может нажать для закрытия окна (рис. 4.1).

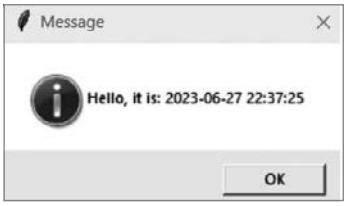


Рис. 4.1. Окно сообщения, отображаемое при запуске сценария hello_world.R

Теперь создадим пару сценариев, которые будут выводить это сообщение на экран по расписанию. Вот код, запускающий данный сценарий каждый час:

```
taskscheduler_create(
  taskname = "Hello World Hourly",
  rscript = "hello_world.R",
  schedule = "HOURLY"
)
```

Первая строка кода вызывает функцию `taskscheduler_create()` из пакета `taskscheduleR`, который позволяет создавать и управлять заданиями планировщика Windows.

Функция `taskscheduler_create()` принимает следующие три аргумента:

- `taskname` — имя запланированного задания;
- `rscript` — путь к R-сценарию, который будет запущен этим заданием;
- `schedule` — расписание запуска сценария.

В данном случае именем задания является `Hello World Hourly`, R-сценарием — `hello_world.R`, а расписанием — `"HOURLY"`.

Вторая строка кода задает `Hello World Hourly` в качестве имени задания, служащего его уникальным идентификатором в планировщике заданий.

Третья строка кода, `rscript = "hello_world.R"`, задает путь к R-сценарию, который будет запускаться запланированным заданием. Данный сценарий должен быть сохранен в месте, доступном планировщику заданий.

Четвертая строка кода, `schedule = "HOURLY"`, задает расписание, определяющее периодичность запуска сценария. Значение `"HOURLY"` говорит о том, что задание будет выполняться каждый час. Выполнение этого кода приведет к созданию нового задания планировщика, благодаря которому сценарий `hello_world.R` будет запускаться ежечасно.

Далее мы поговорим о том, как использовать пакет `RDCOMClient` при работе с Outlook, если вы не используете Microsoft 365. Это позволит создавать сценарии, выполняющие отправку электронных писем внутри других процессов.

Использование пакета RDCOMClient с Outlook

Как уже было сказано, `RDCOMClient` представляет собой мощный R-пакет, который позволяет взаимодействовать с Microsoft Outlook с помощью COM-интерфейса. Используя `RDCOMClient`, вы можете автоматизировать такие задачи в Outlook, как отправка электронных писем, получение доступа к папкам, управление встречами и т. д. Процесс отправки электронных писем через Outlook с помощью `RDCOMClient` очень прост и состоит всего из нескольких этапов. Сценарий выглядит так:

```
install.packages("RDCOMClient")
library(RDCOMClient)
OutApp <- COMCreate("Outlook.Application")
OutMail <- OutApp$CreateItem(0)
outMail[["To"]] <- "recipient@example.com"
outMail[["Subject"]] <- "Hello from RDCOMClient"
outMail[["Body"]] <- "This is the body of the email."
outMail$Attachments$Add("C:/path/to/attachment.txt")
outMail$Send()
outMail <- NULL
OutApp <- NULL
```

Разберем каждую строку приведенного выше кода.

1. Сначала необходимо установить и загрузить пакет `RDCOMClient` в среду R.
2. После установки мы загружаем пакет с помощью функции `library()`.
3. Затем создаем новый объект приложения Outlook с помощью функции `COMCreate()`.
4. В результате устанавливается соединение с приложением Outlook. Теперь мы можем создать новый объект электронного письма с помощью метода `OutApp$CreateItem()` и указать `olMailItem` в качестве его типа.
5. Далее мы задаем различные свойства письма, к которым относятся адрес получателя, тема, тело, вложения и не только. Вот пример того, как это можно сделать:

   ```
   outMail[["To"]] <- "recipient@example.com"
   outMail[["Subject"]] <- "Hello from RDCOMClient"
   outMail[["Body"]] <- "This is the body of the email."
   outMail$Attachments$Add("C:/path/to/attachment.txt")
   ```

6. Задав нужные свойства, мы можем отправить письмо с помощью метода `Send()`. Вот и все!

   Письмо с заданными свойствами будет отправлено из вашей учетной записи Outlook указанному получателю (или получателям). Не забудьте освободить COM-объект и разорвать соединение после завершения работы.

   Эти действия гарантируют правильное высвобождение ресурсов.

Используя пакет `RDCOMClient`, вы можете автоматизировать решение задач Outlook, связанных с отправкой электронной почты непосредственно из среды R, оптимизировать рабочий процесс и сэкономить время. Это позволит вам интегрировать R с Outlook и задействовать возможности этого приложения программным способом.

Использование пакетов Microsoft365R и blastula

В данном разделе мы рассмотрим пакеты `Microsoft365R` и `blastula`, с помощью которых можно отправлять электронные письма через приложение Outlook компании Microsoft. Пакет `Microsoft365R` является альтернативой библиотеке `RDCOMClient` и подходит тем, кто использует платформу Microsoft Azure вместо обычного SMTP-соединения Outlook.

Пакет Microsoft365R

`Microsoft365R` — это R-пакет, предоставляющий интерфейс для взаимодействия с облачным сервисом Microsoft 365 (ранее известным как Office 365). Он создан на основе пакета `AzureGraph`, предоставляющего высокоуровневую абстракцию для взаимодействия с API Microsoft Graph. В настоящее время пакет `Microsoft365R` поддерживает следующие сервисы Microsoft 365:

- Teams;
- Outlook;
- SharePoint Online;
- OneDrive.

`Microsoft365R` предоставляет несколько клиентских функций верхнего уровня для получения доступа к каждому из перечисленных выше сервисов. Например, функция `get_personal_onedrive()` позволяет получить доступ к вашей личной учетной записи OneDrive, а функция `get_business_outlook()` — к вашему аккаунту Outlook.

Кроме того, `Microsoft365R` предусматривает несколько вспомогательных функций для работы с API Microsoft Graph. Например, функцию `list_drive_contents()` можно использовать для получения списка содержимого диска OneDrive, а функцию `send_email()` — для отправки электронного письма из Outlook.

Библиотека хорошо документирована и содержит несколько примеров, помогающих начать работу с ней. Кроме того, она поддерживается активным сообществом и регулярно пополняется новыми функциями и исправлениями.

Ниже перечислены некоторые преимущества пакета `Microsoft365R`:

- он предоставляет простой и удобный интерфейс, позволяющий получить доступ к облачным сервисам Microsoft 365;
- он создан на основе пакета `AzureGraph`, предоставляющего высокоуровневую абстракцию для взаимодействия с API Microsoft Graph.

Таким образом, данный пакет хорошо документирован, активно сопровождается и поддерживает широкий спектр сервисов Microsoft 365. В книге мы будем использовать `Microsoft365R` только с Outlook.

Пакет blastula

R-пакет `blastula` — это инструмент, позволяющий легко создавать и отправлять электронные HTML-письма из среды R. По сравнению с другими почтовыми пакетами он имеет ряд преимуществ.

- Пакет `blastula` позволяет использовать текст в формате Markdown, блочные компоненты и даже HTML-код для создания профессиональных электронных писем с тремя областями содержимого: телом, заголовком и нижним колонтитулом.
- Пакет предоставляет функции настройки учетных данных для использования *простого протокола передачи почты* (simple mail transfer protocol, SMTP), позволяющие отправлять письма через традиционный SMTP-сервер. Если вы являетесь пользователем RStudio Connect, то можете отправлять письма через данный сервис.
- Полученное письмо будет отлично смотреться на любом экране.
- Интуитивно понятный синтаксис позволяет вам быстро и эффективно создавать код, концентрируя основное внимание на сути сообщения.

Установить пакет `blastula` можно из репозитория CRAN с помощью команды `install.packages("blastula")`.

Рассмотрим пример использования пакета `blastula` для создания и отправки простого HTML-сообщения.

1. Сначала нужно загрузить пакет `blastula` с помощью `library(blastula)` и `library(glue)`.

2. Далее мы можем использовать функцию `compose_email()` для создания электронного письма. В качестве аргумента `body` можно передать текст в формате Markdown, который будет преобразован в HTML, например:

```
email <- compose_email(
    body = md(glue(
        "Hello, this is {sale_file} contains the latest quarters data.
```

```
        Here is a picture of the graph that will be on our
        dashboard:![sales dashboard] https://dashboard.company.com/)"
    )
)
```

3. Затем мы можем просмотреть сообщение в программе просмотра, вызвав объект электронного письма:

```
email
```

4. Наконец, мы можем отправить электронное письмо через SMTP-сервер с помощью функции `smtp_send()`. Для этого нужно указать адреса электронной почты получателя и отправителя, тему и учетные данные для получения доступа к SMTP-серверу:

```
email %>%
  smtp_send(
     to = "jane_doe@example.com",
     from = "joe_public@example.net",
     subject = "Testing the `smtp_send()` function",
     credentials = creds_file("email_creds")
  )
```

Функция `creds_file()` — вспомогательная, она создает файл с учетными данными для доступа к SMTP-серверу.

Мы можем использовать функцию `create_smtp_creds_file()` для создания этого файла в интерактивном режиме:

```
create_smtp_creds_file(
  file = "email_creds",
  user = "joe_public@example.net",
  provider = "gmail"
)
```

В результате нам будет предложено ввести пароль и код двухфакторной аутентификации для Gmail (если она включена). Файл с учетными данными будет сохранен в безопасном месте и в будущем может быть использован при отправке электронных сообщений.

Теперь, когда мы рассмотрели эти пакеты по отдельности, обсудим преимущества их совместного использования.

Сначала приведем сценарий целиком, а затем поговорим о том, как он работает.

```
install.packages("blastula")
install.packages("Microsoft365R")
library(blastula)
library(Microsoft365R)

# Получение электронной почты
outlb <- get_business_outlook()
```

```
# Составление письма с помощью пакета blastula
bl_body <- "## Hello!

You can write email in **Markdown** with the blastula package."

bl_em <- compose_email(
      body=md(bl_body),
      footer=md("sent via Microsoft365R and R")
)
em <- outlb$create_email(bl_em, subject="Hello from R", to="email@ example.com")

# Добавление вложения и отправка письма
em$add_attachment("mydocument.docx")
em$send()
```

Теперь рассмотрим отдельные фрагменты сценария и выясним, что они делают.

1. Сначала мы должны установить пакеты `blastula` и `Microsoft365R`.
2. Затем мы вызываем эти библиотеки в рамках текущего сеанса с помощью функции `library()`.
3. После этого мы создаем переменную `outlb`, которая будет содержать переменную для функции `get_business_outlook()`.
4. Затем мы создаем тело письма. Для его написания можно использовать синтаксис Markdown.
5. После создания тела письма мы создаем переменную для хранения самого письма, полученного с помощью функции `compose_email()`.
6. Затем мы добавляем вложение и отправляем письмо.

Для отправки письма используется учетная запись, полученная ранее. Таким образом, приведенные выше строки кода устанавливают необходимые пакеты, загружают нужные библиотеки, получают учетную запись электронной почты, составляют письмо с помощью пакета `blastula`, добавляют вложение и, наконец, отправляют письмо с помощью пакета `Microsoft365R`.

Итак, мы обсудили способы запланированного запуска сценариев и отправки электронных писем с помощью R. Теперь посмотрим, как того же самого можно добиться с помощью Python!

Запланированный запуск сценариев Python

Автоматизация не только избавляет от необходимости выполнять сценарии вручную, но и позволяет задать периодичность их запуска или сделать так, чтобы они запускались при наступлении определенных событий. Планирование запуска сценариев Python позволяет автоматизировать повторяющиеся задачи, выполнять периодическое обновление данных, генерировать отчеты и сопровождать системы, обходясь без непосредственного вмешательства в эти процессы.

В этом разделе мы рассмотрим различные методы и инструменты для планирования запуска сценариев Python и обеспечения их эффективного и своевременного выполнения.

В ходе изучения данного раздела вы получите представление о различных методах и инструментах планирования, доступных для сценариев Python, а также знания и навыки, благодаря которым вы сможете составлять расписание запуска этих сценариев, автоматизировать рутинные задачи и работать максимально продуктивно. В чем бы вы ни выполняли эти сценарии: в Windows или Unix-подобных системах — эта глава поможет вам реализовать надежные и эффективные решения, которые позволят планировать запуск сценариев Python.

Важность запланированного запуска сценариев Python

Планирование запуска сценариев Python играет важную роль в автоматизации задач и повышении эффективности рабочего процесса в самых разных областях.

- **Автоматизация повторяющихся задач.** Многие задачи, связанные с обработкой данных, анализом и созданием отчетов, предполагают выполнение повторяющихся действий. Планирование запуска сценариев Python позволяет автоматизировать решение этих задач, а значит, сэкономить время и силы. Вне зависимости от того, идет ли речь об извлечении данных из внешних источников, выполнении вычислений, создании отчетов или обновлении баз данных, планирование запуска сценариев позволяет обеспечить последовательную и надежную автоматизацию этих процессов.
- **Периодическое обновление данных.** Многие приложения требуют регулярного обновления данных с использованием внешних источников или внутренних систем. Планирование запуска сценариев Python позволяет настроить расписание для автоматического получения, преобразования и обновления данных. Указав необходимую периодичность выполнения этих операций, вы сможете поддерживать актуальность данных, не вмешиваясь в этот процесс.
- **Создание отчетов.** Как правило, отчеты тоже создаются с определенной периодичностью, например ежедневно, еженедельно или ежемесячно. Этот процесс можно автоматизировать, обеспечив выполнение сценария Python через определенные промежутки времени. Такой подход гарантирует последовательность создаваемых отчетов и их своевременную доставку, позволяя экономить время и силы.
- **Сопровождение системы.** Вне зависимости от того, о чем идет речь: выполнении резервного копирования, обслуживании баз данных, проверке системы или выполнении ее рутинной очистки, — используя запланированный запуск сценариев Python, вы сможете поддерживать бесперебойную работу систем, не вмешиваясь в сам процесс.

- **Применение в различных отраслях и сферах реального мира.** Например, в сфере электронной коммерции запланированный запуск сценариев может использоваться для обновления сведений о товарных запасах, синхронизации цен и создания отчетов о продажах. В финансовой сфере этот подход может применяться для получения данных о фондовом рынке, оценки эффективности портфеля и формирования финансовой отчетности. В ИТ-сфере он позволяет выполнять мониторинг системы, анализ журналов и автоматическое реагирование на инциденты. Это лишь несколько примеров того, как планирование запуска сценариев Python позволяет повышать производительность и эффективность работы.

Встроенные инструменты для планирования заданий

Python предусматривает встроенные инструменты для планирования заданий, которые можно применять как в Windows, так и в Unix-подобных системах, не используя внешние библиотеки. Эти инструменты предоставляют удобный и легкий способ планирования и выполнения сценариев Python с заданной периодичностью. В этом подразделе мы рассмотрим встроенные инструменты планирования, в том числе их возможности, процесс использования и ограничения.

Планировщик заданий cron

Данный планировщик доступен в Unix-подобных операционных системах, в том числе Linux и macOS. К его ключевым особенностям можно отнести то, что он:

- позволяет автоматизировать повторяющиеся задачи, указав время, дату и периодичность их выполнения с помощью cron-выражения;
- обеспечивает гибкость при составлении расписания, позволяя запускать сценарии в определенное время, в конкретные дни недели или через регулярные промежутки времени;
- широко поддерживается и уже много лет является фактическим стандартом планирования заданий в Unix-подобных системах;
- не является специфическим для Python инструментом, но вы можете использовать его для планирования запуска сценариев Python, указав соответствующий вызов в командной строке.

Планировщик заданий ОС Windows

Данный планировщик является встроенным инструментом операционной системы Windows. Он позволяет:

- выполнять задачи на компьютерах под управлением ОС Windows в определенное время или при наступлении определенных событий;

- запланировать задачу с помощью графического интерфейса пользователя или инструмента командной строки `schtasks`;
- задать триггеры, запускающие задание однократно в указанное время либо ежедневно, еженедельно или ежемесячно, а также при наступлении определенных системных событий.

Кроме того, данный планировщик предоставляет удобный интерфейс для управления запланированными задачами, позволяющий настраивать зависимости, приоритизировать задачи и обрабатывать ошибки.

Функция Python time.sleep()

Ниже перечислены ключевые особенности данной функции.

- Хотя встроенный в Python модуль `time` не предназначен специально для планирования задач, он предоставляет простой способ добавления задержек между запусками сценариев.
- Функция `time.sleep()` приостанавливает выполнение сценария на заданное количество секунд, что позволяет контролировать временной интервал между его запусками.
- Эта функция не обладает расширенными возможностями, свойственными инструментам, предназначенным специально для планирования, но может пригодиться, если вам достаточно фиксированной задержки между запусками сценария. Но если вам требуются более точные или сложные настройки расписания, то функцию `time.sleep()` не стоит использовать в качестве механизма планирования заданий.

Следует отметить, что эти встроенные инструменты планирования имеют свои сильные и слабые стороны. Они подходят для многих ситуаций, но не обладают расширенными возможностями и гибкостью, необходимыми для составления сложных расписаний. В таких случаях более надежными решениями могут стать внешние библиотеки и инструменты.

Сторонние библиотеки для планирования задач

Встроенные в Python функции позволяют решить многие распространенные задачи планирования, но есть несколько сторонних библиотек, которые предоставляют более широкие возможности и настройки, позволяя вам контролировать расписание запуска сценариев Python и легко справляться со сложными задачами планирования.

В этом разделе мы рассмотрим несколько таких библиотек и обсудим их особенности и преимущества. Мы затронем такие темы, как составление расписаний с помощью синтаксиса cron, работа с часовыми поясами и управление конкурентным выполнением сценариев.

Библиотека schedule

`schedule` — легковесная библиотека Python, которая упрощает процесс планирования заданий. У нее есть ряд преимуществ, поскольку она:

- предлагает интуитивно понятный и простой в использовании API для создания расписаний;
- поддерживает различные способы планирования заданий, в том числе использование cron-выражений, указание конкретного времени или периодичности выполнения;
- позволяет выполнять функции, методы и даже команды оболочки в соответствии с заданным расписанием;
- хорошо подходит для небольших проектов или тех случаев, когда для составления расписания требуется минималистичное решение.

Простой и наглядный пример использования библиотеки `schedule` в Python выглядит так:

```
import schedule
import time

def job():
    print("This job is executed every day at 8:00 AM.")

# Планирование выполнения задания на 08:00 каждого дня
schedule.every().day.at("08:00").do(job)

# Обеспечение продолжения работы программы
while True:
    schedule.run_pending()
    time.sleep(1)
```

В этом примере мы импортируем библиотеку `schedule` и определяем функцию `job()`, которая выводит на экран сообщение. Затем мы используем синтаксис `schedule.every().day.at(08:00).do(job)`, чтобы запланировать выполнение функции `job` на 08:00 каждого дня. Наконец, с помощью цикла `while` мы постоянно проверяем наличие ожидающих запланированных заданий и их выполнение. Обратите внимание: данное решение предполагает блокировку консоли, то есть процесс Python, выполняющий этот бесконечный цикл, будет недоступен для других задач. Чтобы остановить работу планировщика, просто прервите работу ядра/консоли или завершите основной процесс Python.

Вы можете настроить расписание, используя различные методы, предусмотренные в библиотеке `schedule`, такие как `.every().day.at()`, `.every().monday`, `.every().hour` и др. Кроме того, можно сделать так, чтобы задания выполнялись с определенной периодичностью, или комбинировать несколько способов планирования в соответствии с вашими потребностями.

Планировщик APScheduler

Еще один популярный инструмент Python — `APScheduler` (Advanced Python Scheduler, улучшенный планировщик Python). К числу его преимуществ можно отнести то, что он:

- предоставляет богатый набор функций и различные типы триггеров, таких как конкретная дата, временные интервалы и cron-выражения;
- поддерживает множество хранилищ заданий, в том числе оперативную память, базы данных SQL и Redis;
- предоставляет расширенные возможности планирования, такие как объединение заданий, распределенное планирование и поддержка часовых поясов;
- имеет хорошо документированный API и широко используется в различных программах, начиная от простых сценариев и заканчивая сложными веб-приложениями.

Ниже приведен фрагмент кода, демонстрирующий использование библиотеки `APScheduler`:

```
from apscheduler.schedulers.blocking import BlockingScheduler

# Создание экземпляра планировщика
scheduler = BlockingScheduler()

# Определение функции задания
def send_email():
    # Код для отправки электронного письма
    print("Email sent!")

# Планирование ежечасного выполнения задания
scheduler.add_job(send_email, 'interval', hours=1)

# Запуск планировщика
scheduler.start()
```

В этом коде мы выполняем следующие действия.

1. Сначала импортируем класс `BlockingScheduler` из модуля `apscheduler.schedulers.blocking`. Затем создаем экземпляр планировщика с помощью функции `BlockingScheduler()`.
2. Далее определяем функцию `send_email()`, которая представляет собой задание, связанное с отправкой электронного письма. Внутри этой функции вы можете написать код для отправки письма или выполнения любого другого нужного действия.
3. Для планирования задачи мы используем метод планировщика `add_job()`. В данном случае в качестве задачи, подлежащей выполнению, мы указываем

функцию `send_email`. Кроме того, мы настраиваем расписание с помощью параметра `interval`, который указывает на то, что задача должна выполняться через регулярные промежутки времени. В данном примере мы сделали так, чтобы она выполнялась каждый час (`hours=1`).

4. Наконец, запускаем планировщик, вызывая метод `start()`, после чего задание начинает выполняться в соответствии с заданным расписанием.

Библиотека `APScheduler` предоставляет различные способы планирования, такие как фиксированные интервалы, cron-выражения, триггеры, основанные на дате/времени, и др. Вы можете настроить расписание в соответствии с вашими конкретными требованиями, изучив документацию и используя параметры планирования, предусмотренные в `APScheduler`.

Пакет Celery

Библиотека Celery — это распределенная очередь задач, которая может использоваться как для выполнения задач, так и для их планирования. Как и другие инструменты, она тоже обладает рядом преимуществ:

- предоставляет надежное и масштабируемое решение для планирования выполнения заданий Python в нескольких рабочих потоках или на нескольких машинах;
- поддерживает различные способы планирования, в том числе временные интервалы, cron-выражения и многое другое;
- предусматривает такие расширенные возможности, как приоритизация задач, отслеживание результатов, управление повторными попытками и составление цепочек задач;
- хорошо интегрируется с такими брокерами сообщений, как *RabbitMQ, Redis* или *Apache Kafka*, предоставляя надежные и масштабируемые решения для планирования заданий.

Используя расписания, созданные с помощью планировщика Celery, вы можете автоматизировать выполнение повторяющихся или привязанных ко времени операций в вашем приложении. Обсуждение процесса настройки и развертывания приложений Celery выходит за рамки нашей книги, поэтому здесь мы упоминаем об этом пакете лишь для полноты картины. Если хотите углубиться в изучение данной темы, то обратитесь к официальной документации, доступной по адресу https://docs.celeryq.dev/en/stable/userguide/periodic-tasks.html.

Перечисленные выше сторонние библиотеки предоставляют мощные и гибкие решения для планирования запуска сценариев Python. Вы можете выбрать ту, которая лучше всего соответствует вашим потребностям и уровню сложности стоящей перед вами задачи.

Практические примеры использования этих библиотек для планирования запуска сценариев Python мы рассмотрим чуть позже, а пока поговорим о том, как обеспечить надежность автоматизации.

Рекомендации по обеспечению надежной автоматизации

В этом подразделе мы рассмотрим основные нюансы, которые следует учитывать при составлении расписаний, управляющих выполнением сценариев Python.

Если вы будете следовать приведенным ниже рекомендациям, то сможете создавать надежные и удобные в сопровождении запланированные задачи.

Обработка ошибок и логирование

- Реализуйте надежные механизмы для обработки любых непредвиденных исключений и ошибок, которые могут возникнуть во время выполнения сценария.
- Используйте фреймворки логирования для сбора подробной информации о процессе выполнения сценария, в том числе об ошибках, предупреждениях и уведомлениях.
- Правильная обработка ошибок и логирование позволяют диагностировать проблемы, отслеживать производительность сценария и гарантировать, что запланированные задачи будут выполняться ожидаемым образом.

Управление ресурсами

- Учитывайте ресурсы, необходимые запланированным сценариям Python, в том числе ресурсы процессора, память и дисковое пространство
- Оптимизируйте использование ресурсов, выявляя и минимизируя ресурсоемкие операции и узкие места
- Убедитесь в том, что ваши сценарии правильно освобождают ресурсы, чтобы избежать их утечки или конфликтов.

Обеспечение безопасности

- Оцените последствия планирования запуска сценариев Python для безопасности, особенно если эти сценарии используют конфиденциальные данные или взаимодействуют с внешними системами.
- Принимайте необходимые меры безопасности, такие как шифрование учетных данных, защита конечных точек API и соблюдение протоколов аутентификации и авторизации.
- Регулярно анализируйте и обновляйте меры безопасности, позволяющие минимизировать потенциальные риски и уязвимости.

Тестирование и проверка

- Проводите модульное, интеграционное и сквозное тестирование сценариев Python, чтобы убедиться в том, что они надежны, работают ожидаемым образом и справляются с различными ситуациями.
- Рассмотрите возможность создания промежуточной среды, в которой вы сможете тестировать запланированные задачи перед их развертыванием в производственной среде.

Мониторинг и оповещения

- Создайте механизмы мониторинга и оповещения для отслеживания статуса выполнения запланированных задач.
- Отслеживайте производительность и состояние системы планирования, в том числе время выполнения сценариев, использование ресурсов, а также любые потенциальные ошибки или сбои.
- Настройте оповещения, чтобы получать уведомления о сбоях при выполнении задач и иметь возможность оперативно вмешаться и устранить неполадки.

Документирование и сопровождение

- Документируйте процесс планирования, в том числе конфигурацию, зависимости и любые конкретные инструкции по управлению запланированными задачами.
- Поддерживайте актуальность расписания запланированных сценариев Python, в том числе частоту и ожидаемые результаты выполнения каждой задачи.
- Регулярно пересматривайте и обновляйте настройки расписаний, сценарии и зависимости, чтобы адаптироваться к изменяющимся требованиям и технологиям.

Следуя этим рекомендациям, вы сможете обеспечить надежность, эффективность и безопасность выполнения ваших запланированных сценариев Python и поддерживать хорошо структурированную и устойчивую систему автоматизации.

О том, как применять эти рекомендации в реальных ситуациях, мы поговорим чуть позже, а пока кратко напомним, о чем шла речь в данном разделе.

- Вы изучили встроенные инструменты планирования заданий, доступные в различных операционных системах, такие как cron в Unix-подобных системах и планировщик заданий Windows, позволяющие запускать сценарии Python в определенное время или с определенной периодичностью.
- Вы познакомились со сторонними библиотеками для планирования заданий, в частности с `APScheduler` и `schedule`, которые позволяют создавать более сложные расписания с помощью cron-выражений и триггеров, запускающих выполнение сценариев через фиксированные временные интервалы, и увидели примеры применения этих двух библиотек.

- Вы узнали о рекомендациях по планированию запуска сценариев Python, в том числе связанных с обработкой ошибок, управлением ресурсами, обеспечением безопасности, тестированием, проверкой, мониторингом и оповещением, а также документированием и сопровождением.

Освоив концепции и приемы, описанные в этом разделе, вы сможете эффективно планировать запуск сценариев Python, автоматизировать решение повторяющихся задач и оптимизировать рабочий процесс.

Электронные уведомления и автоматизация с помощью Python

Как и запланированный запуск сценариев Python, отправка уведомлений по электронной почте и автоматизация позволяют существенно повысить эффективность вашей повседневной работы. В этом разделе вы узнаете, зачем отправлять электронные уведомления из Python, как настроить необходимую для этого среду и отправлять простые сообщения, после чего мы рассмотрим процесс отправки электронных уведомлений о состоянии сценария.

Важность использования электронных уведомлений в сценариях Python

Уведомления, отправляемые по электронной почте, играют важнейшую роль в автоматизации, позволяя нам своевременно получать обновления, предупреждения и отчеты. В этом подразделе мы поговорим о важности интеграции функции отправки электронных уведомлений в сценарии Python, обсудим различные варианты их использования, а также преимущества, благодаря которым сможем оптимизировать рабочие процессы и улучшить коммуникацию.

Для начала определимся с тем, зачем вообще могут пригодиться эти уведомления.

Варианты использования электронных уведомлений и преимущества их интеграции в сценарии

Уведомления, отправляемые по электронной почте, весьма универсальны и могут применяться в самых разных ситуациях. Среди наиболее распространенных вариантов их использования можно выделить следующие.

- **Уведомления об успешном завершении задачи или процесса.** Помогают информировать заинтересованных лиц о том, что все идет должным образом.
- **Сообщения об ошибках и исключениях, возникающих во время выполнения сценария.** Позволяют оповестить ответственных членов команды или администраторов, что помогает оперативно устранять неполадки и решать проблемы.

- **Уведомления о ходе выполнения задачи.** Такое информирование заинтересованных лиц или клиентов обеспечивает прозрачность рабочих процессов и поддерживает открытость взаимодействий.
- **Отчеты с результатами анализа данных.** Автоматизированное создание и доставка таких отчетов по электронной почте экономят время и силы, избавляя от необходимости создавать и распространять эти отчеты вручную.
- **Мониторинг системы.** Электронные уведомления позволяют отслеживать состояние системы, например время бесперебойной работы сервера, использование дискового пространства или производительность приложений. Мгновенные оповещения помогают устранить потенциальные проблемы до того, как они повлияют на другие операции.

Добавив функцию отправки электронных уведомлений в сценарии Python, вы получите следующие преимущества.

- **Обновление в режиме реального времени.** Уведомления позволяют отслеживать ход выполнения автоматизированных задач, их состояние и результаты.
- **Расширение возможностей для совместной работы.** Своевременная рассылка уведомлений облегчает взаимодействие и сотрудничество членов команды.
- **Обнаружение и устранение ошибок.** Получение уведомлений об ошибках или исключениях в сценариях позволяет вам быстро выявлять и решать возникающие проблемы.
- **Настройка и персонализация.** Вы можете настраивать электронные уведомления, в том числе их содержимое, форматирование и список адресатов, в соответствии с конкретными требованиями.
- **Оптимизация рабочих процессов.** Автоматизация доставки отчетов, обновлений и предупреждений позволит вам сэкономить время и силы, избавив от необходимости осуществлять рассылку вручную.

В этом разделе мы рассмотрим различные аспекты применения электронных уведомлений в сценариях Python, что позволит вам использовать этот мощный инструмент коммуникации в своих рабочих процессах автоматизации.

Для начала обсудим процесс настройки службы электронной почты.

Настройка служб электронной почты

Прежде чем приступить к отправке уведомлений по электронной почте из среды Python, необходимо настроить почтовые службы, чтобы установить соединение между сценарием и почтовым сервером.

Изучив материал этого подраздела, вы получите четкое представление о том, как настраивать почтовые службы для своих сценариев Python, а также будете знать,

как выбрать подходящего поставщика услуг электронной почты, получить данные SMTP-сервера, настроить безопасное соединение и протестировать его.

Настройка почтовой службы состоит из нескольких этапов.

Выбор поставщика услуг электронной почты

Разные поставщики услуг электронной почты предлагают различные функции и возможности. Приведенный в этом разделе код можно легко адаптировать для работы с любыми поставщиками услуг электронной почты, например с Microsoft Exchange.

Получение данных SMTP-сервера

Для осуществления программной отправки электронных писем вам нужно получить данные SMTP-сервера у своего поставщика услуг электронной почты. Эти данные необходимы для установления соединения с почтовым сервером и безопасной отправки электронных сообщений.

Вам нужно получить информацию, необходимую для эффективной настройки параметров электронной почты.

- **Адрес SMTP-сервера.** Это имя хоста или IP-адрес сервера, отвечающего за отправку исходящих электронных писем. Данная информация предоставляется поставщиком услуг электронной почты и обычно имеет формат `smtp.example.com` или `mail.example.com`. Обязательно проверьте правильность адреса SMTP-сервера, используемого вашей почтовой службой.
- **Номер порта.** Определяет конечную точку связи для SMTP-сервера. Как правило, для подключения по протоколу SMTP используются порты 25, 465 (при установке соединений, зашифрованных по протоколу SSL/TLS) и 587 (при установке соединений, зашифрованных по протоколу STARTTLS). Ваш поставщик услуг электронной почты укажет номер порта, который следует использовать для отправки сообщений.
- **Учетные данные.** Чтобы авторизоваться на SMTP-сервере, вам нужно будет указать имя пользователя и пароль. Эти данные используются для установки безопасного соединения и гарантируют, что электронные письма будут отправляться только авторизованными пользователями. Ваш поставщик услуг электронной почты предоставит вам данные, необходимые для прохождения процесса аутентификации. В некоторых случаях вместо традиционных имени пользователя и пароля для аутентификации может потребоваться ключ API или маркер доступа.

Обратите внимание: процесс получения данных SMTP-сервера может различаться у разных поставщиков услуг электронной почты. Некоторые поставщики предоставляют специальные настройки учетной записи или инструкции, в которых можно найти необходимую информацию. Если вы не знаете, где искать данные

SMTP-сервера, то обратитесь к документации или другим вспомогательным ресурсам, предоставляемым вашим поставщиком.

Получив данные SMTP-сервера, вы можете установить соединение с почтовым сервером, чтобы осуществить программную отправку электронных писем. Это позволит вам использовать возможности Python для автоматизации отправки уведомлений, обновлений и решения других коммуникационных задач в рамках ваших приложений или сценариев.

Проверка соединения

Чтобы удостовериться в успешной настройке почтовой службы, важно протестировать соединение между сценарием Python и почтовым сервером.

В следующем подразделе мы покажем, как можно проверить соединение, отправив тестовое письмо с использованием данных SMTP-сервера, предоставленных поставщиком услуг электронной почты. Это поможет убедиться в том, что соединение установлено правильно и ваш сценарий может взаимодействовать с почтовым сервером.

Отправка простых электронных писем

Настроив почтовые службы, можно переходить к следующему этапу — отправке простых электронных писем из среды Python.

Прочитав этот раздел, вы научитесь отправлять простые электронные письма из среды Python, импортировать необходимые библиотеки, создавать сообщения, устанавливать соединение с почтовым сервером и отправлять электронные письма по протоколу SMTP. Эти знания позволят вам осуществлять отправку простых текстовых сообщений из своих сценариев Python и послужат основой для создания более сложных электронных уведомлений.

Итак, отправка электронных писем подразумевает выполнение ряда действий.

- **Импорт необходимых библиотек.** Чтобы иметь возможность отправлять электронные письма из среды Python, нужно импортировать необходимые библиотеки, предоставляющие соответствующие функции. Ниже мы рассмотрим популярные библиотеки, в частности `smtplib` и `email.mime`, которые предлагают удобные функции для создания и отправки электронных писем. Мы проведем вас через весь процесс установки и покажем, как импортировать эти библиотеки в сценарий Python.
- **Создание сообщения электронной почты.** Перед отправкой электронного письма необходимо создать сообщение электронной почты, которое содержит адреса отправителя и получателя, тему и тело письма. Чуть позже мы рассмотрим процесс создания такого сообщения с помощью библиотеки `email.mime`,

позволяющей настраивать различные аспекты письма, в частности добавлять вложения или HTML-содержимое. Приведенные ниже примеры кода и пояснения помогут вам понять структуру сообщения электронной почты и разобраться с его компонентами.

- **Установка соединения с почтовым сервером.** Чтобы отправить электронное письмо, нужно установить соединение с почтовым сервером по протоколу SMTP. Ниже мы представим процесс установки такого соединения с использованием полученных ранее данных SMTP-сервера. Мы проведем вас через все его этапы, такие как создание объекта SMTP-сервера, настройка соединения и обработка возможных ошибок и исключений, которые могут возникнуть при установке соединения.
- **Отправка электронного письма по протоколу SMTP.** После установки соединения можно приступать к отправке письма. Чуть позже мы покажем, как использовать библиотеку `smtplib` для отправки сообщения электронной почты. Мы рассмотрим все необходимые для этого действия, такие как аутентификация на почтовом сервере, указание адресов отправителя и получателя, а также вызов метода `sendmail()`. Кроме того, мы обсудим способы обработки ошибок и рекомендации по обеспечению успешной доставки электронного сообщения.

Описанные выше действия реализованы в следующем фрагменте кода:

```
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart

# Определение почтового сервера и учетных данных
smtp_server = 'ваш_smtp_сервер'
smtp_port = 587
smtp_username = 'ваш_логин'
smtp_password = 'ваш_пароль'

# Создание MIME-сообщения
message = MIMEMultipart()
message['From'] = 'sender@example.com'
message['To'] = 'recipient@example.com'
message['Subject'] = 'Test Email'

# Добавление тела письма
body = MIMEText('This is the email body.')
message.attach(body)

# Установка соединения с почтовым сервером
with smtplib.SMTP(smtp_server, smtp_port) as server:
    # Запуск TLS-шифрования
    server.starttls()
```

```
# Авторизация на сервере электронной почты
server.login(smtp_username, smtp_password)

# Отправка электронного письма
server.send_message(message)
```

В этом фрагменте кода мы выполняем следующие действия.

1. Сначала импортируем библиотеку `smtplib` для установки SMTP-соединения и классы `MIMEText` и `MIMEMultipart` из модуля `email.mime` для создания сообщения электронной почты.
2. Далее определяем данные SMTP-сервера, в том числе его адрес, порт и учетные данные (имя пользователя и пароль), необходимые для прохождения аутентификации.
3. Затем создаем объект `MIMEMultipart` под названием `message`, который представляет собой сообщение электронной почты, указываем адреса отправителя и получателя, а также тему письма.
4. Далее отправляем письмо с помощью метода `send_message()` объекта SMTP-сервера, передавая сообщение в качестве аргумента.
5. Не забудьте заменить значения-заполнители (*`ваш_smtp_сервер`* и *`ваш_логин`*). Чтобы добавить тело письма, мы создаем объект `MIMEText` под названием `body` и прикрепляем его к сообщению.
6. После этого мы устанавливаем соединение с почтовым сервером с помощью класса `smtplib.SMTP`, передавая в качестве аргументов адрес сервера и порт, и запускаем процесс TLS-шифрования с помощью метода `starttls()`.
7. Затем мы авторизуемся на почтовом сервере с помощью метода `login()` и указываем имя пользователя и пароль.

Вместо *`ваш_пароль`*, `sender@example.com` и `recipient@example.com` используйте данные вашего SMTP-сервера и реальные адреса электронной почты.

Используя этот пример кода как образец, вы сможете отправлять простые сообщения электронной почты с помощью библиотек Python `smtplib` и `email.mime`.

Отправка электронных уведомлений о состоянии сценария

В этом подразделе вы узнаете о том, как определять триггеры, реализовывать логику отправки электронных уведомлений, настраивать содержимое сообщения электронной почты, а также эффективно обрабатывать ошибки, возникающие в процессе доставки электронных писем. Овладев этими навыками, вы расширите свои возможности по мониторингу сценариев, что позволит вам получать актуальную информацию о состоянии сценария или процесса. Реализация данной функ-

циональности позволяет отслеживать прогресс и результаты автоматизированных процессов и принимать необходимые меры в случае необходимости.

Для начала рассмотрим действия, которые требуются для отправки электронных уведомлений о состоянии сценария.

- **Определение триггеров.** Прежде чем отправлять электронные уведомления, необходимо задать триггеры, которые будут служить сигналом для их отправки. Этими триггерами могут быть возникновение определенных условий или наступление определенных событий во время выполнения сценария. Например, вы можете сделать так, чтобы электронное уведомление отправлялось при успешном завершении сценария, возникновении ошибки или достижении определенного этапа. Возможность определения триггеров предоставляет вам полный контроль над временем и обстоятельствами отправки электронных уведомлений.
- **Реализация логики отправки электронных уведомлений.** Этот процесс предполагает добавление в сценарий необходимого кода для проверки триггеров и инициирования процесса отправки электронного письма. Вам нужно будет импортировать необходимые библиотеки, установить соединение с почтовым сервером и настроить содержимое письма. Реализованная логика должна обеспечивать отправку электронных уведомлений в ответ на срабатывание указанных триггеров.
- **Настройка содержимого электронного письма.** Чтобы сделать свои электронные уведомления максимально актуальными и информативными, вы можете настроить их различные аспекты, такие как тема, тело и пр. Уведомление может содержать информацию о состоянии выполняемого сценария, сообщения об ошибках, релевантные данные или статистику, а также любые другие сведения, имеющие отношение к ходу выполнения сценария. Настроив содержимое электронного письма, вы сможете гарантировать, что пользователи получат информацию, которая нужна им для выполнения необходимых действий или принятия обоснованных решений.
- **Обработка ошибок, возникающих в процессе доставки электронной почты.** Отправка электронных писем предполагает взаимодействие с почтовыми серверами, и их успешная доставка зависит от множества факторов. Важно предусмотреть механизм разрешения таких потенциальных проблем, как неполадки с сетевым подключением, ошибки в работе сервера электронной почты или некорректный электронный адрес получателя. Внедрив подобный механизм, вы сможете решать проблемы, возникающие в процессе доставки электронных уведомлений. Средства логирования и составления отчетов об ошибках тоже могут быть полезны для диагностики и устранения проблем, связанных с отправкой электронной почты.

Таким образом, используя описанные в этой главе методы, вы сможете автоматизировать свои процессы и получать актуальные сведения об их выполнении.

Резюме

В данной главе вы узнали о различных инструментах для составления расписаний запуска сценариев Python, в том числе о встроенных планировщиках и таких сторонних библиотеках, как `schedule` и `APScheduler`. Мы говорили о том, почему при работе с электронными уведомлениями важно следовать рекомендациям, касающимся обработки ошибок, управления ресурсами, обеспечения безопасности, тестирования, мониторинга, документирования и сопровождения.

Теперь вы знаете, что электронные уведомления позволяют своевременно информировать заинтересованных лиц о состоянии и результатах выполнения процессов в автоматизированных системах. Вы познакомились с такими мощными библиотеками и модулями Python, как `smtplib` и `email.mime`, с помощью которых можно создавать и настраивать сообщения электронной почты. Кроме того, вы получили знания, которые помогут вам настраивать почтовые службы путем конфигурирования SMTP-серверов, использовать учетные данные и задавать другие необходимые параметры. Вдобавок вы узнали о таких R-пакетах, как `taskscheduleR`, `blastula` и `Microsoft365R`, которые выполняют аналогичные функции и отличаются простым синтаксисом, что делает их весьма мощными инструментами.

Получив представление об основных понятиях, вы научились реализовывать базовые функции отправки электронной почты и создавать простые сообщения с необходимыми заголовками и содержимым. Кроме того, вы получили представление об электронных уведомлениях как об универсальном средстве информирования заинтересованных лиц о состоянии сценария, его успешном или неудачном завершении, а также о наступлении конкретных событий в ходе его выполнения. Благодаря этим знаниям вы сможете эффективно интегрировать функции отправки электронных уведомлений в свои проекты Python.

Часть II

Наводим красоту: форматирование, графики и многое другое

В этой части мы поговорим о том, как улучшить внешний вид и повысить функциональность листов Excel. Вы научитесь форматировать данные с помощью таких библиотек R, как `styledTables`, `tablaxlsx`, `excelR`, `basictabler` и `tidyxl`, а также пакетов Python `pandas` и `openpyxl`. Вы освоите простые способы вставки графиков `ggplot2` и `matplotlib` в листы Excel и научитесь создавать сводные таблицы с помощью R-пакета `tidyquant`, библиотек Python `win32com` и `pywin32` и пакета `gt`.

5 Форматирование листа Excel

В этой главе мы рассмотрим две библиотеки R и различные библиотеки Python, которые можно использовать для форматирования таблиц и данных, представленных на рабочем листе Excel.

Мы обсудим следующие R-пакеты:

- `styledTables`;
- `basictabler`.

В дополнение к встроенному в R набору данных Iris мы создадим некоторое количество фиктивных данных, а затем применим к ним методы стилизации, предусмотренные в вышеупомянутых пакетах. Рабочие процессы использования этих методов слегка варьируются в зависимости от пакета, так что вам следует ознакомиться с ними и выбрать тот, который кажется наиболее удобным.

В разделе этой главы, посвященном инструментам Python, мы рассмотрим предоставляемые ими возможности по созданию красивых таблиц в Excel. В частности, мы будем использовать пакеты `pandas`, `openpyxl` и `pywin32`.

В этой главе мы рассмотрим следующие темы:

- установка и использование R-пакета `styledTables`;
- расширенные возможности форматирования с помощью Python.

Изучив материал этой главы, вы будете иметь четкое представление о форматировании ячеек, условном форматировании и создании сводных таблиц.

Технические требования

Файлы с кодом из примеров этой главы доступны по адресу https://github.com/PacktPublishing/Extending-Excel-with-Python-and-R/tree/main/Chapter%205.

Как уже было сказано, в этой главе мы будем использовать несколько R-пакетов. Пакет `styledTables` можно установить только с GitHub, используя пакет `devtools`.

Для изучения раздела, посвященного Python (в частности, для задач форматирования, решаемых с помощью библиотеки `pandas`), вам понадобится пакет `jinja2==3.1.2`.

Установка и использование R-пакета styledTables

Прежде чем использовать необходимые пакеты, их нужно установить. В данном разделе мы добавим пакет `styledTables`. Он отсутствует в репозитории CRAN, поэтому мы не можем установить его с помощью функции `install.packages()`. Нам придется взять его с GitHub, в связи с чем потребуется установить еще и пакет `devtools`.

Вот код, необходимый для установки этого пакета:

```
install.packages("devtools")
# Установка версии разработки с GitHub
devtools::install_github(
'R-package/styledTables',
build_vignettes = TRUE
)
```

Когда этот фрагмент кода будет выполнен, можно вызвать библиотеку в рамках текущего сеанса, введя в консоли команду `library(styledtables)`. Теперь, когда мы загрузили библиотеку, можно приступить к созданию нашего первого сценария, который будет стилизовать таблицу на основе простого критерия. Начнем:

```
library(TidyDensity)
library(styledTables)
library(xlsx)

st <- tidy_normal() |>
  styled_table(keep_header = TRUE) |>
  set_border_position("all", row_id = 1) |>
  set_bold(row_id = 1) |>
  set_fill_color("#00FF00", col_id = 3, condition = X >= 0.5)

# Открытие новой рабочей книги xlsx и создание рабочего листа
wb <- createWorkbook()
sheet <- createSheet(wb, "tidy_normal")

# Вставка стилизованной таблицы в рабочий лист
write_excel(sheet, st)

# Сохранение рабочей книги
saveWorkbook(wb, "chapter5/styledTables_test.xlsx")
```

В этом коде на языке R используется несколько библиотек (`TidyDensity`, `styledTables` и `xlsx`) для создания стилизованной таблицы из набора данных,

ее сохранения в файл Excel и применения к ней методов форматирования. Разберем его поэтапно.

1. Сначала мы загружаем необходимые библиотеки: `TidyDensity` для создания таблицы со случайными данными, сгенерированными на основе нормального распределения, `styledTables` для стилизации таблиц и `xlsx` для работы с файлами Excel.
2. Затем мы создаем и стилизуем таблицу. Данный фрагмент кода создает стилизованную таблицу `st` на основе вывода функции `tidy_normal()`, которая генерирует нормально распределенный набор данных и представляет их в виде тиббла. Затем таблица стилизуется с помощью функции `styled_table()` с параметром `keep_header = TRUE`. Это означает, что строка заголовка будет сохранена и стилизована отдельно от строк с данными. Следующие три строки кода применяют определенные стили к первой строке таблицы (строке заголовка): добавляют границу вокруг всех ячеек в первой строке, выделяют содержащийся в ней текст полужирным начертанием и задают цвет заливки (в данном случае зеленый, `#00FF00`) для ячеек в третьем столбце, значение которых (обозначенное буквой `X`) больше или равно 0,5.
3. Затем мы создаем новую рабочую книгу и рабочий лист Excel с помощью пакета `xlsx`. Для создания книги мы используем функцию `createWorkbook()`, после чего с помощью функции `createSheet()` создаем лист этой книги, названный `tidy_normal`.
4. Далее мы вставляем стилизованную таблицу в рабочий лист с помощью функции `write_excel()` из библиотеки `styledTables`. Данная функция помещает стилизованную таблицу `st` в рабочий лист `tidy_normal`.
5. Наконец, с помощью функции `saveWorkbook()` из пакета `xlsx` мы сохраняем рабочую книгу `wb` в файле Excel `styledTables_test.xlsx` в подкаталоге chapter5.

Таким образом, приведенный выше код генерирует стилизованную таблицу на основе некоторых данных (предоставленных функцией `tidy_normal()`), применяет определенное форматирование к строке заголовка и ячейкам третьего столбца, сохраняет стилизованную таблицу в файле Excel и помещает ее в рабочий лист `tidy_normal`. Полученный в итоге файл Excel будет содержать стилизованную таблицу с заданным форматированием.

Установка и использование R-пакета basictabler

R-пакет `basictabler` предоставляет простой способ создания информативных таблиц на основе объектов `DataFrame` или матриц. Эти таблицы могут быть представлены в виде HTML-кода, HTML-виджетов или электронных таблиц Excel.

Прежде чем создавать объект `basictabler`, нужно добавить `DataFrame` или матрицу. Подготовив необходимые данные, вы можете создать объект `basictabler`, вызвав

функцию `qhtbl()`, которая принимает два аргумента: `DataFrame` или матрицу и список параметров форматирования. В этом подразделе мы рассмотрим функции `qhtbl()` и `BasicTable()`. Функция `qhtbl()` позволяет создать таблицу быстро, а функция `BasicTable()` делает возможным поэтапное создание таблицы. Для начала создадим простую таблицу с помощью функции `qhtbl()`.

```
# Загрузка библиотеки
library(basictabler)

# Создание датафрейма
data <- data.frame(
  name = c("John Doe", "Jane Doe"),
  age = c(30, 25),
  salary = c(100000, 50000))

# Создание простой таблицы
table_plain <- qhtbl(data, theme = "largeplain")
table_plain
```

Результат выполнения этого кода выглядит так (рис. 5.1).

Рис. 5.1. Использование пакета basictabler для создания простой таблицы

Теперь, когда мы знаем, как выглядит результат, поговорим о том, что происходит в коде. Сначала мы создаем небольшой простой набор данных `data` исключительно для того, чтобы показать процесс использования пакета `basictabler`. Создав этот набор, мы вызываем на нем функцию `ghtbl()`, задаем тему `largeplain` и присваиваем переменной `table_plain`. Этот код создает таблицу, но не отправляет ее в Excel. О том, как это делается, мы поговорим позже.

Теперь рассмотрим чуть более сложный пример:

```
# Создание объекта basictabler
table <- qhtbl(data,
  theme = "largeplain",
  tableStyle = list("border-color" = "maroon"),
  headingStyle = list(
    "color" = "cornsilk", "background-color" = "maroon",
    "font-style" = "italic", "border-color" = "maroon"
  ),
```

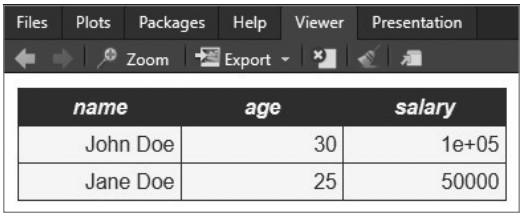
```
  cellStyle = list(
    "color" = "maroon", "background-color" = "cornsilk",
    "border-color" = "maroon"
  )
)

# Вывод таблицы в формате HTML
table
```

Этот фрагмент кода создает таблицу, содержащую те же данные, но предусматривающую определенную стилизацию. Теперь обсудим использованные параметры стилизации, а затем рассмотрим итоговый результат.

Первое различие заключается в применении параметра `tableStyle`, принимающего в качестве значения объект-список деклараций стилей CSS, которые будут применены к таблице. Для данного параметра мы указали значение `"border-color" = "maroon"`.

Следующий параметр, который мы использовали, `headingStyle`, тоже принимает объект-список деклараций стилей CSS, применяемых к заголовкам таблицы. В предыдущем фрагменте кода мы добавили в этот список четыре параметра: `color` со значением `cornsilk`, `background-color` со значением `maroon`, `font-style` со значением `italic` и, наконец, `border-color` со значением `maroon`. Посмотрим, как выглядит получившаяся в итоге таблица (рис. 5.2).



name	age	salary
John Doe	30	1e+05
Jane Doe	25	50000

Рис. 5.2. Таблица, созданная с помощью пакета basictabler с применением различных стилей

Теперь рассмотрим более длинный пример, в котором будут использованы различные возможности пакета `basictabler`, которые позволяют применять стили в соответствии с заданной нами логикой. Обратимся к следующему фрагменту кода:

```
# Более длинный пример
library(TidyDensity)
tn <- tidy_normal(.n = 10)
```

Строка `library(TidyDensity)` импортирует пакет `TidyDensity` в среду R. Этот пакет предоставляет инструменты для визуализации и обобщения распределений. Мы используем его для создания тиббла нормального распределения с десятью точками. Нам не нужно вызывать `basictabler`, так как к этому моменту он уже загружен.

Следующая строка `tn <- tidy_normal(.n = 10)` создает набор данных `tn`, генерируя десять случайных точек из стандартного нормального распределения. Кроме того, эта функция создает столбцы таблицы со значениями плотности вероятности (`dx, dy`), а также ее кумулятивного и обратного кумулятивного распределения (`pnorm` и `qnorm`).

```
tbl <- BasicTable$new()
# Форматирование значений (объясняется во вступлении)
columnFormats <- list(
  NULL, NULL, "%.4f", "%.4f", "%.4f", "%.4f", "%.4f"
)
```

Строка `tbl <- BasicTable$new()` создает новый экземпляр объекта `BasicTable` с именем `tbl`. `BasicTable` — это объект класса `R6` из пакета `basictabler`. С ним связано множество публичных методов, которые помогают создать саму таблицу, а также определить применяемые к ней стили.

```
tbl$addData(tn,
  firstColumnAsRowHeaders = TRUE,
  explicitColumnHeaders = c(
    "Simulation", "x", "y", "dx", "dy", "p", "q"
  ),
  columnFormats = columnFormats
)
```

В предыдущих фрагментах кода происходит следующее.

- Строка `tbl$addData(tn, ...)` добавляет данные из набора `tn` в объект `tbl`, а также задает некоторые параметры отображения данных, в частности, использует первый столбец в качестве заголовков строк и явно задает заголовки столбцов.
- Строка `columnFormats <- list(...)` создает список с именем `columnFormats`, который содержит форматы, заданные для столбцов со второго по седьмой (то есть имеющих индекс с 1 по 6) и представленные в виде `strings.tbl$renderTable()`. В данном случае мы использовали `%.4f`.
- Строка `tbl$renderTable()` создает таблицу на основе предоставленных ранее данных и параметров форматирования и отображает ее в среде R. Благодаря этому мы можем предварительно стилизовать таблицу и сравнить ее исходный и итоговый варианты.

Следующий фрагмент кода выглядит так:

```
# Добавление условного форматирования
cells <- tbl$getCells(
  rowNumbers = 2:11,
  columnNumbers = 3:7,
  matchMode = "combinations"
)
```

В данном случае строка `cells <- tbl$getCells(...)` извлекает подмножество ячеек из объекта таблицы `tbl`. Она выбирает ячейки из строк со 2-й по 11-ю и столбцов с 3-го по 7-й (`y, dx, dy, p, q`) с помощью метода `getCells()`.

```
tbl$mapStyling(
  cells = cells,
  styleProperty = "background-color",
  valueType = "color",
  mapType = "logic",
  mappings = list(
    "v<=-3", "red",
    "-3<v<=-2", "orange",
    "-2<v<=-1", "pink",
    "-1<v<= 0", "white",
    "0<v<=1", "white",
    "1<v<=2", "lightgreen",
    "2<v<=3", "lightblue",
    "3<v", "green"
  )
)
```

Строка `tbl$mapStyling(...)` применяет условное форматирование к выделенным ячейкам. Метод `mapStyling()` используется для сопоставления стилей (например, цвета фона) со значениями ячеек в соответствии с определенными условиями.

```
tbl$renderTable()
```

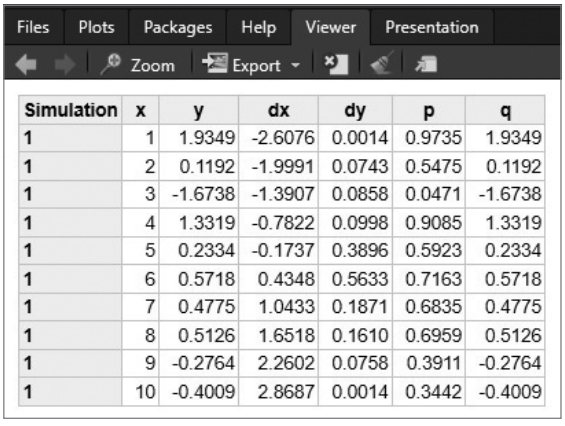
После применения условного форматирования строка `tbl$renderTable()` отображает обновленную таблицу с отформатированными ячейками в среде R.

Итак, приведенный выше код на языке R импортирует пакет, создает набор случайных чисел, генерирует таблицу с отформатированными данными, отображает ее, применяет условное форматирование к определенным ячейкам и, наконец, снова отображает таблицу с отформатированными ячейками. Функция условного форматирования заполняет ячейки разными цветами в зависимости от содержащихся в них значений.

Теперь посмотрим на созданную нами таблицу. Помните о том, что в вашем случае данные могут быть другими, так как значения сгенерированы случайным образом. Сначала рассмотрим простую таблицу (рис. 5.3), а затем стилизованную.

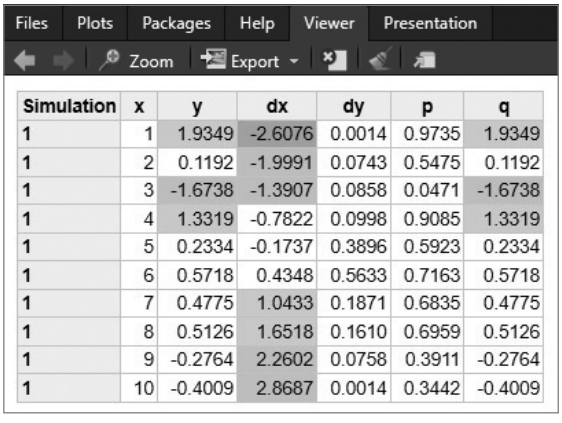
На рис. 5.4 показана таблица, стилизованная в соответствии с логикой, заданной в функции `mapStyling()`.

Теперь, когда мы создали таблицы, посмотрим, как их можно сохранить в файл Excel с помощью пакета `basictabler`. Сначала мы используем пакет `openxlsx`, с которым вы уже знакомы.



Simulation	x	y	dx	dy	p	q
1	1	1.9349	-2.6076	0.0014	0.9735	1.9349
1	2	0.1192	-1.9991	0.0743	0.5475	0.1192
1	3	-1.6738	-1.3907	0.0858	0.0471	-1.6738
1	4	1.3319	-0.7822	0.0998	0.9085	1.3319
1	5	0.2334	-0.1737	0.3896	0.5923	0.2334
1	6	0.5718	0.4348	0.5633	0.7163	0.5718
1	7	0.4775	1.0433	0.1871	0.6835	0.4775
1	8	0.5126	1.6518	0.1610	0.6959	0.5126
1	9	-0.2764	2.2602	0.0758	0.3911	-0.2764
1	10	-0.4009	2.8687	0.0014	0.3442	-0.4009

Рис. 5.3. Простая таблица BasicTable R6



Simulation	x	y	dx	dy	p	q
1	1	1.9349	-2.6076	0.0014	0.9735	1.9349
1	2	0.1192	-1.9991	0.0743	0.5475	0.1192
1	3	-1.6738	-1.3907	0.0858	0.0471	-1.6738
1	4	1.3319	-0.7822	0.0998	0.9085	1.3319
1	5	0.2334	-0.1737	0.3896	0.5923	0.2334
1	6	0.5718	0.4348	0.5633	0.7163	0.5718
1	7	0.4775	1.0433	0.1871	0.6835	0.4775
1	8	0.5126	1.6518	0.1610	0.6959	0.5126
1	9	-0.2764	2.2602	0.0758	0.3911	-0.2764
1	10	-0.4009	2.8687	0.0014	0.3442	-0.4009

Рис. 5.4. Стилизованная таблица BasicTable R6

Вот сценарий, который нам для этого понадобится; как видите, здесь используется каталог chapter5, поскольку он был создан первым:

```
# Запись стилизованной таблицы в Excel
library(openxlsx)

# Создание рабочей книги
wb <- createWorkbook()

# Добавление листа Data
addWorksheet(wb, "Data")
```

```
# Использование пакета basictabler для записи объекта tbl в Excel
tbl$writeToExcelWorksheet(
  wb = wb,
  wsName = "Data",
  topRowNumber = 1,
  leftMostColumnNumber = 1,
  applyStyles = TRUE
)

# Использование пакета openxlsx для сохранения файла
saveWorkbook(
  wb,
  file="chapter5/basictabler_excel.xlsx",
  overwrite = TRUE
)
```

Здесь мы использовали публичный метод `writeToExcelWorksheet()` из пакета `basictabler`. Он не записывает данные напрямую в файл Excel, но позволяет получить объект в нужном формате, который можно записать в Excel с помощью такого пакета, как `openxlsx`.

Теперь, когда мы обсудили некоторые возможности форматирования в R, поговорим об аналогичных инструментах Python, позволяющих форматировать ячейки и таблицы Excel. Получить дополнительные возможности вы можете, используя такие пакеты, как `gt` и `gtextras`.

Расширенные возможности форматирования с помощью Python

В этом разделе мы поговорим о форматировании ячеек, условном форматировании и работе со сводными таблицами. Кроме того, рассмотрим ряд практических примеров. Изучив материал этого раздела, вы сможете представлять ваши данные в таком виде, который будет способствовать их глубокому пониманию и анализу.

Форматирование ячеек

Форматирование ячеек позволяет эффективно представлять данные в Excel. Библиотеки `pandas` и `openpyxl` предлагают мощные инструменты для настройки внешнего вида ячеек. Вы можете применять широкий спектр стилей форматирования, чтобы сделать свои таблицы визуально приятными и повысить удобство восприятия данных.

Начнем с обсуждения процесса настройки различных свойств шрифта, таких как размер, цвет, жирность и начертание, позволяющих выделять определенные точки данных и создавать последовательную визуальную иерархию в таблицах.

Управление цветом фона ячеек позволяет сгруппировать связанные данные или выделить конкретные значения. Задав цвет фона ячеек, вы можете определить четкие границы между разделами таблицы, облегчив тем самым интерпретацию данных.

Выравнивание текста в ячейках — еще один важный метод форматирования. С помощью пакетов `pandas` и `openpyxl` вы можете выравнивать текст по горизонтали и вертикали, представляя данные в упорядоченном виде.

Настройка свойств шрифта

В первую очередь настроим свойства шрифта в ячейках с помощью библиотек `pandas` и `openpyxl`.

Для более сложной стилизации, в том числе для применения пользовательских CSS-подобных стилей, мы можем использовать метод `Styler.apply` из пакета `pandas`, а также пользовательские функции для форматирования ячеек, позволяющие настроить свойства шрифта в соответствии с нашими предпочтениями, как показано ниже:

```
# Импорт библиотеки pandas
import pandas as pd

data = {'Name': ['John', 'Alice', 'Michael'],
        'Age': [25, 30, 22],
        'City': ['New York', 'London', 'Paris']}

df = pd.DataFrame(data)

# Определение функции для применения свойств шрифта
def apply_font_properties(value):
  return 'font-weight: bold; font-size: 14px; font-style: italic; color: blue'

# Применение свойств шрифта
styled_df = df.style.applymap(apply_font_properties, subset='Name')

# Сохранение стилизованного объекта DataFrame в файле Excel
styled_df.to_excel('styled_table_pandas.xlsx', index=False)
```

Полученный лист Excel можно найти в репозитории GitHub, в папке, соответствующей этой главе.

При использовании пакета `openpyxl` настроить свойства шрифта можно с помощью класса `Font`. Например, чтобы сделать текст жирным, нужно задать значение `True` для атрибута `bold` объекта `font`. Чтобы добиться желаемого форматирования, вы можете настроить и другие свойства шрифта, такие как размер (`size`) и цвет (`color`), как показано в следующем примере:

```
# Пример настройки свойств шрифта с помощью openpyxl
from openpyxl import Workbook
from openpyxl.styles import Font
```

```
wb = Workbook()
ws = wb.active

# Применение свойств шрифта
font = Font(size=14, bold=True, italic=True, color='0000FF')
ws['A1'].font = font
ws['A1'] = 'Name'
ws['B1'] = 'Age'
ws['C1'] = 'City'
wb.save('styled_table_openpyxl.xlsx')
```

Цвет фона ячеек

Изменение цвета фона ячеек — еще один прием форматирования, который позволяет визуально выделить различные части таблицы. При использовании пакета `pandas` вы можете определить CSS-стиль `background-color` с помощью объекта `Styler`:

```
# Пример настройки цвета фона ячеек с помощью Pandas
import pandas as pd
data = {'Name': ['John', 'Alice', 'Michael'],
        'Age': [25, 30, 22],
        'City': ['New York', 'London', 'Paris']}

df = pd.DataFrame(data)

# Создание объекта Styler
styled_df = df.style

# Определение стиля ячеек
styled_df = styled_df.applymap( \
    lambda _: 'background-color: yellow', \
    subset=pd.IndexSlice[0, ['Name', 'Age']])

# Сохранение стилизованного объекта DataFrame в файле Excel
styled_df.to_excel('colored_table_pandas.xlsx', index=False)
```

Приведенный выше код показывает, как с помощью `pandas` можно создать объект `DataFrame`, содержащий некоторые данные, а затем задать цвет фона для определенных его ячеек. Этот `DataFrame` содержит имена, возраст и города проживания людей. Используя объект `Styler` библиотеки `pandas`, мы можем задать цвет фона для определенных ячеек. В данном примере мы выделяем желтым цветом ячейки `Name` и `Age` в первой строке. Наконец, стилизованный объект `DataFrame` сохраняется в файле Excel `colored_table_pandas.xlsx`. Описанный прием позволяет легко и гибко форматировать ячейки при экспорте данных из среды Python в Excel.

Теперь посмотрим, как можно добиться той же цели, используя пакет `openpyxl`. Цвет фона ячеек можно задать с помощью класса `Fill`.

```
# Пример задания цвета фона ячеек с помощью openpyxl
from openpyxl import Workbook
from openpyxl.styles import PatternFill

wb = Workbook()
ws = wb.active

# Применение фоновых цветов к ячейкам
yellow_fill = PatternFill(start_color='FFFF00', end_color='FFFF00',
fill_type='solid')
ws['A1'].fill = yellow_fill
ws['A1'] = 'Name'
ws['B1'] = 'Age'
ws['C1'] = 'City'
wb.save('colored_table_openpyxl.xlsx')
```

Выравнивание текста в ячейках

Выравнивание текста в ячейках может существенно улучшить визуальное восприятие таблицы. В pandas для выравнивания текста можно использовать объект `Styler`.

```
# Пример выравнивания текста в ячейках с помощью pandas
import pandas as pd
data = {'Name': ['John', 'Alice', 'Michael'],
        'Age': [25, 30, 22],
        'City': ['New York', 'London', 'Paris']}
df = pd.DataFrame(data)

# Применение стиля выравнивания текста
alignment_styles = {'text-align': 'center'}
styled_df = df.style.set_properties( \
    subset=['Name', 'Age', 'City'], **alignment_styles)
styled_df.to_excel('aligned_table_pandas.xlsx', index=False)
```

Этот код выравнивает текст в указанных столбцах `DataFrame` по центру. Стиль выравнивания применяется к указанным столбцам с помощью метода `set_properties`. Полученный объект `DataFrame` сохраняется в файле Excel `aligned_table_pandas.xlsx`.

При использовании пакета `openpyxl` выравнивание текста можно реализовать с помощью класса `Alignment`.

```
# Пример выравнивания текста в ячейках с помощью openpyxl
from openpyxl import Workbook
from openpyxl.styles import Alignment
wb = Workbook()
ws = wb.active

# Применение стиля выравнивания текста
alignment = Alignment(horizontal='center', vertical='center')
```

```
ws['A1'].alignment = alignment
ws['A1'] = 'Name'
ws['B1'] = 'Age'
ws['C1'] = 'City'
wb.save('aligned_table_openpyxl.xlsx')
```

Таким образом, изучив приведенные в этом подразделе примеры, вы узнали о том, как форматировать ячейки, задавать свойства шрифта, изменять цвет фона ячеек и выравнивать содержащийся в них текст с помощью пакетов `pandas` и `openpyxl`. Используя эти приемы форматирования, вы сможете эффективно представлять данные в Excel, помещая их в эстетично оформленные и информативные таблицы.

Далее мы поговорим об условном форматировании.

Условное форматирование

Условное форматирование — мощная функция Excel, которая позволяет автоматически применять стили форматирования к ячейкам на основе определенных критериев, визуально выделять важные данные, выявлять тенденции и делать листы Excel более интерактивными. В этом подразделе мы рассмотрим, как добавить условное форматирование с помощью пакета `openpyxl`. Вы научитесь выделять ячейки на основе таких критериев, как диапазоны значений, текст и дата, а также создавать пользовательские правила условного форматирования, отвечающие вашим потребностям.

Изучив материал этого подраздела, вы обретете навыки добавления динамичного и эстетичного условного форматирования в листы Excel непосредственно из среды Python.

Визуализация данных с помощью условного форматирования

Применяя условное форматирование в листах Excel, вы можете эффективно визуализировать данные и мгновенно извлекать из них ценную информацию. Например, можно выделить наибольшие и наименьшие значения в столбце, окрасить ячейки в разные цвета в соответствии с определенными критериями или обозначить значительные изменения, произошедшие с течением времени.

Условное форматирование особенно полезно при работе с большими наборами данных, поскольку позволяет быстро выявлять ключевую информацию и принимать взвешенные решения.

Пакет `openpyxl` предоставляет функциональные возможности для реализации условного форматирования в листах Excel, позволяя применять различные стили на основе определенных критериев. Процесс применения `openpyxl` для условного форматирования состоит из следующих этапов.

1. Импортируйте необходимые модули из пакета `openpyxl` и загрузите свои данные в объект рабочей книги.

2. Создайте правило условного форматирования с помощью `openpyxl.formatting.rule`.
3. Определите условия применения этого правила, например, используя такие критерии, как значения ячеек, содержащийся в них текст или дата.
4. Примените правило к нужному диапазону ячеек с помощью функции `openpyxl.worksheet.conditional.ConditionalFormatting.add()`.

Используя пакет `openpyxl`, вы можете легко применять правила условного форматирования и добавлять визуальные подсказки в листы Excel, улучшая представление данных и облегчая их анализ.

Рассмотрим пример кода, который реализует описанные выше концепции:

```
import pandas as pd
import openpyxl
from openpyxl.formatting.rule import ColorScaleRule, CellIsRule

# Создание образцов данных
data = {'Name': ['John', 'Alice', 'Michael', 'Emily'],
        'Age': [25, 30, 22, 28],
        'City': ['New York', 'London', 'Paris', 'Sydney'],
        'Sales': [1000, 800, 1200, 900]}
df = pd.DataFrame(data)

# Запись объекта DataFrame в рабочий лист
df.to_excel("conditional_formatting.xlsx", index=False)

# Загрузка рабочей книги
wb = openpyxl.load_workbook('conditional_formatting.xlsx')
ws = wb.active

# Определение правила для изменения цвета текста на красный
red_text_rule = CellIsRule( \
    operator="lessThan", formula=["1000"], stopIfTrue=True, \
    font=openpyxl.styles.Font(color="FF0000"))
ws.conditional_formatting.add(f"D2:D{len(df)+1}", red_text_rule)

# Определение правила для создания зеленой цветовой шкалы
min_sales = min(df['Age'])
max_sales = max(df['Age'])

green_fill_rule = ColorScaleRule( \
    start_type='num', start_value=min_sales, start_color='0000FF00', \
    end_type='num', end_value=max_sales, end_color='00FFFF00')

ws.conditional_formatting.add(f"B2:B{len(df)+1}", green_fill_rule)

# Сохранение рабочей книги Excel
wb.save('conditional_formatting.xlsx')
```

В этом коде мы создаем рабочую книгу Excel из объекта `DataFrame` пакета `pandas`. Затем мы определяем два правила условного форматирования. Одно изменяет цвет текста на красный, если значение `Sales` меньше `1000`. А второе (`green_fill_rule`) создает шкалу цветов, заполняя ячейки в столбце `Age` оттенками зеленого цвета, исходя из их относительных значений в пределах диапазона от минимального до максимального. Эти правила добавляются в свойство рабочего листа `conditional_formatting`. Наконец, мы сохраняем рабочую книгу Excel. Заданные правила условного форматирования будут применяться при открытии файла в программе Microsoft Excel.

Благодаря пользовательским правилам условного форматирования вы получаете точный контроль над отображением данных в листах Excel и можете легко выявлять закономерности, тенденции и выбросы (точки данных, значительно отличающиеся от нормы).

Практический пример: использование динамической тепловой карты с условным форматированием

Чтобы продемонстрировать дополнительные возможности условного форматирования, рассмотрим пример создания динамической тепловой карты с помощью Python и Excel. Мы используем функцию условного форматирования для окрашивания ячеек в зависимости от величины содержащихся в них значений, что позволит нам эффективно визуализировать относительную плотность распределения данных и присутствующие в них закономерности. То, что создается в данном примере, также называется *таблицей с подсветкой*, поскольку, помимо разноцветных ячеек, она содержит аннотацию в виде чисел.

Реализация тепловой карты выглядит так:

```
import pandas as pd
import openpyxl
from openpyxl.utils.dataframe import dataframe_to_rows
from openpyxl.formatting.rule import ColorScaleRule

# Образцы данных для тепловой карты
data = {
    'Category': ['A', 'B', 'C', 'D'],
    'Jan': [10, 20, 30, 40],
    'Feb': [15, 25, 35, 45],
    'Mar': [12, 22, 32, 42],
    'Apr': [18, 28, 38, 48]
}

# Преобразование данных в объект pandas DataFrame
df = pd.DataFrame(data)

# Запись объекта DataFrame в рабочий лист
df.to_excel("heatmap_with_conditional_formatting.xlsx", index=False)
```

```
# Загрузка рабочей книги
wb = openpyxl.load_workbook('heatmap_with_conditional_formatting.xlsx')
ws = wb.active

# Определение диапазона для условного форматирования (исключая столбец 'Category')
data_range = f'B2:E{len(df) + 1}' # Настройка диапазона в зависимости
                                  # от размера объекта DataFrame

# Применение цветового градиента к диапазону
color_scale_rule = ColorScaleRule(start_type='min', \
    start_color='FFFFFF', end_type='max', end_color='FF0000')
ws.conditional_formatting.add(data_range, color_scale_rule)

# Сохранение рабочей книги
wb.save('heatmap_with_conditional_formatting.xlsx')
```

Условное форматирование — весьма ценный инструмент, позволяющий улучшить визуальное представление данных в Excel. Используя пакет `pandas` или `openpyxl`, вы можете легко реализовать правила условного форматирования, которые позволяют динамически применять стили к ячейкам на основе заданных критериев. Внедрив условное форматирование в свои листы Excel, вы сможете создавать визуализации и будете лучше понимать свои данные, благодаря чему ваши процессы их анализа и принятия решений будут более эффективными.

Сводные таблицы

Сводные таблицы (pivot tables) — мощный инструмент Excel, позволяющий быстро обобщать и анализировать большие наборы данных. Благодаря таким таблицам вы можете агрегировать данные, группировать их и выполнять вычисления, получая ценные сведения с помощью всего нескольких щелчков кнопкой мыши. В этом подразделе мы поговорим о том, как создавать сводные таблицы и управлять ими, используя библиотеку Python `pywin32`, а также обсудим некоторые приемы настройки промежуточных и общих итогов, подписей и стилей.

Создание сводных таблиц с помощью pywin32

Библиотека `pywin32` позволяет взаимодействовать с Microsoft Excel через COM-интерфейс. С ее помощью вы можете управлять функциями Excel, в том числе создавать сводные таблицы.

Чтобы создать сводную таблицу с помощью `win32com.client` в среде Python, вы можете воспользоваться фрагментами кода, приведенными ниже.

Сначала необходимо импортировать нужный модуль, запустить экземпляр Excel, как мы это делали в главе 3, и получить лист, с которым мы собираемся работать.

```
# Импорт необходимых модулей из пакета win32com.client
import win32com.client as win32
import os.path
```

```
# Создание нового экземпляра Excel и обеспечение его видимости
excel = win32.Dispatch('Excel.Application')
excel.Visible = True

# Создание новой рабочей книги или открытие существующей
workbook = excel.Workbooks.Add() # Создание новой рабочей книги
# Или открытие существующей рабочей книги:
# workbook = excel.Workbooks.Open('путь/к/файлу/workbook.xlsx')

# Получение ссылки на лист, на котором будет создана сводная таблица:
sheet = workbook.ActiveSheet   # Получение активного листа
# Или указание имени нужного листа:
# sheet = workbook.Sheets('Sheet1')
```

Далее нужно сгенерировать несколько образцов данных и записать их на лист (это действие не является обязательным; выполняйте его только в том случае, если у вас есть данные для анализа).

```
# Образцы данных
data = [
    ['Product', 'Category', 'Sales'],
    ['Product A', 'Category 1', 100],
    ['Product B', 'Category 2', 200],
    ['Product C', 'Category 1', 150],
    ['Product D', 'Category 2', 50],
    # Сюда можно добавить дополнительные строки данных...
]

# Запись данных на лист
for row_index, row in enumerate(data, start=1):
    for col_index, value in enumerate(row, start=1):
        sheet.Cells(row_index, col_index).Value = value
```

ЗАБАВНЫЙ ФАКТ

При выполнении этого вложенного цикла for вы можете увидеть, как рабочая книга заполняется значениями!

Теперь мы можем приступить к созданию сводной таблицы! Для начала создадим новый лист, на котором она будет располагаться:

```
# Добавление в рабочую книгу рабочего листа, где будет находиться сводная таблица
pivot_table_sheet = workbook.Worksheets.Add()
pivot_table_sheet.Name = 'Pivot Table'
```

Далее мы можем создать саму сводную таблицу, используя метод `Create()` свойства рабочей книги `PivotCaches` и вызов функции `CreatePivotTable()`:

```
# Создание кэша Pivot Cache на основе диапазона данных
# (UsedRange выделяет весь используемый диапазон на листе)
pivot_cache = workbook.PivotCaches().Create(SourceType=1, \
    SourceData=sheet.UsedRange)

# Создание сводной таблицы на новом листе с помощью кэша Pivot Cache
pivot_table = pivot_cache.CreatePivotTable( \
    TableDestination=pivot_table_sheet.Cells(3, 1), \
    TableName='MyPivotTable')
```

Определив сводную таблицу, мы можем добавить поля, которые собираемся использовать в качестве строк, столбцов и полей данных:

```
# Добавление полей в сводную таблицу с указанием их ориентации
# (строки, столбцы, данные и т. д.):
pivot_table.PivotFields('Product').Orientation = 1   # поле строки
pivot_table.PivotFields('Category').Orientation = 2  # поле столбца
pivot_table.PivotFields('Sales').Orientation = 4     # поле данных
```

Мы почти у цели! Теперь у нас есть рабочая сводная таблица, однако нам может понадобиться возможность включать и выключать отображение общих и промежуточных итогов.

```
# Управление отображением общих итогов по строкам и столбцам
pivot_table.ColumnGrand = False
pivot_table.RowGrand = False

# Настройка полей, предусматривающих промежуточные итоги
pivot_table.PivotFields('Sales').Subtotals = [False]*12
pivot_table.PivotFields('Product').Subtotals = [False]*12
pivot_table.PivotFields('Category').Subtotals = [True]*12
```

Наконец, мы можем настроить подписи и стили:

```
# Настройка подписей и стилей
pivot_table.ShowTableStyleRowStripes = True
pivot_table.PivotFields('Product').Caption = 'Product Name'
pivot_table.PivotFields('Sales').NumberFormat = '#,##0'
pivot_table.PivotFields('Sales').Caption = 'Total Sales'
# Примечание: подписи изменяют не саму таблицу, а мастер создания сводной
# таблицы, который открывается при щелчке на соответствующем пункте меню

# Сохранение рабочей книги и закрытие приложения Excel:
# Примечание: замените путь к файлу актуальным для вашего компьютера.
file_path = os.path.join('C:' + os.sep, 'Users', 'david_1q5aypk', \
    'Extending-Excel-with-Python-and-R')
workbook.SaveAs(os.path.join(file_path, 'pivot_table.xlsx'))

workbook.Close()
excel.Quit()
```

Итак, теперь вы знаете, как создать сводную таблицу, помогающую эффективно анализировать и обобщать данные, с помощью `win32com.client`. Используя эту библиотеку, вы сможете получить полный контроль над функциями Excel, в том числе у вас будет возможность создавать и настраивать сводные таблицы, отвечающие вашим нуждам.

Резюме

Эта глава была посвящена форматированию листов Excel, которое позволяет представлять данные в эстетичном и организованном виде. Мы показали вам основные приемы преобразования исходных данных в профессионально выглядящие и удобные для восприятия таблицы.

В разделе, посвященном форматированию ячеек, мы показали, как применять к ним различные стили, например настраивать свойства шрифта, цвет фона и выравнивание текста. Освоение этих методов позволит вам создавать упорядоченные таблицы, имеющие эстетичный вид.

Затем вы познакомились с мощной функцией условного форматирования, благодаря которой можно динамически форматировать ячейки на основе определенных критериев. Мы рассмотрели практические примеры использования R-пакетов `styledTables` и `basictabler`, а также библиотек Python `pandas` и `openpyxl`, предназначенных для реализации различных правил условного форматирования, таких как цветовые шкалы, которые позволяют выделить важнейшие данные и извлечь из них ценную информацию.

И наконец, мы обсудили потенциал сводных таблиц — незаменимого инструмента для обобщения и анализа данных. Вы создали несколько сводных таблиц с помощью библиотеки `pywin32` и научились настраивать промежуточные и общие итоги, а также подписи и стили.

Таким образом, изучив материал этой главы, вы узнали, как использовать Excel в сочетании с пакетами `styledTables`, `basictabler`, `pandas`, `openpyxl` и `pywin32`, позволяющими представлять данные в виде профессионально оформленных и визуально приятных таблиц, а также извлекать из них значимую информацию, которая помогает принимать более обоснованные решения. Освоив эти методы, вы сможете улучшить свое мастерство анализа и визуализации данных и будете применять функции Excel более эффективно.

В следующей главе мы поговорим о том, как добавлять красивые визуализации данных в листы Excel с помощью R и Python.

6 Вставка графиков ggplot2/matplotlib

Визуализация данных — незаменимый инструмент анализа и представления сложной информации в доступной и интуитивно понятной форме. Она играет важную роль в различных областях, начиная от бизнес-аналитики и программирования и заканчивая научными исследованиями и повседневным принятием решений.

В этой главе мы рассмотрим следующие темы:

- области применения визуализации данных;
- визуализация данных с помощью пакетов `ggplot2` и `cowplot`;
- столбчатые и гантельные диаграммы;
- знакомство с библиотеками для визуализации данных;
- создание графиков с помощью библиотеки Python `plotnine`;
- создание графиков с помощью `matplotlib`;
- встраивание визуализаций в отчеты Excel.

Технические требования

Для изучения этой главы вам необходимо установить следующие R-пакеты:

- `ggplot2 3.4.4`;
- `cowplot 1.1.3`.

Файлы с кодом из примеров этой главы доступны по адресу https://github.com/PacktPublishing/Extending-Excel-with-Python-and-R/tree/main/chapter6.

Области применения визуализации данных

Прежде чем переходить к обсуждению методов визуализации данных, перечислим основные области, в которых она может быть полезна.

- **Эффективная коммуникация.** Визуализация данных — мощный инструмент коммуникации, позволяющий преодолеть языковые барьеры и упростить

понимание сложных концепций. Представление данных в виде диаграмм, графиков и интерактивных дашбордов помогает более убедительно излагать информацию на совещаниях, в презентациях и отчетах. Визуальное представление данных облегчает их усвоение и понимание. Например, диаграмма Санки помогает наглядно показать поток данных от начальной до конечной точки.

- **Принятие обоснованных решений.** Визуализации позволяют лицам, принимающим решения, основывать свои суждения на фактических данных. Представление информации в наглядной форме упрощает выявление потенциальных возможностей, рисков и областей, требующих улучшения, что приводит к принятию более эффективных решений.
- **Выявление тенденций и аномалий.** Инструменты визуализации позволяют техническим и нетехническим специалистам быстро выявлять тенденции, изменения и аномалии в данных. Эта возможность особенно ценна в таких областях, как финансы, где своевременное обнаружение аномалий позволяет предотвратить значительные финансовые потери.
- **Разведочный анализ данных** (exploratory data analysis, EDA). Визуализация данных — неотъемлемый этап их анализа. Создавая гистограммы, тепловые карты, диаграммы рассеяния и размаха, программисты могут изучить распределение данных и существующие в них взаимосвязи, прежде чем приступать к более глубокому анализу.
- **Мониторинг в режиме реального времени.** В приложениях, работающих с постоянно изменяющимися данными, их визуализация в режиме реального времени позволяет отслеживать ключевые показатели и оперативно реагировать на возникающие ситуации.
- **Геопространственный анализ.** Визуализации географических данных, например обычные и тепловые карты, крайне важны для анализа такой основанной на местоположении информации, как распределение клиентов, распространение заболеваний и изменения, происходящие в окружающей среде.
- **Прогнозирование и предсказательная аналитика.** Визуализация данных позволяет представить результаты применения прогностических моделей и выявленные тенденции, облегчая понимание возможных сценариев будущего и принятие упреждающих мер.
- **Интерактивная отчетность.** Благодаря интерактивным визуализациям конечные пользователи могут настраивать представления данных и взаимодействовать с ними, получая более персонализированный опыт и проводя более глубокое исследование имеющейся информации.

К популярным способам визуализации данных относятся:

- линейные диаграммы для анализа временных рядов;
- столбчатые и круговые диаграммы для сравнения категориальных данных;
- диаграммы рассеяния для выполнения корреляционного анализа;

- пузырьковые диаграммы для сравнения многомерных данных;
- хороплеты (фоновые картограммы) для представления геопространственных данных.

Теперь перейдем к практике.

Визуализация данных с помощью пакета ggplot2

`ggplot2` — мощный и широко используемый R-пакет для визуализации данных, разработанный Хэдли Викхэмом и являющийся частью экосистемы `tidyverse`. С его помощью пользователи могут создавать высококачественную и настраиваемую графику, используя декларативный подход, при котором диаграммы создаются путем указания источника данных, сопоставления визуальных параметров (эстетических характеристик) с переменными и добавления слоев геометрических фигур, статистик и тем. Реализованная в `ggplot2` грамматика графики позволяет легко создавать сложные визуализации, что делает данный пакет популярным инструментом для разведочного анализа данных и представления его результатов.

В этом разделе мы создадим несколько графиков с помощью `ggplot2` и встроенного в R набора данных `iris`. В первую очередь нужно установить и загрузить этот пакет в текущую среду:

```
install.packages("ggplot2")
library(ggplot2)
```

Теперь, когда библиотека установлена и загружена, можем создать простейшую R-версию графика, применив функцию `hist()`, а затем перебрав в цикле значения в столбце `Species`.

Разберем остальную часть сценария:

```
hist(iris$Sepal.Width)

par(mfrow = c(2,2))
for (species in unique(iris$Species)) {
    hist(iris$Sepal.Width[iris$Species == species], main = species,
           xlab = species)
}
hist(iris$Sepal.Width, main = "All Species")
par(mfrow = c(1,1))
```

Первая строка кода, `hist(iris$Sepal.Width)`, создает гистограмму распределения значений ширины чашелистика для всех видов ирисов. Набор данных `iris` встроен в R, поэтому нам не нужно загружать его с помощью функции `data()`. Столбец `Sepal.Width` этого набора данных содержит значения ширины чашелистика каждого из цветков. Функция `hist()` создает гистограмму на основе данных, содержащихся в указанном столбце.

Вторая строка кода, `par(mfrow = c(2,2))`, разделяет область создания графика на четыре квадранта, что позволяет нам создать четыре гистограммы, расположив их рядом друг с другом.

Третья строка кода, `for(species in unique(iris$Species)) {`, запускает цикл `for`, который перебирает уникальные значения в столбце `Species` набора данных `iris`. Функция `unique()` возвращает вектор, содержащий все уникальные значения в столбце.

Тело цикла `for`, `hist(iris$Sepal.Width[iris$Species == species], main = species, xlab = species)`, создает гистограмму распределения значений ширины чашелистика для текущего вида ирисов. Выражение `iris$Sepal.Width[iris$Species == species]` выбирает значения ширины чашелистика для текущего вида. Аргумент `main` задает название гистограммы, а аргумент `xlab` — подпись для оси *X*.

Следующая строка кода, `hist(iris$Sepal.Width, main = "All Species")`, создает гистограмму распределения значений ширины чашелистика для всех видов в последнем квадранте области создания.

Последняя строка кода, `par(mfrow = c(1,1))`, возвращает область создания диаграммы в исходное состояние.

Результат выполнения кода выглядит так (рис. 6.1).

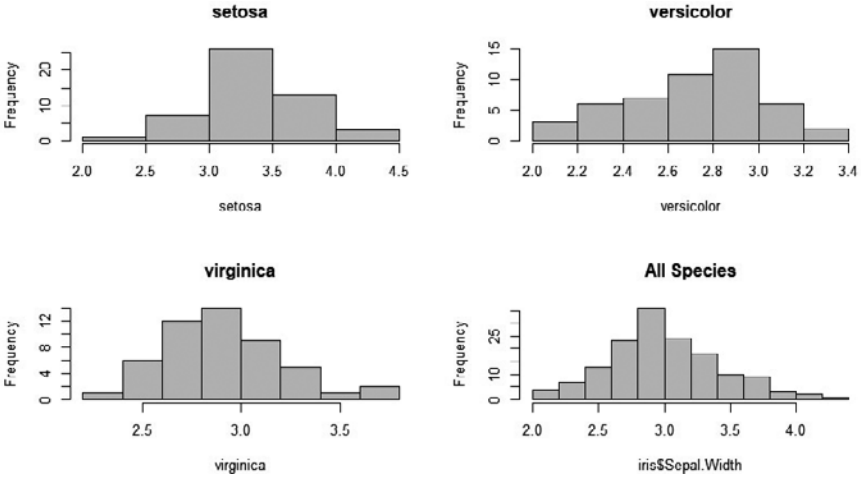


Рис. 6.1. Гистограммы, созданные с помощью базового функционала R

Теперь, когда у нас есть простейшая R-версия гистограмм, посмотрим, как они будут выглядеть при использовании ggplot2:

```
# Создание гистограммы распределения значений ширины
# чашелистика для всех видов ирисов
iris |>
ggplot(aes(x = Sepal.Width)) +
  geom_histogram(alpha = 0.328) +
  theme_minimal()

# Создание гистограммы распределения значений ширины
# чашелистика для каждого вида ирисов
iris |>
ggplot(aes(x = Sepal.Width, fill = Species)) +
  geom_histogram(alpha = 0.328) +
  theme_minimal()
```

Код `iris |> ggplot(aes(x = Sepal.Width)) + geom_histogram(alpha = 0.328) + theme_minimal()` создает гистограмму распределения значений ширины чашелистика для всех видов ирисов. Набор данных `iris` встроен в R, поэтому нам не нужно загружать его с помощью функции `data()`. R-пакет `ggplot()` — мощный инструмент для создания графиков, позволяющий создавать настраиваемые визуализации. Функция `aes()` сопоставляет переменные с визуальными параметрами графика. В данном случае мы сопоставляем значения ширины чашелистика с осью *X*. Функция `geom_histogram()` создает гистограмму на основе данных в указанном столбце. Аргумент `alpha` задает степень прозрачности столбцов гистограммы. Функция `theme_minimal()` применяет к графику минималистичную тему.

Код `iris |> ggplot(aes(x = Sepal.Width, fill = Species)) + geom_histogram(alpha = 0.328) + theme_minimal()` создает гистограмму распределения значений ширины чашелистика для каждого вида. Аргумент `fill` функции `aes()` заполняет цветом столбцы гистограммы. В данном случае мы сопоставляем этот визуальный параметр со столбцом `Species`, чтобы увидеть распределение значений ширины чашелистика для каждого вида ирисов.

Теперь, когда мы разобрались с кодом, посмотрим на результат его выполнения.

В данном случае мы получим два графика (рис. 6.2 и 6.3), поскольку при использовании пакета `ggplot2` объекты создаются не так, как при использовании функции `par()` языка R.

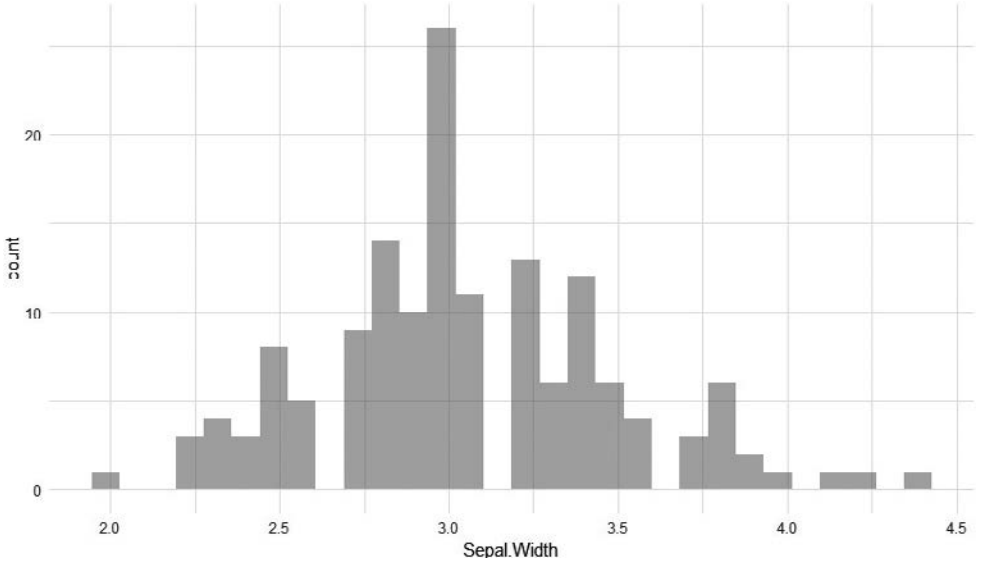


Рис. 6.2. Гистограмма распределения значений ширины чашелистика для всех видов ирисов, созданная с помощью ggplot2

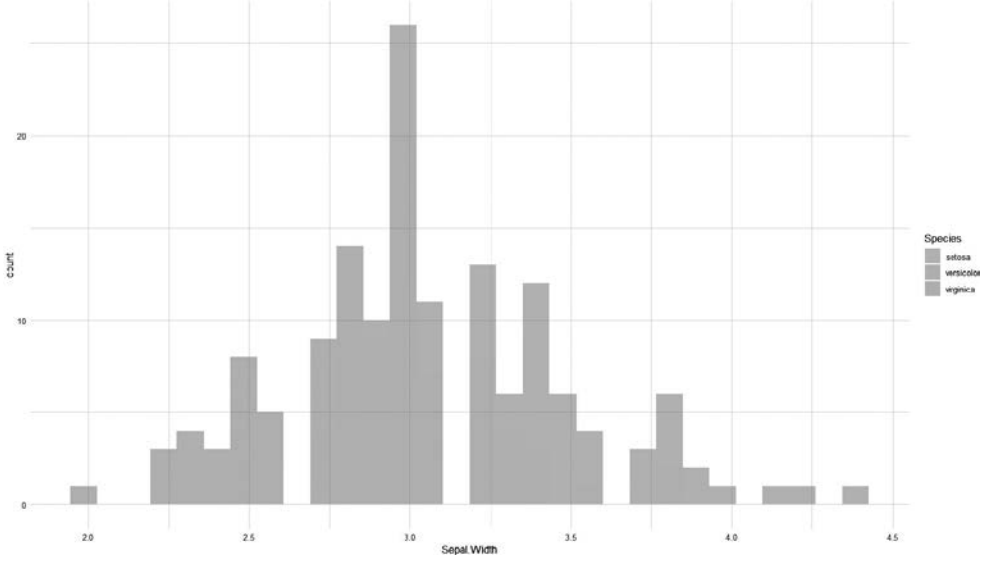


Рис. 6.3. Гистограмма распределения значений ширины чашелистика для каждого вида ирисов, созданная с помощью ggplot2

Чтобы отобразить гистограммы рядом друг с другом, можно использовать функцию ggplot2 `facet_wrap()`.

```
iris |>
  ggplot(aes(x = Sepal.Width, fill = Species)) +
  geom_histogram(alpha = 0.328) +
  facet_wrap(~ Species, scales = "free") +
  theme_minimal()
```

Функция `aes()` сопоставляет данные с визуальными параметрами графика. В данном случае мы сопоставляем значения ширины чашелистика с осью *X*, а виды ирисов (`Species`) — с цветом заливки столбцов гистограммы. Функция `geom_histogram()` создает гистограмму на основе данных, содержащихся в указанных столбцах. Аргумент `alpha` задает степень их прозрачности. Функция `facet_wrap()` создает отдельный график для каждого из значений в столбце `Species`. Аргумент `scales = "free"` указывает функции `facet_wrap()` на то, что масштабы осей *X* и *Y* могут быть разными для каждого графика. Функция `theme_minimal()` применяет к графику минималистичную тему.

Результат выполнения этого кода — серия из трех гистограмм, которые отражают распределение значений ширины чашелистиков для каждого вида ирисов (рис. 6.4). Разные цвета столбцов позволяют легко увидеть разницу в этом распределении.

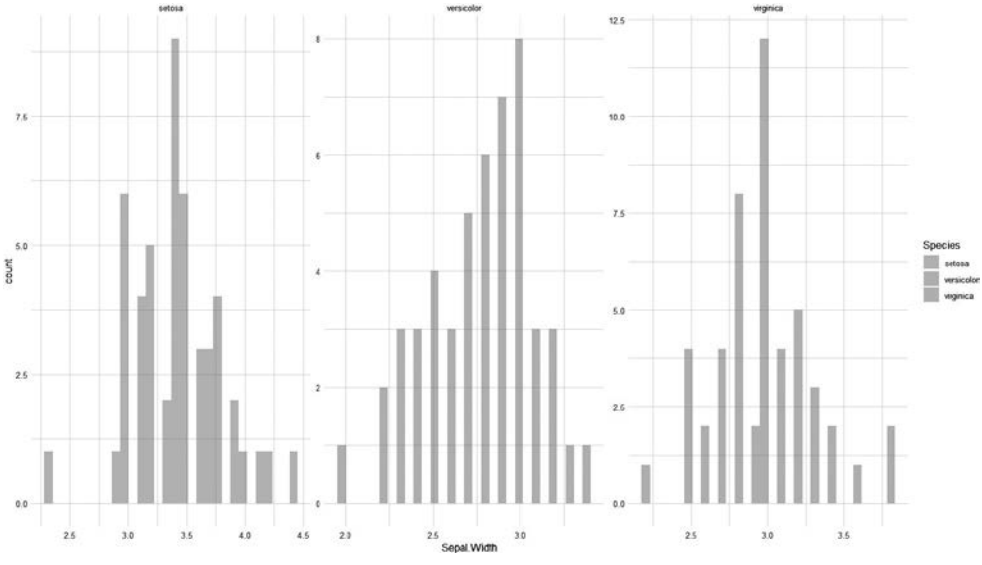


Рис. 6.4. Гистограммы распределения значений ширины чашелистика, созданные с помощью функции ggplot2 facet_wrap

Теперь, когда мы разобрались с ggplot2, посмотрим, как можно создать более сложные и высококачественные графики с помощью пакета cowplot.

Визуализация данных с помощью пакета cowplot

R-пакет cowplot предоставляет множество функций для создания высококачественных, готовых к публикации графиков. Его можно использовать для следующих целей.

- **Расположение графиков в виде сетки.** Для организации нескольких графиков в виде сетки с подписями и аннотациями используется функция plot_grid().
- **Выравнивание графиков.** С помощью функции align_plots() можно выровнять оси и другие элементы нескольких графиков, чтобы их представление было единообразным.
- **Комбинирование результатов применения разных графических фреймворков.** Функция plot_grid() позволяет комбинировать графики, созданные с помощью разных фреймворков, например пакета ggplot2 и базовых графических функций языка R.
- **Добавление аннотаций к графикам.** Функции ggdraw() и draw_*() можно использовать для добавления к графикам таких аннотаций, как текст, изображения и геометрические фигуры.
- **Создание готовых к публикации изображений.** Пакет cowplot содержит несколько тем, предназначенных для создания высококачественных изображений.

Теперь, когда мы выяснили, что представляет собой пакет cowplot, рассмотрим первый пример:

```
# Установка библиотек
install.packages("ggplot2")
install.packages("cowplot")

# Загрузка необходимых библиотек
library(ggplot2)
library(cowplot)

# Загрузка набора данных Iris
data(iris)
```

В этом фрагменте кода мы устанавливаем библиотеки ggplot2 и cowplot. Данный код сработает, даже если они уже установлены, но мы хотим установить последние версии пакетов.

Теперь создадим график для каждого вида ирисов с помощью цикла `for`:

```
# Создание отдельных гистограмм для каждого вида ирисов
histograms <- list()
for (species in unique(iris$Species)) {
  data_subset <- iris[iris$Species == species, ]

  histogram <- ggplot(data_subset, aes(x = Sepal.Width)) +
     geom_histogram(binwidth = 0.1, fill = "lightblue", color = "black") +
     labs(title = paste("Sepal Width Histogram for", species)) +
     labs(x = "", y = "") +
     theme_minimal()

  histograms[[species]] <- histogram
}
```

Разберем этот код более подробно.

1. Строка `histograms <- list()` создает пустой список `histograms`, в котором мы будем хранить гистограммы для каждого вида ирисов.
2. Описанный в следующей строке цикл, `for (species in unique(iris$Species)) { ... }`, перебирает все уникальные виды в наборе данных `iris` и выполняет над ними следующие операции. Строка `data_subset <- iris[iris$Species == species, ]` создает подмножество данных, добавляя в него только те строки, которые содержат вид, совпадающий с текущим видом, обрабатываемым в цикле в данный момент. Это помогает нам последовательно сосредоточиваться на данных, относящихся к одному виду.
3. В строке `histogram <- ggplot(data_subset, aes(x = Sepal.Width)) + ...` происходит нечто интересное. Мы используем библиотеку `ggplot2` для создания гистограммы, что можно уподобить рисованию графика, показывающего количество цветков, имеющих определенную ширину чашелистика. Параметр `aes(x = Sepal.Width)` указывает на то, что мы хотим отложить значения `Sepal.Width` вдоль оси *X*.
4. Строка `geom_histogram(binwidth = 0.1, fill = "lightblue", color = "black")` создает столбцы гистограммы, указывая на то, как они должны выглядеть, в частности, задавая их ширину и цвет.
5. Строка `labs(title = paste("Sepal Width Histogram for", species))` добавляет заголовок гистограммы, сообщающий нам о том, какие данные она отображает. Заголовок гистограммы меняется в зависимости от текущего вида ирисов, обрабатываемого в цикле.
6. Строка `labs(x = "", y = "")` удаляет подписи осей *X* и *Y*, благодаря чему гистограмма выглядит менее загроможденной. Затем мы применяем тему к графику с помощью функции `theme_minimal()`, которая делает фон гистограммы простым и чистым.

7. Наконец, строка `histograms[[species]] <- histogram` сохраняет гистограмму, созданную для текущего вида ирисов, в списке `histograms`. Чуть позже мы используем название вида для получения доступа к этой гистограмме.

Этот код несколько отличается от приведенного выше, но дает те же результаты.

```
histograms <- lapply(unique(iris$Species), function(species) {
  data_subset <- iris[iris$Species == species, ]

  histogram <- ggplot(data_subset, aes(x = Sepal.Width)) +
     geom_histogram(binwidth = 0.1, fill = "lightblue", color = "black") +
     labs(title = paste("Sepal Width Histogram for", species)) +
     labs(x = "", y = "") +
     theme_minimal()

  return(histogram)
})
```

Если коротко, то этот код создает гистограммы распределения значений ширины чашелистиков для разных видов ирисов, содержащихся в наборе данных `iris`. Для этого используется цикл, который последовательно создает подмножество данных, относящихся к конкретному виду, а затем создает гистограмму с соответствующим форматированием и подписями. Все гистограммы сохраняются в списке `histograms`, что позволяет нам использовать их в дальнейшем. В этом списке каждый вид представляет собой объект гистограммы, имеющий соответствующее название. Теперь мы создадим гистограммы на основе полного набора данных, а затем объединим их, используя функцию `plot_grid()` из пакета `cowplot`. Для начала рассмотрим код:

```
# Создание гистограммы, охватывающей все виды
all_species_hist <- ggplot(iris, aes(x = Sepal.Width)) +
  geom_histogram(binwidth = 0.1, fill = "lightblue", color = "black") +
  labs(title = "Sepal Width Histogram for All Species") +
  theme_minimal()

# Организация гистограмм с помощью пакета cowplot
plot_grid(
  histograms[["setosa"]],
  histograms[["versicolor"]],
  histograms[["virginica"]],
  all_species_hist,
  ncol = 2,
  align = "hv"
)
```

Первая строка кода, где мы вводим переменную `all_species_hist`, создает гистограмму на основе столбца `Sepal.Width` для всех видов ирисов. Ширина столбца (бина) этой гистограммы равна `0.1`, цвет заливки — светло-голубой, а цвет границ — черный. Заголовок гистограммы — `Sepal Width Histogram for All Species`, а тема — `theme_minimal()`.

Код `plot_grid(histograms[["setosa"]], histograms[["versicolor"]], histograms[["virginica"]], all_species_hist, ncol = 2, align = "hv")` использует пакет `cowplot` для организации гистограмм в виде сетки, состоящей из двух столбцов, и их выравнивания по горизонтали и вертикали.

Функция `plot_grid()` принимает в качестве аргументов список графиков. Эти графики располагаются в сетке, количество столбцов и строк в которой задается аргументами `ncol` и `nrow`. Аргумент `align` определяет способ выравнивания графиков друг относительно друга. В данном случае графики выравниваются по горизонтали и вертикали с помощью аргумента `align = "hv"`.

Результат работы функции `plot_grid()` — сетка с графиками, которую можно сохранить в файл или вывести в консоль R (рис. 6.5).

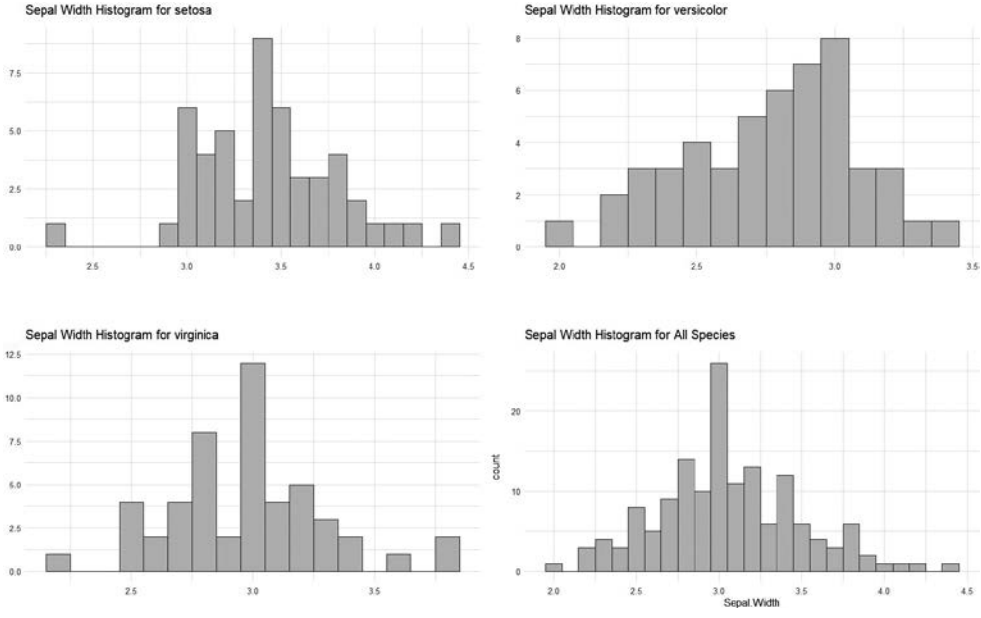


Рис. 6.5. Использование пакета cowplot для создания четырех гистограмм

Столбчатые диаграммы

Столбчатая диаграмма (или столбчатый график) состоит из прямоугольных столбцов одинаковой ширины, используемых для сравнения различных категорий данных. Длина или высота каждого столбца соответствует величине значения, относящегося к определенной категории. Столбцы могут быть расположены и горизонтально, но чаще всего они располагаются вертикально, а промежуток между всеми столбцами должен быть одинаковым.

Вертикальная ось, располагающаяся в левой или правой части графика, называется осью *Y*, а горизонтальная ось в его нижней части — осью *X*. Высота или длина каждого столбца соответствует значению определенной категории данных. Столбчатые диаграммы позволяют легко сравнивать наборы данных, относящиеся к разным группам, и хорошо иллюстрируют изменения, произошедшие в данных за определенный период.

У столбчатых диаграмм есть ряд преимуществ.

- **Их легко читать и интерпретировать.** Столбчатые диаграммы просты и интуитивно понятны, что делает их доступными даже для людей, далеких от статистики или незнакомых с различными методами визуализации данных.
- **Они позволяют эффективно проводить сравнение.** Благодаря этим диаграммам можно быстро и легко сравнивать различные категории данных, выявляя закономерности или тенденции.
- **Подходят для представления категориальных данных.** Столбчатые диаграммы — отличный инструмент представления категориальных или дискретных переменных.

А вот некоторые из недостатков этих диаграмм.

- **Ограниченность в плане представления непрерывных данных.** Столбчатые диаграммы не особенно эффективны для представления непрерывных переменных или данных, содержащих большое количество различающихся значений.
- **Потенциальные проблемы с удобочитаемостью.** Очень большое количество столбцов или слишком близкая величина представленных значений затрудняет восприятие графика.

Создание столбчатых диаграмм с помощью пакета ggplot2

Для начала загрузим необходимые библиотеки, из которых `healthyR` и `healthyR.data` вам, скорее всего, придется установить, если вы еще этого не сделали:

```
library(healthyR.data)
library(healthyR)
library(ggplot2)
library(dplyr)
library(forcats)
library(purrr)
```

Здесь мы загружаем несколько пакетов, которые предоставляют функции и инструменты для манипулирования данными (`dplyr`, `forcats` и `purrr`) и их визуализации (`ggplot2`). Эти пакеты расширяют возможности R в плане работы с данными

и создания графиков. Мы загружаем библиотеку `healthyR.data` для получения набора данных, с которым будем работать, и библиотеку `healthyR` для использования функции `category_counts_tbl()`.

Следующий блок кода используется для создания набора данных, на основе которого мы будем создавать график:

```
df <- healthyR_data |>
  filter(payer_grouping != '?') |>
  category_counts_tbl(
     .count_col = payer_grouping
     , .arrange = TRUE
     , ip_op_flag
  ) |>
  group_by(ip_op_flag) |>
  mutate(order_var = paste0(
     sprintf("%02i", as.integer(rank(n))),
     " - ",
     payer_grouping
     )) |>
  ungroup()
```

На этом этапе для выполнения ряда операций с данными используется конвейер (`|>`).

- `healthyR_data` — это набор данных, предоставляемый пакетом `healthyR.data`.
- Функция `filter()` удаляет строки, в которых значение `payer_grouping` не равно `'?'`.
- Функция `category_counts_tbl()` подсчитывает количество вхождений каждого из уникальных значений, содержащихся в столбце `payer_grouping`, во всех комбинациях значений `ip_op_flag`. Аргумент `.count_col` задает столбец, значения в котором требуется подсчитать, `.arrange` сортирует результаты подсчета в порядке убывания, а `ip_op_flag` группирует данные по этому столбцу.
- Данные группируются по столбцу `ip_op_flag` с помощью функции `group_by()`.
- Функция `mutate()` добавляет новый столбец `order_var`, содержащий отформатированные значения рангов, основанные на столбце `n`, в котором выполнялся подсчет. В данном случае мы применяем функцию `sprintf()`, преобразующую ранг в двузначное целое число, после чего разгруппируем данные, так как в их группах больше нет необходимости.

Теперь создадим график:

```
ggplot(df, aes(x = order_var, y = n)) +
  geom_col(alpha = 0.328) +
  labs(x = "", y = "") +
```

```
theme(legend.position = "none") +
facet_wrap(~ ip_op_flag, scale = "free") +
scale_x_discrete(
   labels = with(df, as.character(payer_grouping) |>
                          set_names(order_var))) +
xlab(NULL) +
theme(axis.text.x = element_text(
   angle = 90, hjust=1, vjust=.5)) +
coord_flip() +
theme_minimal()
```

Этот фрагмент кода создает столбчатую диаграмму с помощью пакета `ggplot2`:

- функция `ggplot()` инициализирует график, а функция `aes()` задает его визуальные параметры (в частности, сопоставляет переменные с осями *X* и *Y*);
- `geom_col()` добавляет на график столбцы, высота каждого из которых соответствует значению `n`. Параметр `alpha` задает степень прозрачности столбцов;
- `labs()` делает подписи осей пустыми;
- `theme()` позволяет настроить тему, например убрать легенду и применить минималистичную тему;
- `facet_wrap()` используется для создания отдельных панелей для каждого уникального значения `ip_op_flag`;
- `scale_x_discrete()` используется для настройки подписей оси *x* с помощью значений `payer_grouping`, индексированных по `order_var`;
- фрагмент `xlab(NULL)` удаляет подпись оси *X*;
- строка `theme(axis.text.x = element_text(angle = 90, hjust = 1, vjust = .5))` настраивает отображение текстовых значений, отложенных вдоль оси *X*, изменяя направление текста с горизонтального на вертикальное;
- функция `coord_flip()` переворачивает оси *X* и *Y*;
- функция `theme_minimal()` применяет к графику минималистичную тему.

Итак, приведенный выше код принимает набор данных, фильтрует и обрабатывает их, а затем создает столбчатую диаграмму с несколькими панелями, каждая из которых отображает столбцы, основанные на результатах подсчетов категориальной переменной. Внешний вид графика настраивается с помощью различных функций и настроек пакета `ggplot2`. Конечный результат выглядит так (рис. 6.6).

В этом разделе вы научились строить довольно сложные горизонтальные столбчатые диаграммы. В частности, вы создали столбчатую диаграмму, отображающую данные, которые относятся к двум разным группам и упорядочены по одному и тому же фактору: распределение объемов по группам плательщиков страховых взносов, покрывающих стационарное и амбулаторное лечение. В следующем разделе вы научитесь строить гантельные диаграммы.

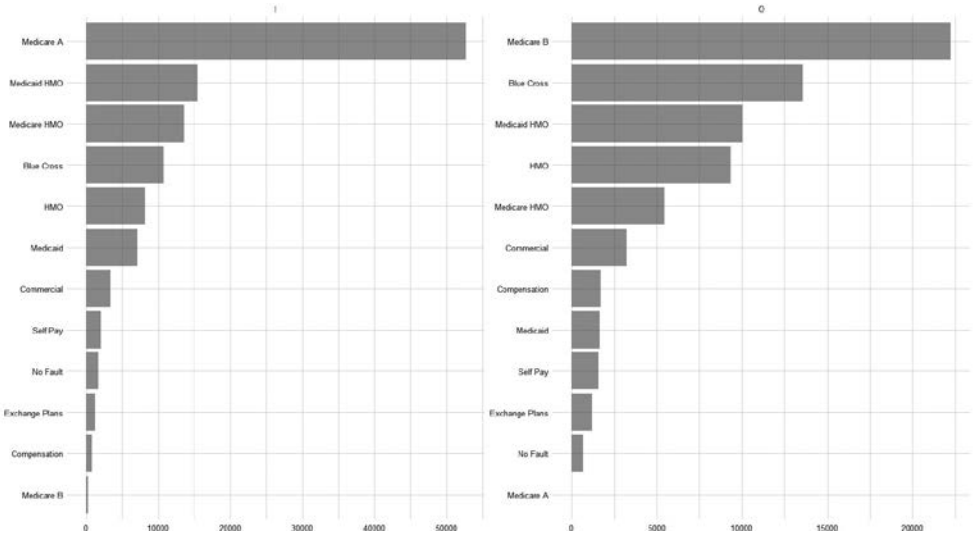


Рис. 6.6. Столбчатые диаграммы, созданные с помощью пакета ggplot2

Гантельные диаграммы

Гантельная диаграмма, иногда называемая *точечной диаграммой с линиями*, позволяет сравнить значения, относящиеся к двум категориям или группам, и визуализировать изменения между ними. Она состоит из двух точек или маркеров, соединенных прямой линией. Каждая точка/маркер представляет значение, а линия — диапазон между ними.

Гантельные диаграммы имеют ряд преимуществ.

- *Подчеркивают различия при сравнении.* Эти диаграммы эффективно демонстрируют различия двух категорий или групп.
- *Отображают диапазоны и дисперсию.* Линия, соединяющая точки на гантельной диаграмме, обозначает диапазон или разброс значений, предоставляя дополнительную информацию.
- *Подходят для представления категориальных и непрерывных данных.* Гантельные диаграммы можно использовать для представления как порядковых, так и непрерывных переменных.

У этих диаграмм есть и недостатки.

- *Ограниченность в плане сравнения множественных категорий.* Гантельные диаграммы больше подходят для сравнения двух категорий или групп. Если этих групп больше, чем две, то график может стать громоздким и запутанным.

- *Меньшая распространенность и привычность*. Гантельные диаграммы используются не так широко, как столбчатые или графики временных рядов, поэтому могут быть незнакомы некоторым людям.

Создание гантельных диаграмм с помощью пакета ggplot2

В R гантельную диаграмму можно создать с помощью библиотеки `ggplot2`.

Первым делом убедитесь в том, что у вас установлены и загружены пакеты `ggplot2` и `dplyr`. При необходимости вы можете установить их с помощью функций `install.packages("ggplot2")` и `install.packages("dplyr")`.

Теперь создадим датафрейм, содержащий два столбца с начальными и конечными значениями, а также категориальную переменную для их группировки. Эти данные могут выглядеть, например, так:

```
# Образец данных
data <- data.frame(
  Category = c("A", "B", "C", "D"),
  Initial = c(10, 15, 8, 12),
  Final = c(18, 22, 14, 16)
)
```

Прежде чем создавать график, добавим среднюю точку для наших данных:

```
data <- data |>
  mutate(Midpoint = (Initial + Final)/2)
```

Теперь мы можем приступить к созданию диаграммы:

```
# Создание гантельной диаграммы с помощью ggplot2
dumbbell_plot <- ggplot(data, aes(x = Category, xend = Category, \
    y = Initial, yend = Final)) +
  # Линии, соединяющие точки
  geom_segment(color = "gray50") +
  # Начальные значения
  geom_point(color = "blue", size = 3) +
  # Конечные значения
  geom_point(aes(y = Final), color = "orange", size = 3) +
  # Подписи средних точек
  geom_text(aes(label = Midpoint), vjust = -0.5, size = 3) +
  labs(title = "Dumbbell Plot",
         x = "Category",
         y = "Values") +
  theme_minimal()

# Отображение диаграммы на экране
dumbbell_plot
```

В этом примере предполагается, что в датафрейме присутствует столбец `Category`, определяющий группы/категории. На графике будут отображены начальные значения (синие точки), конечные значения (красные точки), соединяющие их линии и подписи для средней точки, расположенные поверх линий. Вы можете настроить визуальные параметры, цвета, метки и другие элементы в соответствии с вашими предпочтениями. Результат представлен на рис. 6.7.

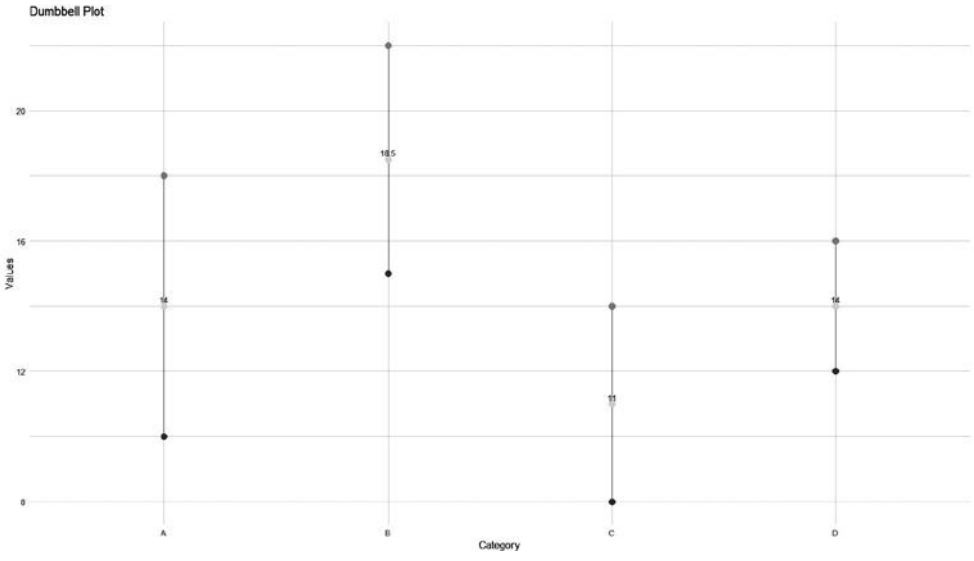


Рис. 6.7. Гантельная диаграмма, созданная с помощью ggplot2

Помните, что это всего лишь простейший пример. В зависимости от имеющихся у вас данных и требований проекта вам может понадобиться скорректировать код, чтобы он лучше отвечал вашим потребностям.

Знакомство с библиотеками для визуализации данных

Визуализация данных — фундаментальный аспект их анализа, и язык Python предлагает обширную экосистему библиотек, позволяющих создавать интересные и информативные графики. В этом разделе мы дадим краткое описание трех известных библиотек для визуализации данных: `plotnine`, `matplotlib` и `plotly`, а также скажем несколько слов еще об одной библиотеке — `seaborn`. Понимание преимуществ и вариантов применения каждой из них позволит вам эффективно представлять ваши данные в отчетах Excel.

plotnine — элегантная грамматика графики

Выше вы уже встречались с `ggplot2` — популярным R-пакетом для визуализации данных, который известен своим выразительным и декларативным синтаксисом. Его адаптация для Python называется `plotnine`.

В основе этого пакета лежит такая концепция, как *грамматика графики*, которая позволяет создавать визуализации путем компоновки отдельных графических элементов. Библиотека `plotnine` отлично справляется с созданием сложных и высококачественных графиков. Благодаря ей вы сможете получить контроль над визуальными параметрами и настраивать каждый аспект вашей визуализации.

matplotlib — классические и настраиваемые графики

`matplotlib` — фундаментальная библиотека для создания статичных высококачественных визуализаций с помощью Python. Она предоставляет широкий набор функций для создания различных типов графиков, в том числе линейных и столбчатых диаграмм, а также диаграмм рассеяния.

Используя `matplotlib`, вы получаете широкие возможности для настройки графиков и сможете контролировать каждую их деталь, от цвета и подписей до шрифтов и линий сетки. Эта библиотека — оптимальный инструмент для создания статичных графиков, предназначенных для исследовательских работ, презентаций и отчетов Excel.

plotly — интерактивные визуализации

Универсальная библиотека Python `plotly` предназначена для создания интерактивных веб-визуализаций. Она позволяет создавать интерактивные графики и дашборды, диаграммы рассеяния, линейные диаграммы и многое другое. Благодаря ей пользователи могут динамически изменять масштаб графика, наводить указатель мыши на точки данных или фильтровать их.

Библиотека `plotly` легко интегрируется с блокнотами Jupyter, что делает ее излюбленным инструментом специалистов по работе с данными и аналитиков. Она идеально подходит для создания интерактивных визуализаций для веб-приложений, отчетов и проектов, связанных с исследованием данных.

К сожалению, весь функционал библиотеки `plotly` невозможно применить в последних версиях Excel из-за мер безопасности, введенных компанией Microsoft. Это ограничение может усложнить интеграцию данной библиотеки в Excel для тех пользователей, которые хотят использовать новейшие возможности этого приложения. Основное внимание в данной книге уделяется улучшению рабочих процессов в Excel, поэтому мы не будем рассматривать все аспекты библиотеки `plotly`.

seaborn — визуализация статистических данных

Библиотека Python `seaborn` тоже предлагает широкий спектр возможностей для настройки и создания сложных графиков, однако по сравнению с такими библиотеками, как `matplotlib` и `plotnine`, достижение тех же результатов часто требует большего объема кода.

Таким образом, понимание возможностей и предпочтительных вариантов использования этих библиотек позволит вам выбирать самый подходящий инструмент для визуализации данных в каждой конкретной ситуации.

В следующих разделах мы более подробно опишем библиотеки `plotnine` и `matplotlib`, приведем практические примеры и расскажем о том, как создавать визуализации данных для отчетов Excel с помощью Python.

Начнем с библиотеки `plotnine`.

Создание графиков с помощью plotnine

В этом разделе мы рассмотрим возможности библиотеки Python `plotnine`, вдохновленной R-пакетом `ggplot2`, о котором мы говорили ранее в данной главе. Как будет показано далее, `plotnine` и `ggplot2` являются родственными пакетами, поскольку, если не учитывать различия в синтаксисе между R и Python, их код и возможности чрезвычайно похожи.

Изучив материал этого раздела, вы научитесь создавать широкий спектр визуализаций, настраивать каждую их деталь и добавлять в них дополнительные слои, позволяющие сделать представление данных более наглядным.

Концепция грамматики графики

Прежде чем приступать к созданию графиков с помощью `plotnine`, необходимо разобраться с концепцией грамматики графики. Этот структурированный подход к визуализации данных составляет основу `plotnine` и позволяет создавать сложные графики, комбинируя данные, визуальные параметры и геометрические объекты.

Разберем некоторые ключевые понятия.

- **Сопоставление данных.** Библиотека `plotnine` связывает данные с визуальными элементами вашего графика, позволяя вам сопоставить переменные с осями *X* и *Y*, цветом, формой, размером и т. д.
- **Визуальные параметры.** Контролируют визуальное представление атрибутов данных. Вы можете использовать их для кодирования такой информации, как цвет, форма и размер.
- **Слои.** Библиотека `plotnine` поощряет применение подхода, предполагающего послойное создание графика, при котором каждый слой добавляет на график новый элемент, например точки, линии или подписи. Это позволяет создавать сложные и информационно насыщенные визуализации.

Создание различных типов графиков

Одно из достоинств библиотеки `plotnine` — универсальность. С ее помощью вы можете создать множество типов графиков в соответствии с особенностями стоящей перед вами задачи.

- **Диаграммы рассеяния.** Простые, но эффективные диаграммы рассеяния помогают визуализировать взаимосвязь, существующую между двумя числовыми переменными.

```
from plotnine import ggplot, aes, geom_point, geom_bar, geom_histogram, \
geom_boxplot, geom_tile, geom_violin, theme_minimal, labs
import pandas

# Образец данных
data = pandas.DataFrame({'x': [1, 2, 3, 4, 5], 'y': [2, 4, 1, 3, 5]})

# Создание диаграммы рассеяния
gg = ggplot(aes(x='x', y='y'), data) + geom_point()
print(gg)
```

График, генерируемый приведенным выше фрагментом кода, показан на рис. 6.8.

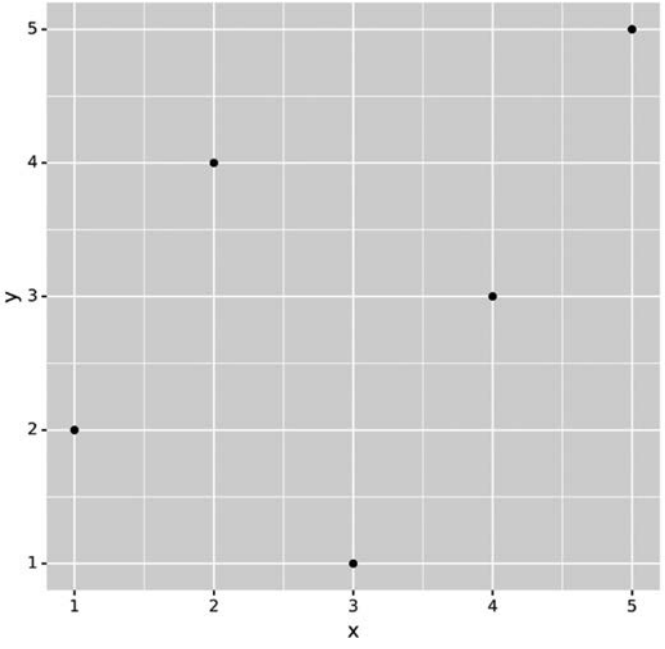


Рис. 6.8. Простейшая диаграмма рассеяния

- **Столбчатые диаграммы.** Это идеальный инструмент для отображения категориальных данных и проведения сравнения.

```
# Образец данных
data = pandas.DataFrame({'category': ['A', 'B', 'C', 'D'],
                         'value': [10, 25, 15, 30]})

# Создание столбчатой диаграммы
gg = ggplot(aes(x='category', y='value'), data) + geom_bar(stat='identity')
print(gg)
```

Результат выглядит так (рис. 6.9).

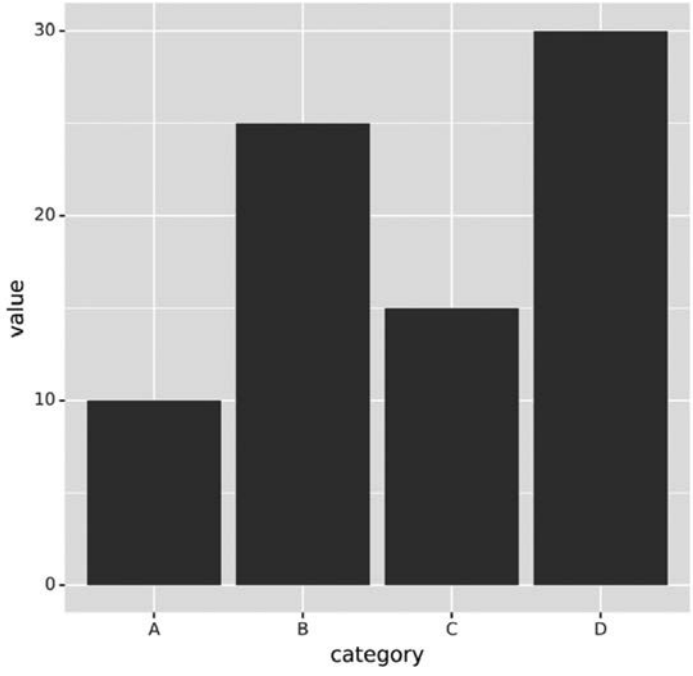


Рис. 6.9. Простейшая столбчатая диаграмма

- **Гистограммы.** Используются для изучения распределения значений одной переменной.

```
# Образец данных
data = pandas.DataFrame({'values': [1, 2, 2, 3, 3, 3, 4, 4, 5]})

# Создание гистограммы
gg = ggplot(aes(x='values'), data) + geom_histogram(binwidth=1,
```

```
fill='blue', \
    color='black', alpha = 0.5)
print(gg)
```

Результат выглядит так (рис. 6.10).

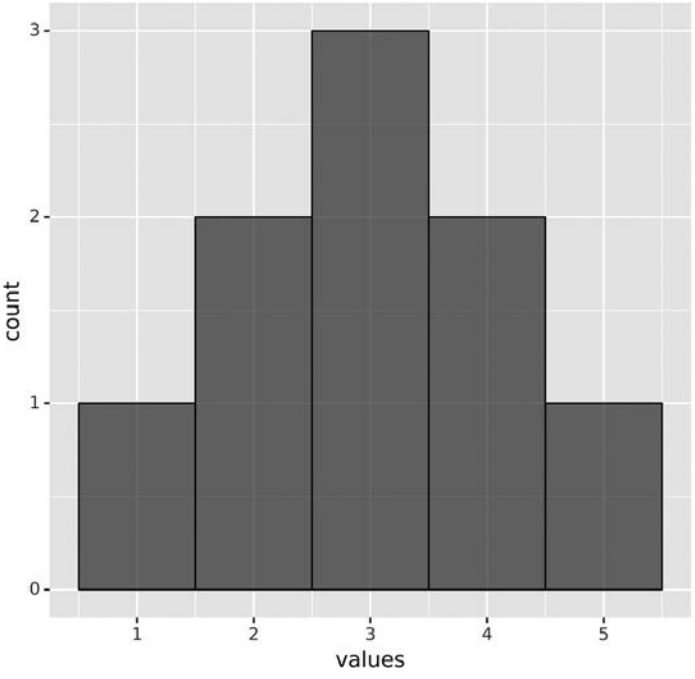


Рис. 6.10. Простейшая гистограмма

Обратите внимание: вплоть до параметра `alpha`, управляющего степенью прозрачности, мы используем синтаксис, полностью аналогичный синтаксису R-пакета `ggplot2`. Этот факт говорит о том, что некоторые библиотеки R и Python очень похожи!

- **Диаграммы размаха** представляют собой распределение набора данных в сжатой форме, отображая медиану, квартили и потенциальные выбросы.

```
# Образец данных
data = pandas.DataFrame({'category': ['A', 'A', 'B', 'B', 'C', 'C'],
                                      'value': [10, 15, 20, 25, 30, 35]})

# Создание диаграммы размаха
gg = ggplot(aes(x='category', y='value'), data) + geom_boxplot()
print(gg)
```

Результат представлен на рис. 6.11.

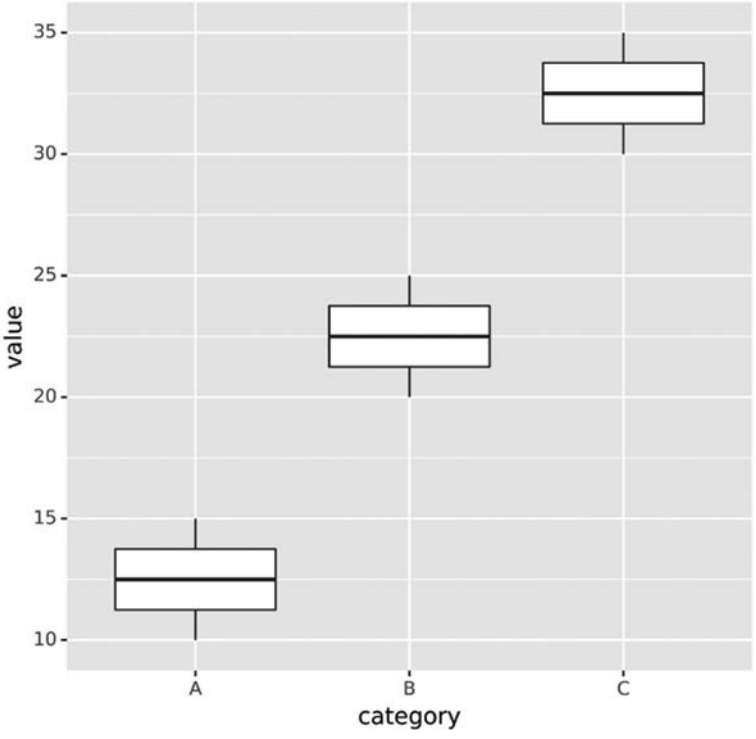


Рис. 6.11. Простейшая диаграмма размаха

- **Тепловые карты.** Идеально подходят для выявления закономерностей в больших наборах данных и используют интенсивность цвета для представления относительной величины значений. Их можно создать с помощью функции `geom_tile()`:

```
data = {
        'x': ['A', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'C', 'C', 'C',
              'C', 'D', 'D', 'D', 'D'],
        'y': ['W', 'X', 'Y', 'Z', 'W', 'X', 'Y', 'Z', 'W', 'X', 'Y',
              'Z', 'W', 'X', 'Y', 'Z'],
        'value': [10, 15, 5, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70,
                  75, 80]
}

# Преобразование данных в объект DataFrame
data = pandas.DataFrame(data)
```

```
# Создание тепловой карты
gg = (ggplot(data, aes(x='x', y='y', fill='value'))
            + geom_tile()
            + theme_minimal()
            + labs(title='Heatmap Example', x='X-Axis', y='Y-Axis',
                   fill='Values'))
print(gg)
```

Результат выглядит так (рис. 6.12).

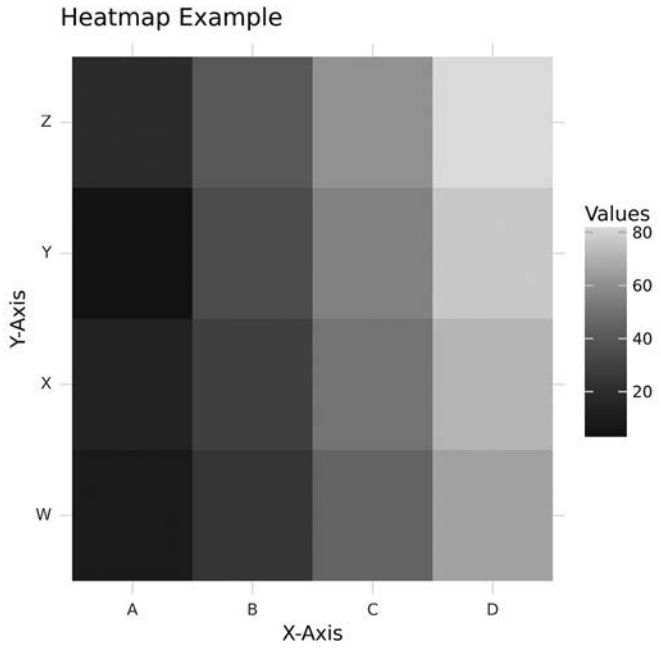


Рис. 6.12. Простейшая тепловая карта

- **Скрипичные диаграммы.** Являются сочетанием диаграммы размаха и ядерной оценки плотности. Предоставляют высокодетализированную иллюстрацию распределения данных.

```
# Образец данных
data = {
        'Category': ['A', 'A', 'B', 'B', 'B', 'C', 'C', 'D', 'D', 'D'],
        'Value': [10, 15, 25, 30, 35, 45, 50, 65, 70, 75]
}
```

```
# Преобразование данных в объект DataFrame
df = pandas.DataFrame(data)

# Создание скрипичной диаграммы
gg = (ggplot(df, aes(x='Category', y='Value', fill='Category'))
            + geom_violin()
            + theme_minimal()
            + labs(title='Violin Plot Example', x='Category', y='Value',
                   fill='Category'))
print(gg)
```

Результат представлен на рис. 6.13.

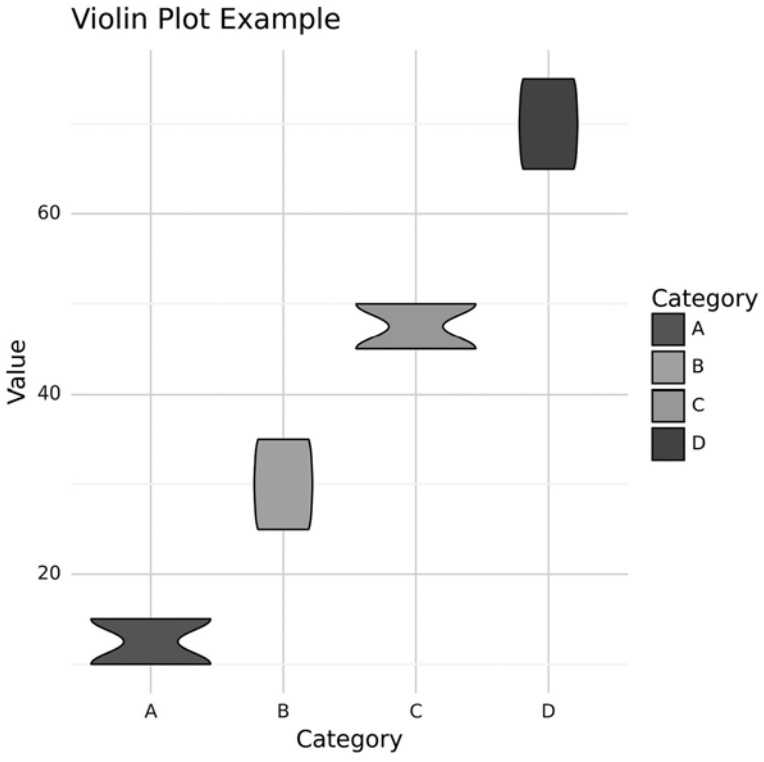


Рис. 6.13. Простейшая скрипичная диаграмма

Теперь, когда мы разобрались с основами, рассмотрим возможности настройки, благодаря которым библиотека `plotnine` так популярна.

Настройка визуальных элементов графика plotnine

Используя библиотеку `plotnine`, вы можете настроить каждый аспект визуализации данных в соответствии с конкретными требованиями. Ниже указаны элементы графиков и описаны способы их настройки.

- **Подписи и заголовки.** Библиотека `plotnine` позволяет настроить подписи для осей *X* и *Y* с помощью функций `xlab()` и `ylab()` соответственно. Чтобы добавить к графику заголовок и подзаголовок, используйте функции `ggtitle()` и `labs()`.

```
# Настройка подписей и заголовков
gg = gg + xlab("Custom X Label") + ylab("Custom Y Label")
gg = gg + ggtitle("Custom Plot Title") + labs(subtitle="Custom Subtitle")
```

- **Оси и легенды.** Вы можете гибко настраивать масштабы осей, добавлять деления шкалы и создавать легенды, используя такие функции, как `scale_x_continuous()` и `scale_y_continuous()`.

```
# Настройка осей и легенд
gg = gg + scale_x_continuous(breaks=[1, 2, 3, 4, 5],
    labels=["One", "Two", "Three", "Four", "Five"])

gg = gg + scale_y_continuous(limits=(0, 10))
gg = gg + scale_color_manual(values={'A': 'red', 'B': 'blue'})
```

- **Темы.** Применяя их, вы можете оформить все графики в едином стиле. Для минималистичного оформления отчетов или презентаций можно использовать такие темы, как `theme_minimal()` или `theme_light()`.

```
# Применение тем
# gg = gg + theme_minimal()
gg = gg + theme_light()
```

- **Текст.** Вы можете форматировать его, то есть менять размер, стиль и выравнивание с помощью функции `theme()`. Настройте такие параметры, как `text`, `text_size`, `text_family` и `text_align`, чтобы добиться желаемого результата.

```
# Форматирование текста
gg = gg + theme(text=element_text(size=12, family="Arial", face="bold",
    color="black"), axis_text_x=element_text(angle=45, hjust=1)
```

Результатом применения описанных выше настроек является следующий график (рис. 6.14).

Пакет `plotnine` позволяет также использовать слои. В следующем подразделе мы поговорим о том, что это и как с их помощью создавать наилучшую визуализацию.

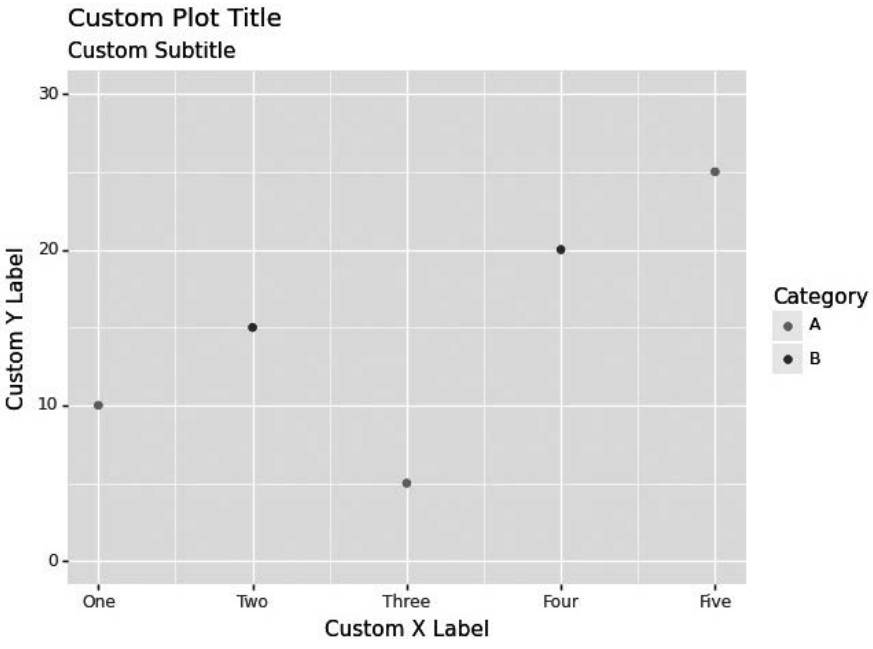


Рис. 6.14. Пользовательские визуальные элементы

Добавление дополнительных слоев

Слои — мощная концепция `plotnine`, которая позволяет наложить на один график несколько слоев, каждый из которых передает различные аспекты ваших данных. В этом подразделе мы рассмотрим следующие распространенные слои:

- *линии тренда* — их добавление позволяет выявить основные закономерности в данных;
- *пределы погрешностей* — используя их, можно визуализировать изменчивость и неопределенность;
- *аннотации* — с помощью текстовых меток или геометрических фигур можно выделить определенные точки или области данных.

Чтобы добавить пределы погрешностей, можно использовать следующий код:

```
from plotnine import (
    ggplot, aes, geom_line, geom_point, geom_errorbar, position_dodge,
    geom_text, labs, geom_smooth
)
import pandas
import numpy
```

```
# Образец данных
data = pandas.DataFrame({
        'x': [1, 2, 3, 4, 5],
        'y': [10, 15, 8, 12, 18],
        'group': ['A', 'A', 'B', 'B', 'C'],
        'error': [1, 2, 1.5, 1, 2.5],
        'label_x': [2, 4, 3, 1, 5],
        'label_y': [16, 11, 6, 13, 17],
        'annotation_text': ['Peak', 'Valley', 'Low', 'High', 'Bottom']
})

# Создание объекта ggplot
gg = ggplot(data, aes(x='x', y='y', group='group')) +
        geom_line() +
        geom_point() +
        geom_errorbar(aes(ymin='y - error', ymax='y + error'), width=0.1, \
        size=0.5, position=position_dodge(width=0.2)) +
        geom_text(aes(x='label_x', y='label_y', label='annotation_text'), size=10)

# Рисование графика
print(gg)
```

Результатом выполнения этого кода является следующий график (рис. 6.15).

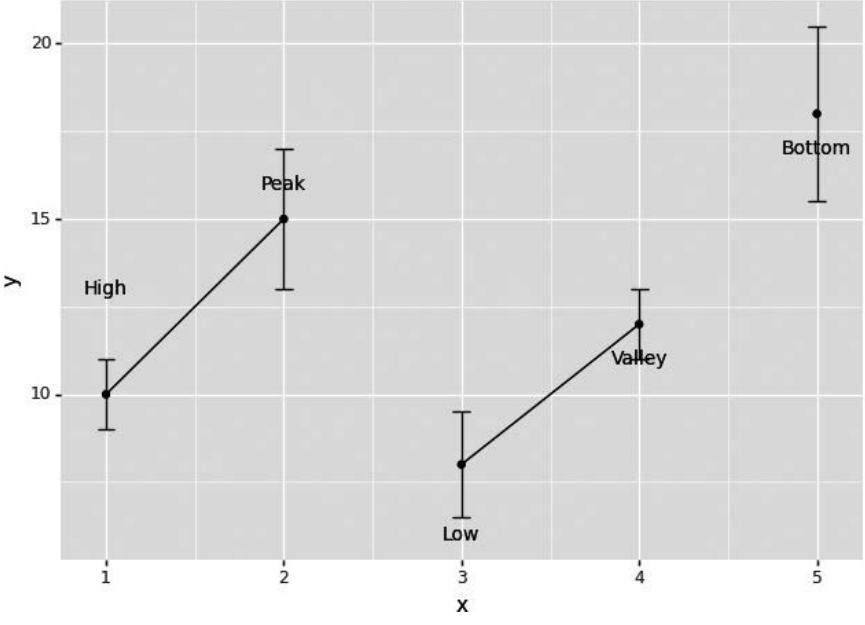


Рис. 6.15. Добавление пределов погрешностей

Для добавления линий тренда можно использовать следующий код:

```
# Образец данных
data = pandas.DataFrame({
        'X': numpy.arange(1, 21),
        'Y': numpy.random.randint(1, 101, size=20)
})

# Создание простого графика
gg = (ggplot(data, aes(x='X', y='Y')) +
            geom_point() +
            labs(title='Scatter Plot with Trendline')
         )

# Добавление линии тренда
gg = gg + geom_smooth(method='lm', se=False, linetype='dashed',
                      color='red', size=1)

print(gg)
```

В результате выполнения этого кода на график будет добавлена линия тренда (рис. 6.16).

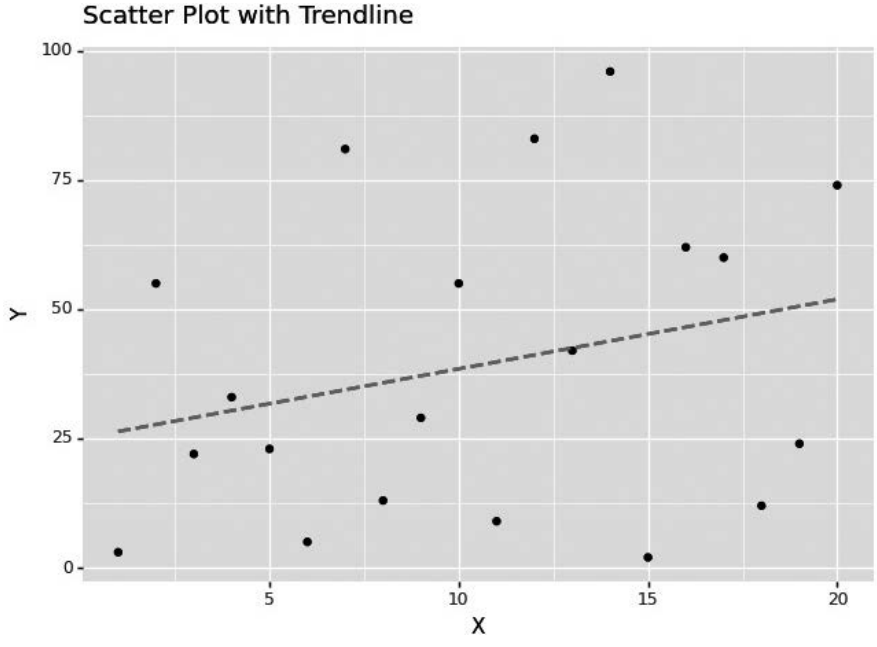


Рис. 6.16. Диаграмма рассеяния с линией тренда

Наконец, для добавления аннотаций мы можем использовать следующий код:

```
# Образец данных
data = pandas.DataFrame({
        'X': numpy.arange(1, 11),
        'Y': numpy.random.randint(1, 101, size=10)
})

# Создание простого графика
gg = (ggplot(data, aes(x='X', y='Y')) +
            geom_point() +
            labs(title='Scatter Plot with Annotations')
        )

# Добавление аннотации и настройка расположения меток вдоль оси Y
# с помощью значения nudge_y, равного 5
gg = gg + geom_text(aes(label='Y'), nudge_y=5, color='blue')

print(gg)
```

В результате выполнения этого кода получается следующий график (рис. 6.17).

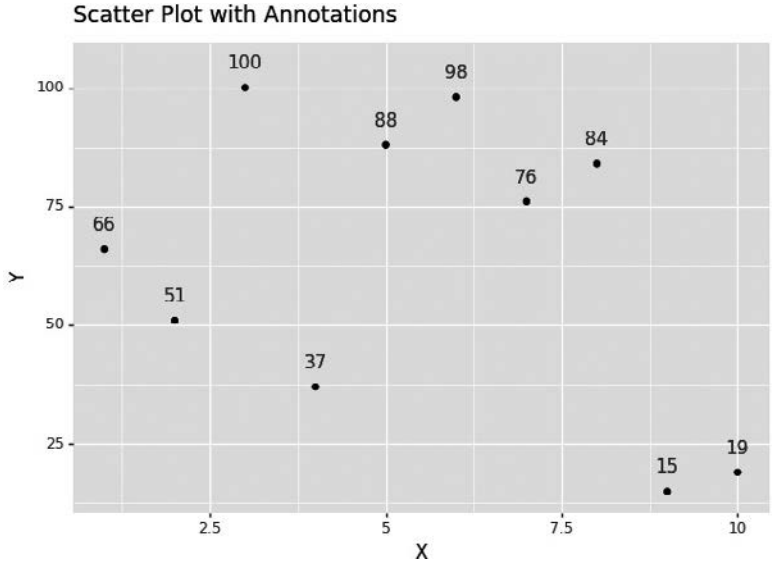


Рис. 6.17. Аннотированная диаграмма рассеяния

Таким образом, какой бы ни была ваша задача — донести до аудитории ценные сведения, сравнить наборы данных или изучить тенденции, — пакет `plotnine`

поможет вам создать информативные и профессиональные визуализации для отчетов Excel.

Теперь посмотрим, как создавать похожие графики с помощью библиотеки `matplotlib`.

Создание графиков с помощью matplotlib

В этом разделе вы познакомитесь с `matplotlib` — одной из самых распространенных библиотек Python для визуализации данных. С ее помощью вы можете создавать различные статичные визуализации, настраивать их внешний вид и адаптировать их под свои потребности. Эта библиотека славится своей универсальностью и позволяет создавать широкий спектр графиков, удовлетворяющих различные потребности в визуализации данных. Вне зависимости от того, чего вы хотите — изучить взаимосвязи между переменными, отобразить категориальные данные или проанализировать распределения значений, — `matplotlib` предоставляет инструменты для создания нужных вам визуализаций.

Далее мы рассмотрим способы создания разных графиков с помощью `matplotlib`, в частности диаграмм рассеяния и размаха, гистограмм, тепловых карт, столбчатых и скрипичных диаграмм.

- **Диаграммы рассеяния.** Код для их создания выглядит так:

```
import numpy
import pandas
import matplotlib.pyplot as plt

### диаграмма рассеяния
data = {
        'Height': [155, 162, 168, 173, 179],
        'Weight': [50, 56, 61, 65, 72]
}

df = pandas.DataFrame(data)

# Создание диаграммы рассеяния
df.plot.scatter(x='Height', y='Weight',
    title='Scatter Plot of Height vs. Weight')

# Сохранение диаграммы в файл (например, .png),
# находящийся в рабочем каталоге
plt.savefig('matplotlib_scatter_plot.png')

# Отображение диаграммы
plt.show()()
```

Этот код создает диаграмму рассеяния, отражающую связь между значениями роста и веса, и сохраняет ее в виде изображения в формате `.png`. Вы можете настроить этот график по своему усмотрению. Результат выглядит следующим образом (рис. 6.18).

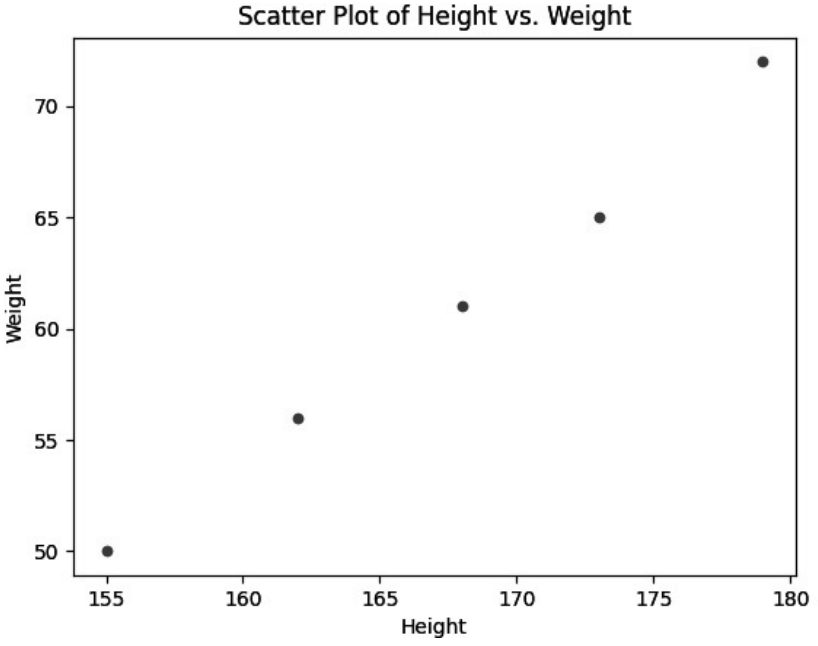


Рис. 6.18. Простейшая диаграмма рассеяния, созданная с помощью matplotlib

- **Столбчатые диаграммы.** Код для их создания представлен ниже:

```
data = {'Category': ['A', 'B', 'C', 'D', 'E'],
        'Values': [15, 28, 24, 20, 32]}

df = pandas.DataFrame(data)

# Создание простейшей столбчатой диаграммы
plt.figure(figsize=(8, 6))
plt.bar(df['Category'], df['Values'], color='skyblue')
plt.xlabel('Categories')
plt.ylabel('Values')
plt.title('Basic Bar Chart')

# Сохранение диаграммы в файл (например, .png)
plt.savefig('matplotlib_bar_chart.png')

plt.show()
```

Этот код создает простую столбчатую диаграмму с категориями, отложенными вдоль оси *X*, и соответствующими значениями, отложенными вдоль оси *Y*. Результат его выполнения показан на рис. 6.19.

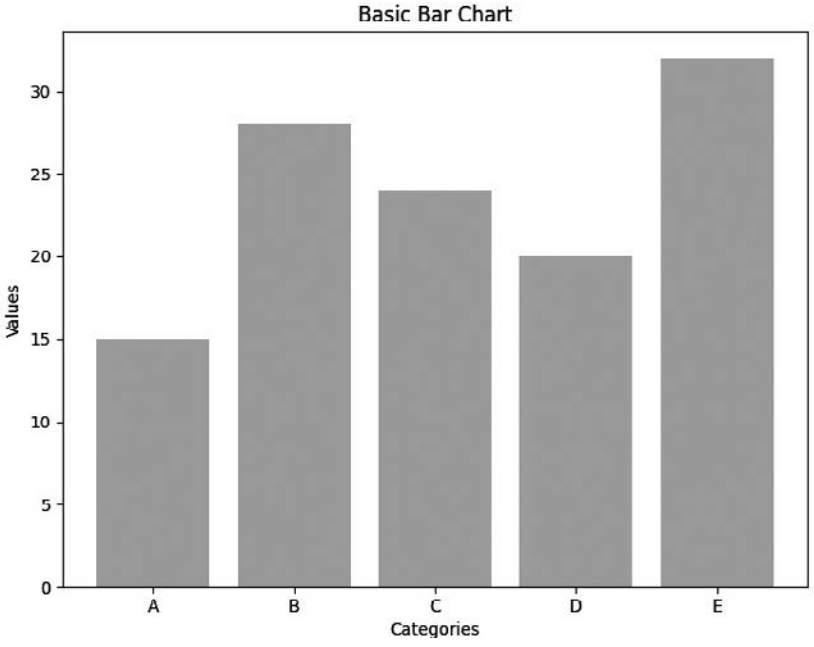


Рис. 6.19. Простейшая столбчатая диаграмма, созданная с помощью matplotlib

- **Гистограммы.** Код для их создания выглядит так:

```
# Генерация случайных данных для создания гистограммы
data = numpy.random.normal(0, 1, 1000)

# Создание простейшей гистограммы
plt.figure(figsize=(8, 6))
plt.hist(data, bins=20, color='lightblue', edgecolor='black')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.title('Basic Histogram')

# Сохранение гистограммы в файл (например, .png)
plt.savefig('matplotlib_histogram.png')

plt.show()
```

Этот код генерирует гистограмму на основе случайных данных, демонстрирующую частотное распределение значений. Вы можете настроить количество

столбцов (`bins`), цвета, подписи и другие свойства гистограммы по своему усмотрению. Результат выполнения данного кода показан на рис. 6.20.

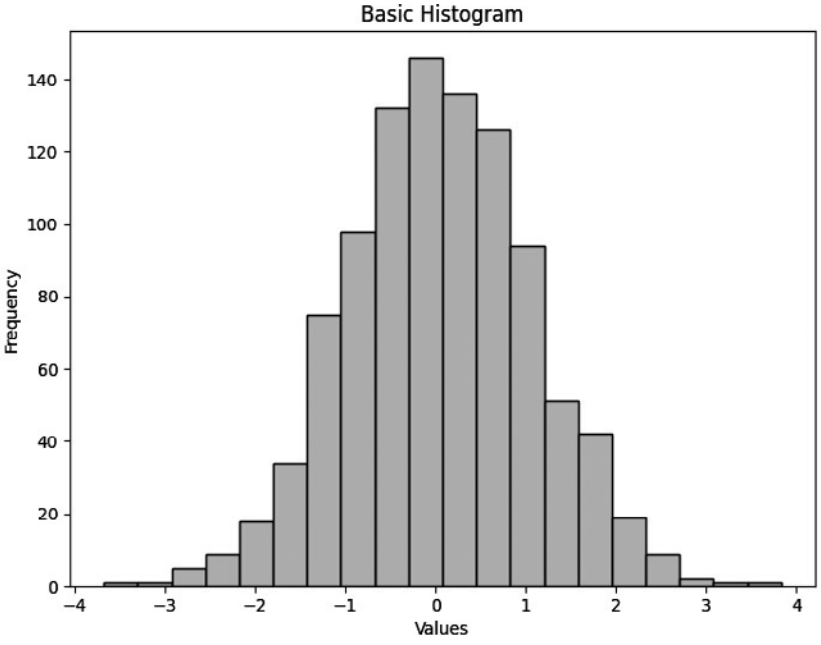


Рис. 6.20. Простейшая гистограмма, созданная с помощью matplotlib

- **Диаграммы размаха.** Код для их создания представлен ниже:

```
# Генерация случайных данных для создания диаграммы размаха
# Три набора случайных данных
data = [numpy.random.normal(0, 1, 100) for _ in range(3)]

# Создание простейшей диаграммы размаха
plt.figure(figsize=(8, 6))
plt.boxplot(data, vert=False, labels=['Set 1', 'Set 2', 'Set 3'])
plt.xlabel('Values')
plt.ylabel('Data Sets')
plt.title('Basic Box Plot')

# Сохранение графика в файл (например, .png)
plt.savefig('matplotlib_boxplot.png')

plt.show()
```

Этот код генерирует простейшую диаграмму размаха на основе случайных значений для сравнения трех наборов данных. При необходимости вы можете

настроить различные параметры этой диаграммы, чтобы изменить ее внешний вид. Результат выполнения кода показан на рис. 6.21.

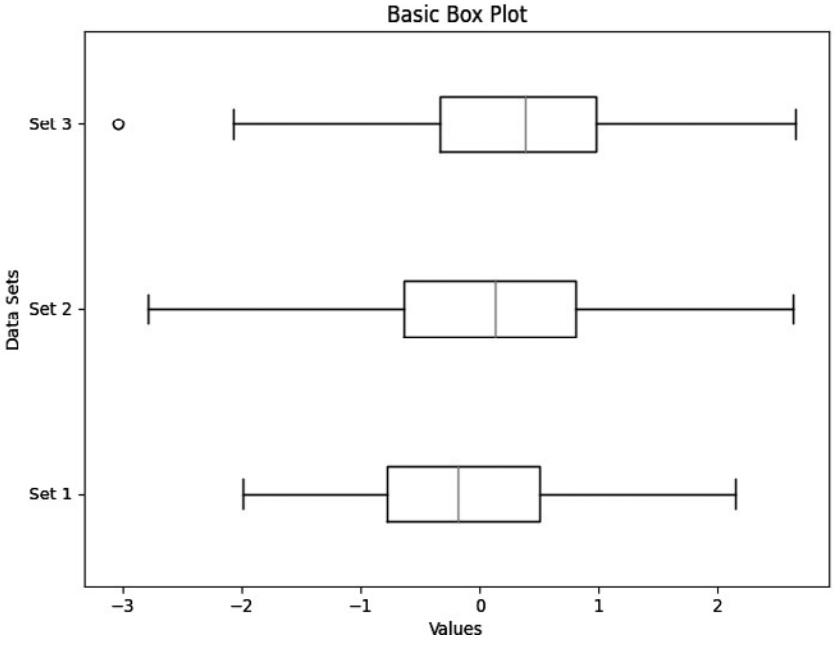


Рис. 6.21. Простейшая диаграмма размаха, созданная с помощью matplotlib

- **Тепловые карты.** Код для их создания выглядит так:

```
# Генерация случайных данных для создания тепловой карты
numpy.random.seed(42)
# Создание матрицы 5 x 5, содержащей случайные значения
data = numpy.random.rand(5, 5)

# Создание тепловой карты
plt.figure(figsize=(8, 6))
heatmap = plt.imshow(data, cmap='viridis', interpolation='nearest')
plt.colorbar(heatmap)
plt.title('Heatmap Example')

# Сохранение графика в файл (например, .png)
plt.savefig('matplotlib_heatmap.png')

plt.show()
```

Этот код генерирует матрицу 5×5, содержащую случайные значения, и создает на ее основе тепловую карту. В данном случае мы использовали цветовую

палитру `viridis`, однако вы можете выбрать другую, чтобы настроить цветовую схему. В данном примере мы показали, как создать простейшую тепловую карту, которую вы можете настроить в соответствии со своими предпочтениями. Результат выполнения кода показан на рис. 6.22.

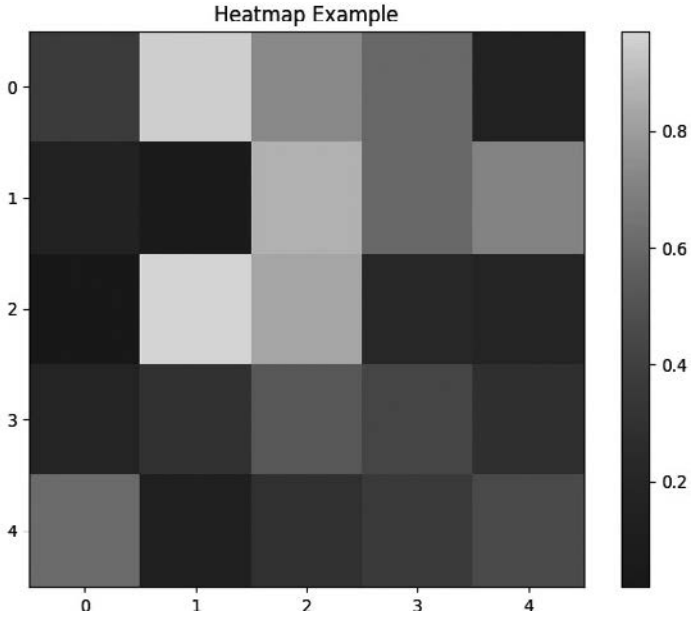


Рис. 6.22. Простейшая тепловая карта, созданная с помощью matplotlib

- **Скрипичные диаграммы.** Код для их создания представлен ниже:

```
# Генерация случайных данных для создания скрипичной диаграммы
numpy.random.seed(42)
data = [numpy.random.normal(0, std, 100) for std in range(1, 4)]

# Создание скрипичной диаграммы
plt.figure(figsize=(8, 6))
plt.violinplot(data, showmedians=True)
plt.title('Violin Plot Example')
plt.xticks([1, 2, 3], ['Group 1', 'Group 2', 'Group 3'])
plt.xlabel('Groups')
plt.ylabel('Values')

# Сохранение графика в файл (например, .png)
plt.savefig('matplotlib_violinplot.png')

plt.show()
```

В этом коде мы генерируем три набора случайных данных и создаем скрипичную диаграмму для визуализации их распределений. Аргумент `showmedians=True` отображает медианные значения внутри каждой скрипки. Вы можете настроить данные, подписи и другие свойства диаграммы в соответствии с собственными потребностями и особенностями вашего набора данных. Получившийся график показан на рис. 6.23.

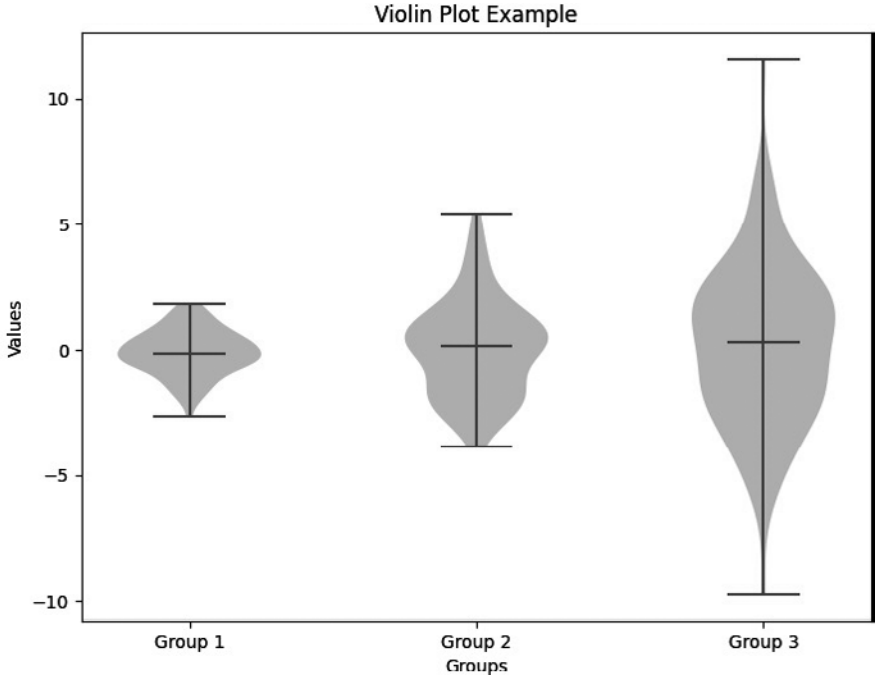


Рис. 6.23. Простейшая скрипичная диаграмма, созданная с помощью matplotlib

Далее мы поговорим о настройке визуальных элементов графика, созданного с помощью библиотеки `matplotlib`.

Настройка визуальных элементов графика matplotlib

В предыдущем разделе мы рассматривали способы, благодаря которым можно настроить графики, созданные с помощью библиотеки `plotnine`. Графики, созданные с использованием `matplotlib`, тоже можно настроить. В этом подразделе мы рассмотрим наиболее важные элементы таких графиков и приведем несколько примеров настройки.

Подписи и заголовки

Библиотека `matplotlib` позволяет легко настраивать подписи осей, а также заголовки и подзаголовки графиков. Вы можете задать подписи для осей *X* и *Y*, озаглавить график и даже добавить подзаголовки, чтобы предоставить дополнительную информацию или контекст. Ниже приведен пример настройки подписей и заголовков в `matplotlib`:

```
# Образец данных
x = [1, 2, 3, 4, 5]
y = [10, 20, 25, 30, 35]

# Создание диаграммы рассеяния
plt.scatter(x, y)

# Настройка подписей и заголовков
plt.xlabel('X-axis Label')
plt.ylabel('Y-axis Label')
plt.title('Custom Title')
plt.suptitle('Subtitle for Additional Context')

# Отображение графика
plt.show()
```

Итоговый график показан на рис. 6.24.

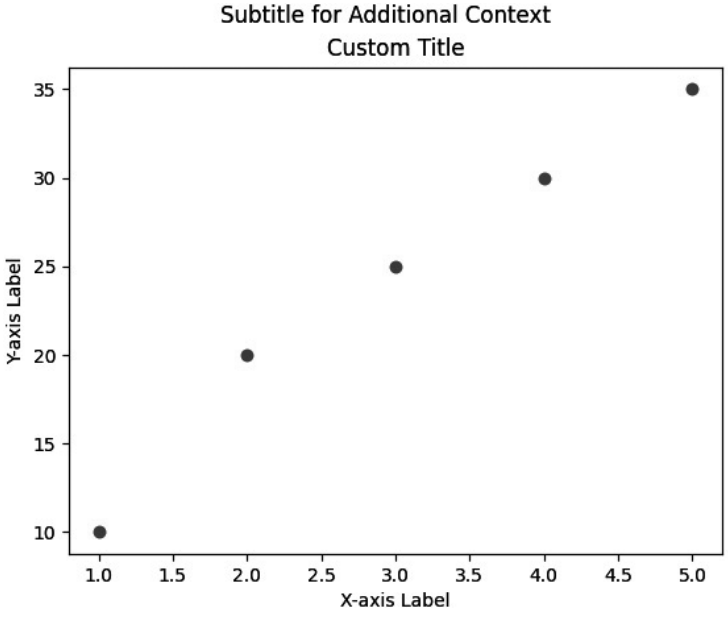


Рис. 6.24. Настройка подписей с помощью matplotlib

Оси и легенды

Библиотека `matplotlib` позволяет настраивать масштабы осей, добавлять деления шкалы и создавать легенды. Вы можете изменить диапазон, масштаб и положение делений на осях *X* и *Y*, а также настроить легенды так, чтобы серии или категории данных были представлены наилучшим образом.

Вот пример настройки осей и легенд графика:

```
import matplotlib.pyplot as plt

# Образец данных
x = [1, 2, 3, 4, 5]
y = [10, 20, 25, 30, 35]

# Создание линейного графика
plt.plot(x, y, label='Data Series A')

# Настройка осей и легенды
plt.xlim(0, 6)
plt.ylim(0, 40)
plt.xticks([1, 2, 3, 4, 5])
plt.yticks([0, 10, 20, 30, 40])
plt.legend()

# Отображение графика
plt.show()
```

Итоговый график показан на рис. 6.25.

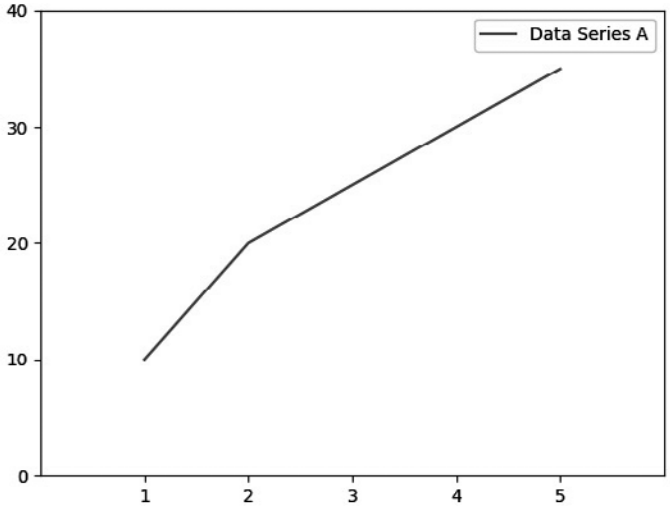


Рис. 6.25. Настройка осей и легенд с помощью matplotlib

Темы

Библиотека `matplotlib` предлагает множество тем, с помощью которых вашим визуализациям можно придать единообразный вид. Вы можете выбирать предопределенные стили, соответствующие эстетике вашего отчета или презентации. Ниже показан код, позволяющий применить другую тему. В качестве примера мы выбрали тему `ggplot`.

```
import matplotlib.pyplot as plt

# Применение другой темы
plt.style.use('ggplot')

# Образец данных и создание графика
x = [1, 2, 3, 4, 5]
y = [10, 20, 25, 30, 35]
plt.plot(x, y)

# Отображение графика
plt.show()
```

Итоговый график выглядит так (рис. 6.26).

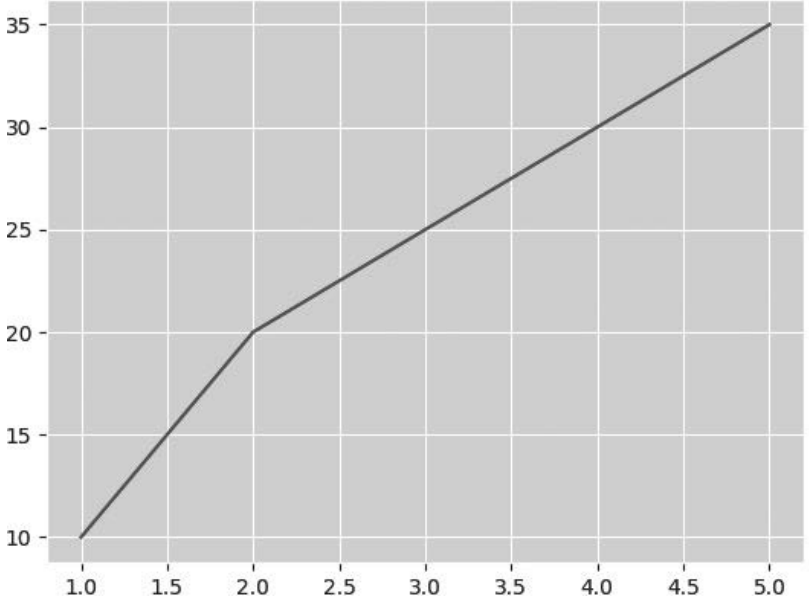


Рис. 6.26. Применение тем с помощью matplotlib

Форматирование текста

Библиотека `matplotlib` позволяет управлять параметрами форматирования текста. Вы можете настроить размер шрифта, использовать полужирное или курсивное начертание, а также задать выравнивание для таких текстовых элементов, как заголовки, подписи и аннотации. Вот пример форматирования текста:

```
import matplotlib.pyplot as plt

# Образец данных и создание графика
x = [1, 2, 3, 4, 5]
y = [10, 20, 25, 30, 35]
plt.plot(x, y)

# Настройка параметров форматирования текста
plt.title('Custom Title', fontsize=16, fontweight='bold', color='blue')
plt.xlabel('X-axis Label', fontsize=12, fontstyle='italic', color='green')
plt.ylabel('Y-axis Label', fontsize=12, fontweight='bold', color='red')

# Отображение графика
plt.show()
```

Итоговый график выглядит следующим образом (рис. 6.27).

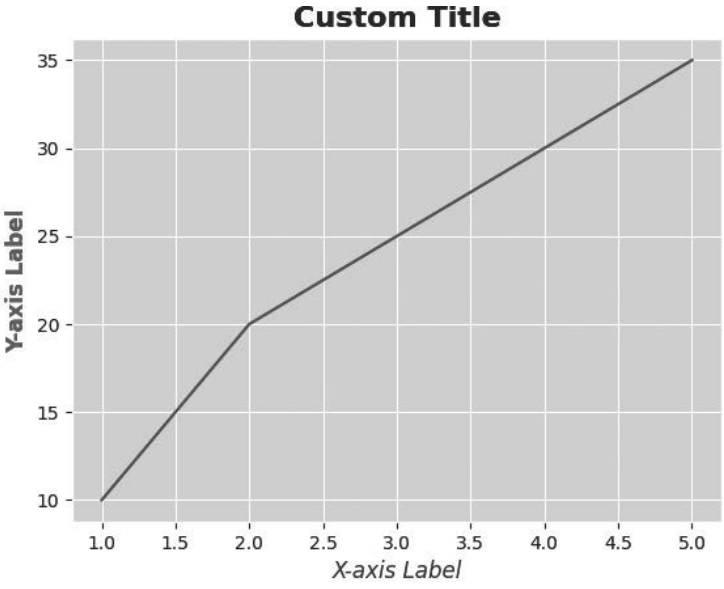


Рис. 6.27. Форматирование текста с помощью matplotlib

На этом мы завершаем обзор самых популярных и мощных библиотек Python для визуализации данных.

Встраивание визуализаций в файлы Excel

Возможно, при работе с такими библиотеками Python для визуализации данных, как `matplotlib` и `plotnine`, вам придется интегрировать полученные визуализации в отчеты или электронные таблицы Excel. Умение встраивать графики и диаграммы в файлы Excel может стать ценным дополнением к вашему набору навыков анализа данных. В этом разделе мы рассмотрим базовый процесс встраивания в Excel визуализаций, созданных с помощью `matplotlib` и `plotnine`, который поможет вам использовать все возможности библиотек Python для визуализации данных наряду с функциями создания отчетов, предусмотренными в Excel.

Базовый процесс встраивания визуализаций

Процесс встраивания визуализаций, созданных с помощью `matplotlib` и `plotnine`, в Excel обычно сводится к экспорту графиков или диаграмм в виде файлов изображений (например, в формате PNG или JPEG) и последующему импорту этих изображений в лист Excel. Данный подход не обеспечивает такой уровень интерактивности, как некоторые другие методы, но позволяет легко улучшить ваши отчеты Excel с помощью визуализаций данных, созданных в среде Python.

Основные этапы этого процесса описаны ниже.

1. **Создание визуализации.** Создайте визуализацию с помощью `matplotlib` или `plotnine` в своем сценарии Python или блокноте Jupyter. Настройте ее в соответствии со своими потребностями.
2. **Экспорт визуализации в виде изображения.** Используйте библиотеку `matplotlib` или `plotnine` для сохранения визуализации в виде файла изображения. Для этого хорошо подходят такие распространенные форматы, как PNG или JPEG. Убедитесь, что место сохранения изображения доступно из листа Excel.
3. **Вставка изображения в файл Excel.** Существуют два варианта решения этой задачи.
   - *Вариант 1.* В данном случае мы вставляем изображение вручную. Откройте лист Excel и перейдите в то место, куда необходимо вставить визуализацию. Выберите пункт меню Вставка ▸ Иллюстрации ▸ Рисунки, чтобы импортировать файл изображения, созданный на втором этапе. Затем вы можете изменить размер и положение изображения на листе Excel.

- *Вариант 2*. Изображение вставляется программно из среды Python с помощью библиотеки `pywin32`.

```
import win32com.client as win32

# Инициализация приложения Excel
excel = win32.gencache.EnsureDispatch('Excel.Application')
excel.Visible = True

# Создание новой рабочей книги
workbook = excel.Workbooks.Add()

# Определение (абсолютного) пути к изображению
image_path = 'путь\\к\\файлу\\image.png'

# Вставка изображения в конкретный лист и ячейку
sheet = workbook.ActiveSheet
cell = sheet.Range("A1") # Вы можете указать ячейку, в которую
                         # будет вставлено изображение

# Добавление изображения на рабочий лист (возможно, придется
# корректировать значения ширины (Width) и высоты (Height))
sheet.Shapes.AddPicture(image_path, LinkToFile=False,
    SaveWithDocument=True, Left=cell.Left, Top=cell.Top,
    Width=300, Height=200)

# Сохранение рабочей книги
workbook.SaveAs('файл_excel_с_изображением.xlsx')

# Закрытие приложения Excel
excel.Application.Quit()
```

4. **Планирование запуска сценария для обновления (необязательно).** Если ваши данные регулярно изменяются, то вы можете запланировать запуск сценария, генерирующего изображение и вставляющего его в Excel, используя метод из описанного выше варианта 2. Таким образом, ваша визуализация будет регулярно обновляться. Такое автоматическое обновление можно задать с помощью приемов, описанных в главе 4.

Этот подход обеспечивает статическое представление визуализаций и тем не менее является вполне практичным способом дополнения ваших отчетов Excel диаграммами и графиками, созданными с помощью Python.

В целом встраивание визуализаций в отчеты Excel позволяет улучшить представление данных, их настройку и автоматизацию. С чем бы вы ни работали — данными о продажах, финансовыми отчетами или любыми другими массивами информации, — этот подход позволит вам создавать информативные отчеты.

Резюме

В этой главе вы узнали о методах визуализации данных, с помощью которых можно улучшить рабочие процессы и отчеты Excel, а также о том, как реализовать их на языках R и Python.

Мы подробно рассмотрели самые популярные и мощные R-пакеты и библиотеки Python. Вы узнали о распространенных видах графиков, применяемых для анализа данных и представления результатов, и познакомились с примерами их использования.

В следующей главе мы рассмотрим еще один способ, позволяющий донести важную информацию, связанную с необработанными данными.

7 Сводные таблицы

В сфере анализа данных и работы с электронными таблицами сводные таблицы представляют собой мощный инструмент, позволяющий пользователям преобразовывать и обобщать большие наборы необработанных данных, представляя их в более удобном и понятном формате. Эти таблицы делают возможными организацию данных и их динамический анализ, поэтому стали незаменимым инструментом для профессионалов в различных областях.

В этой главе мы рассмотрим следующие темы:

- знакомство со сводными таблицами;
- создание таблицы с помощью R-функции `xtabs`;
- создание таблицы с помощью R-пакета `gt`;
- создание сводных таблиц с помощью R-пакета `tidyquant`;
- создание сводных таблиц в Python и управление ими с помощью `win32com` и `pywin32`.

Технические требования

Файлы с кодом из примеров этой главы доступны по адресу https://github.com/PacktPublishing/Extending-Excel-with-Python-and-R/tree/main/Chapter7.

Вот R-пакеты, которые будут использоваться в этой главе:

- `tidyquant >= 1.0.6`;
- `gt >= 0.10.0`.

Знакомство со сводными таблицами

Сводная таблица — это средство обработки данных, используемое в таких программах для работы с электронными таблицами, как Microsoft Excel или Google Sheets, для анализа и извлечения важной информации из сложных наборов данных. Такая таблица позволяет пользователям реструктурировать и сжимать большие объемы

данных в лаконичный и доступный для понимания формат, упрощая процесс принятия решений и изучения данных. Благодаря своей адаптивности, интерактивности и простоте сводные таблицы применяются в различных отраслях при решении аналитических задач.

Прежде чем переходить к созданию сводных таблиц, поговорим об их компонентах.

- **Строки и столбцы.** Это основные компоненты сводных таблиц. В строках содержатся отдельные записи или наблюдения, а в столбцах — определяющие эти записи атрибуты или переменные.
- **Значения.** Сводные таблицы позволяют пользователям агрегировать и обобщать данные, вычисляя значения на основе определенных показателей, таких как сумма, среднее значение, количество или процент.
- **Фильтры и срезы.** Благодаря им пользователи могут сосредоточиться на определенных подмножествах данных в сводной таблице, чтобы провести более детальный анализ. Эти инструменты особенно полезны при работе с большими наборами данных.
- **Подписи строк и столбцов.** Сводные таблицы позволяют пользователям перетаскивать атрибуты в подписи строк и столбцов, динамически определяя макет и структуру таблицы.

Сводная таблица дает возможность реорганизовать и обобщать данные на основе заданных пользователем критериев. Это позволяет пользователю получить многомерную сводку со сведениями, которые могут быть неочевидны в исходном наборе данных. Принцип работы сводной таблицы описан ниже.

- **Выбор.** Пользователи выбирают набор данных, который требуется проанализировать, и определяют столбцы, содержащие релевантные атрибуты и показатели.
- **Организация.** Пользователи размещают эти атрибуты и показатели в определенных областях макета сводной таблицы, таких как строки, столбцы, значения и фильтры.
- **Вычисление.** Сводные таблицы автоматически вычисляют указанные показатели для различных комбинаций выбранных атрибутов. Например, таблица может отобразить общий объем продаж каждой категории товаров в разных регионах.
- **Интерактивность.** Пользователи могут легко изменять макет сводной таблицы, перетаскивая атрибуты, что позволяет исследовать данные в режиме реального времени.

Сводные таблицы обладают рядом преимуществ, которые делают их ценным инструментом анализа данных.

- **Обобщение данных.** Сводные таблицы помогают пользователям выявлять закономерности, тенденции и аномалии в больших массивах информации.
- **Быстрое понимание.** Пользователи могут быстро понять суть данных, не прибегая к написанию сложного кода или использованию запутанных формул.
- **Гибкий анализ.** Сводные таблицы позволяют пользователям экспериментировать с различными перспективами путем реорганизации атрибутов, тем самым помогая им выявлять корреляции и тенденции.
- **Формирование отчетов.** Сводные таблицы играют ключевую роль в создании разнообразных отчетов, дашбордов и визуализаций.
- **Очистка данных.** Сводные таблицы можно использовать перед анализом данных для выявления недостающих значений, выбросов или несоответствий.

Создание таблицы с помощью R-функции xtabs

R-функция `xtabs()` создает таблицу сопряженности из столбцов факторов, содержащихся в датафрейме. На вход этой функции подается формула: `x ~ y`. Таблица сопряженности отображает частотное распределение двух или более категориальных переменных. В описанном далее примере, демонстрирующем принцип работы функции `xtabs()`, мы будем использовать набор данных `UCBAdmissions`.

Синтаксис функции `xtabs()`:

```
xtabs(formula, data, subset, sparse, na.action, addNA, exclude,
drop.unused.levels)
```

Значение каждого из аргументов описано ниже:

- `formula` — объект-формула с перекрестно классифицирующимися переменными (разделенными символом ~);
- `data` — датафрейм, содержащий переменные, которые используются в формуле;
- `subset` — необязательное выражение, указывающее на подмножество наблюдений, которое будет использоваться;

- `sparse` — логическое значение, указывающее на необходимость возвращения разреженной матрицы;
- `na.action` — функция для обработки пропущенных значений;
- `addNA` — логическое значение, указывающее на необходимость добавления строки и столбца для пропущенных значений;
- `exclude` — вектор значений, подлежащих исключению из таблицы;
- `drop.unused.levels` — логическое значение, указывающее на необходимость отбрасывания неиспользуемых уровней факторов.

Чтобы использовать функцию `xtabs()` с набором данных `UCBAdmissions`, сначала нужно преобразовать его в датафрейм с помощью функции `as.data.frame()`. Набор данных `UCBAdmissions` содержит количество абитуриентов мужского и женского пола, а также количество зачисленных и незачисленных абитуриентов обоих полов. С помощью функции `xtabs()` мы можем создать таблицу сопряженности, которая отображает частотное распределение таких переменных, как пол и статус зачисления. Код для решения этой задачи выглядит так:

```
# Преобразование набора данных в датафрейм
df <- as.data.frame(UCBAdmissions)

# Создание таблицы сопряженности с помощью функции xtabs()
xtabs(Freq ~ Gender + Admit, df)
        Admit
Gender   Admitted Rejected
  Male       1198     1493
  Female      557     1278
```

Результатом работы функции `xtabs()` будет таблица, отображающая частотное распределение полов (представленных строками) и статусов зачисления (представленных столбцами). Значения в таблице соответствуют частотам различных комбинаций пола и статуса зачисления.

Теперь, когда вы научились создавать таблицы сопряженности с помощью базовой функции R, можно перейти к изучению R-пакета `gt`, с помощью которого тоже можно создавать сводные таблицы.

Создание таблицы с помощью R-пакета gt

Одно из главных достоинств R-пакета `gt` — удобство использования. Он имеет довольно простой синтаксис, который позволяет быстро создавать таблицы в среде R. Кроме того, этот пакет предлагает широкий спектр возможностей для настройки, благодаря которым пользователи могут создавать таблицы, отвечающие их потребностям.

Еще одно достоинство пакета `gt` — высокая производительность. Он особенно полезен для пользователей, которым приходится создавать таблицы на основе больших наборов данных, так как позволяет делать это быстро и эффективно.

Кроме того, пакет `gt` предлагает широкий выбор вариантов оформления, с помощью которых можно придать таблицам визуально приятный вид, облегчающий их чтение. Пользователи могут настраивать шрифты, цвета и форматирование таблиц в соответствии с конкретным фирменным стилем или дизайнерскими предпочтениями.

Наконец, пакет `gt` имеет открытый исходный код, а значит, постоянно обновляется и совершенствуется сообществом R-разработчиков. Благодаря этому данный пакет остается актуальным и востребованным, регулярно пополняясь новыми функциями и подвергаясь улучшениям.

Рассмотрим пример использования пакета `gt` с набором данных `mtcars`.

В следующем фрагменте кода мы проверяем, установлен ли пакет `gt`. Если нет, то используем функцию `install.packages` для его установки.

```
# Пакет gt
if (!require(gt)) {
    install.packages("gt", dependencies = TRUE)
}
```

Здесь мы загружаем два дополнительных пакета: `dplyr` и `tibble`, которые предоставляют полезные функции и структуры данных, позволяющие обрабатывать данные и выполнять их анализ в среде R.

```
library(dplyr)
library(tibble)
```

В этом фрагменте кода мы выполняем несколько операций над набором данных `mtcars`:

```
tab <- mtcars |>
  rownames_to_column() |>
  arrange(factor(cyl), mpg) |>
  group_by(cyl) |>
  slice(1:3) |>
  gt()
```

Мы делаем следующее:

- `rownames_to_column()` — преобразуем имена строк набора данных в обычный столбец, с которым можно работать. Эта функция относится к пакету `tibble`;
- `arrange(factor(cyl), mpg)` — сортируем набор данных в порядке возрастания сначала по столбцу `cyl`, а затем по столбцу `mpg`. Эта функция, а также функции `group_by` и `slice` относятся к пакету `dplyr`;

- `group_by(cyl)` — группируем набор данных по столбцу `cyl`;
- `slice(1:3)` — выбираем первые три строки в каждой группе;
- `gt()` — создаем таблицу для отображения полученных данных с помощью пакета `gt`.

В следующем фрагменте кода мы добавим в таблицу заголовок `Performance` для группы столбцов: `mpg`, `disp`, `hp`, `drat`, `wt` и `qsec`. Эти столбцы связаны с производительностью автомобилей.

```
tab <- tab |>
  tab_spanner(
    label = "Performance",
    columns = c(mpg, disp, hp, drat, wt, qsec)
  )
```

Аналогичным образом во фрагменте ниже мы добавляем в таблицу заголовок `Specs` для столбцов: `vs`, `am`, `gear` и `carb`, которые содержат информацию о технических характеристиках автомобилей.

```
tab <- tab |>
  tab_spanner(
    label = "Specs",
    columns = c(vs, am, gear, carb)
  )
```

В последнем фрагменте мы задаем название таблицы, состоящее из заголовка и подзаголовка. Заголовком является фраза `The Cars of mtcars` в формате Markdown, а подзаголовком — `These are some fine automobiles`.

```
tab <- tab |>
  tab_header(
    title = md("The Cars of **mtcars**"),
    subtitle = "These are some fine automobiles"
  )
tab
```

Итак, приведенный выше код, написанный на языке R, загружает необходимые пакеты, создает настраиваемую таблицу с информацией о технических характеристиках автомобилей на основе набора данных `mtcars` и задает для нее заголовок. Теперь, когда мы разобрались с кодом, посмотрим на результат его выполнения. Если вы хотите сделать это самостоятельно, то вызовите переменную `tab` в консоли. Итоговая таблица выглядит следующим образом (рис. 7.1).

The Cars of **mtcars**

These are some fine automobiles

	Performance						Specs			
	mpg	disp	hp	drat	wt	qsec	vs	am	gear	carb
4										
Volvo 142E	21.4	121.0	109	4.11	2.780	18.60	1	1	4	2
Toyota Corona	21.5	120.1	97	3.70	2.465	20.01	1	0	3	1
Datsun 710	22.8	108.0	93	3.85	2.320	18.61	1	1	4	1
6										
Merc 280C	17.8	167.6	123	3.92	3.440	18.90	1	0	4	4
Valiant	18.1	225.0	105	2.76	3.460	20.22	1	0	3	1
Merc 280	19.2	167.6	123	3.92	3.440	18.30	1	0	4	4

Рис. 7.1. Использование набора данных mtcars с пакетом gt

Создание сводных таблиц с помощью R-пакета tidyquant

Функция `pivot_table()` из библиотеки `tidyquant` — полезный инструмент для создания сводных таблиц из датафреймов в среде R. С ее помощью вы можете указать строки, столбцы, значения и функции агрегирования для вашей таблицы, а также задействовать другие возможности, такие как сортировка, форматирование и фильтрация.

Чтобы воспользоваться функцией `pivot_table()`, сначала необходимо загрузить библиотеку `tidyquant` с помощью команды `library(tidyquant)`. Затем мы можем передать функции датафрейм в качестве первого аргумента, за которым следуют остальные аргументы, определяющие таблицу. Например, если вы хотите создать таблицу, показывающую среднюю длину и ширину чашелистика у разных видов ирисов, можете использовать следующий код:

```
# Загрузка библиотеки tidyquant
library(tidyquant)
library(purrr)
```

```
# Создание сводной таблицы
pivot_table(.data = iris,
            .rows = ~ Species,
            .values = c(~ mean(Sepal.Length), ~ mean(Sepal.Width))) |>
set_names("Species","Mean_Sepal_Length","Mean_Sepal_Width")
```

Результат выполнения этого кода будет таким:

```
# Тиббл: 3 × 3
  Species    Mean_Sepal_Length Mean_Sepal_Width
  <fct>                  <dbl>            <dbl>
1 setosa                  5.01             3.43
2 versicolor              5.94             2.77
3 virginica               6.59             2.97
```

Мы выполняли следующие действия.

- **Загрузка библиотек.** Мы начали с загрузки двух R-пакетов: `tidyquant` и `purrr`, которые предоставляют функции и инструменты, позволяющие обрабатывать данные и выполнять их анализ.
- **Применение функции** `pivot_table()`. Она используется для изменения формы данных, содержащихся в наборе `iris`, и принимает три основных аргумента:
  - `.data` — набор данных, с которым вы собираетесь работать. В нашем случае это набор данных `iris`;
  - `.rows` — этот аргумент указывает предпочтительный способ группировки или категоризации данных. В приведенном выше коде данные группируются по столбцу `Species`, в котором представлены различные виды ирисов;
  - `.values` — данный аргумент указывает, для каких столбцов необходимо рассчитать и отобразить значения. В нашем случае вычисляется среднее арифметическое двух столбцов `Sepal.Length` и `Sepal.Width` для каждого вида ирисов.
- **Применение функции** `set_names`. Она используется для переименования столбцов результирующей таблицы. В данном случае имена столбцов изменяются на `Species`, `Mean_Sepal_Length` и `Mean_Sepal_Width`.

Итак, приведенный выше код берет набор данных `iris`, группирует их по видам, вычисляет среднюю длину и среднюю ширину чашелистика для каждого вида, а затем переименовывает столбцы итоговой таблицы, чтобы сделать ее более понятной. В результате мы получаем новую таблицу, показывающую среднюю длину и среднюю ширину чашелистика для каждого вида ирисов.

Теперь, когда мы разобрались с кодом на языке R, реализуем то же самое на Python.

Создание сводных таблиц на Python и управление ими с помощью win32com и pywin32

Сводные таблицы реализованы в качестве функции таких приложений для работы с электронными таблицами, как Microsoft Excel, однако вы можете создавать и управлять ими и на программном уровне с помощью Python. В этом разделе вы научитесь задействовать их потенциал с помощью библиотек `win32com` и `pywin32`.

Как вы уже знаете, сводные таблицы предоставляют динамичный способ обобщения, изучения и извлечения ценных сведений из сложных наборов данных. Однако при работе с большими объемами информации настройка и конфигурирование сводных таблиц часто оказываются трудоемким и чреватым ошибками процессом, нередко требующим ручного вмешательства.

Python в сочетании с библиотеками `win32com` и `pywin32` может упростить и автоматизировать процесс создания и управления сводными таблицами. Благодаря такой комбинации аналитики и другие специалисты могут эффективно обрабатывать большие объемы данных, не тратя ресурсы на решение повторяющихся задач и настройку сводных таблиц вручную.

Настройка среды Python

Прежде чем приступать к созданию сводных таблиц, следует установить в среду Python нужные библиотеки. Пакеты `win32com` и `pywin32` необходимы для взаимодействия с Microsoft Excel, поэтому убедитесь в том, что они установлены в вашей системе. Мы рассмотрели процесс их установки и налаживания соединения между Python и Excel в разделе «Интеграция VBA с Python с помощью pywin32» и, в частности, в подразделе «Настройка среды» главы 3. Обратитесь к ней, если вы еще не установили библиотеку `pywin32`.

Создание сводных таблиц

Мы начнем с самых основ и научим вас создавать сводные таблицы с нуля. Ниже приведены этапы решения этой задачи.

1. **Подключение к Excel.** Создайте экземпляр Excel и откройте рабочую книгу. Если она не существует, то вы можете создать новую.

```
import win32com.client as win32

# Создание рабочей книги Excel и добавление листа
excel = win32.gencache.EnsureDispatch('Excel.Application')
workbook = excel.Workbooks.Add()
worksheet = workbook.Worksheets(1)
```

2. **Добавление данных на рабочий лист.** Для создания сводной таблицы вам понадобятся данные. Обычно они уже есть, но для этого примера вы можете добавить образец данных на рабочий лист следующим образом:

```
worksheet.Cells(1, 1).Value = 'Name'
worksheet.Cells(1, 2).Value = 'Category'
worksheet.Cells(1, 3).Value = 'Sales'

worksheet.Cells(2, 1).Value = 'John'
worksheet.Cells(2, 2).Value = 'Electronics'
worksheet.Cells(2, 3).Value = 1000

worksheet.Cells(3, 1).Value = 'Alice'
worksheet.Cells(3, 2).Value = 'Clothing'
worksheet.Cells(3, 3).Value = 800

worksheet.Cells(4, 1).Value = 'John'
worksheet.Cells(4, 2).Value = 'Clothing'
worksheet.Cells(4, 3).Value = 300

# При необходимости добавьте дополнительные данные
```

3. **Выбор диапазона данных.** Определите диапазон данных, который вы хотите использовать для создания сводной таблицы. Для этого укажите начальную и конечную ячейки:

```
data_range = worksheet.Range('A1:C4')  # При необходимости измените диапазон
```

4. **Создание сводной таблицы.** Теперь вы можете создать сводную таблицу на основе выбранного диапазона данных. Укажите место, в которое вы хотите ее поместить, а также столбцы, которые будут использоваться для создания ее строк, столбцов и значений.

```
# Добавление рабочего листа в рабочую книгу для размещения
# сводной таблицы
pivot_table_sheet = workbook.Worksheets.Add()
pivot_table_sheet.Name = 'Pivot Table'

# Создание кэша Pivot Cache на основе диапазона данных
pivot_cache = workbook.PivotCaches().Create(SourceType=1,
    SourceData=data_range)

# Создание сводной таблицы на новом листе с помощью кэша Pivot Cache
pivot_table = pivot_cache.CreatePivotTable(
    TableDestination=pivot_table_sheet.Cells(3, 1),
    TableName='MyPivotTable')
```

```
# Добавление полей строк, столбцов и данных
pivot_table.PivotFields('Name').Orientation = 1 # поле строки

pivot_table.PivotFields('Category').Orientation = 2 # поле столбца

pivot_table.PivotFields('Sales').Orientation = 4 # поле данных

# Добавление вычисляемых полей
calculated_field = pivot_table.CalculatedFields().Add(
    "Total Sales", "=SUM(Sales)")

# Обновление сводной таблицы для применения изменений
pivot_table.RefreshTable()
```

В этом примере параметр `SourceType` задает тип источника данных для сводной таблицы `PivotTable`. Фрагмент `SourceType = 1` означает, что этим источником данных является электронная таблица Excel. Параметр `SourceType` может принимать одно из следующих значений (или представляющее какое-то из них число от 1 до 3):

- `xlDatabase` — говорит о том, что источником данных является электронная таблица Excel или внешняя база данных. Это наиболее распространенный тип источника данных для сводных таблиц;
- `xlExternal` — указывает на то, что источником данных является OLAP-куб или внешний ресурс, к которому невозможно получить доступ непосредственно из Excel;
- `xlConsolidation` — указывает на то, что источником данных является результат объединения, то есть сводная таблица, которая объединяет данные из нескольких рабочих листов или рабочих книг.

5. **Сохранение рабочей книги и закрытие Excel.** Не забудьте сохранить рабочую книгу Excel с только что созданной сводной таблицей.

```
workbook.SaveAs('PivotTableExample.xlsx')
workbook.Close()
excel.Quit()
```

Если вы не хотите сохранять таблицу в рабочем каталоге Python, то при сохранении листа Excel можете указать полный путь.

Вот и все! Вы создали сводную таблицу с помощью библиотеки Python `pywin32`. Вы можете скорректировать данные, расположение сводной таблицы и параметры ее форматирования по своему усмотрению.

Теперь, когда у вас есть готовая сводная таблица, посмотрим, как ее можно изменить в соответствии с вашими потребностями.

Работа со сводными таблицами

Над таблицами можно выполнять различные операции, такие как фильтрация, сортировка и обновление данных. Рассмотрим пример работы со сводной таблицей.

1. Сначала откройте файл Excel из предыдущего подраздела и выберите созданную сводную таблицу.

```
import win32com.client as win32

# Подключение к Excel
excel = win32.gencache.EnsureDispatch('Excel.Application')

# Открытие рабочей книги со сводной таблицей
# Укажите путь к своей книге
workbook = excel.Workbooks.Open('PivotTableExample.xlsx')
worksheet = workbook.Worksheets(1)

# Получение доступа к сводной таблице
# Укажите имя своей сводной таблицы
pivot_table = worksheet.PivotTables('MyPivotTable')
```

2. Вы можете фильтровать данные в сводной таблице по значениям. В этом примере мы отфильтруем поле `Category`, чтобы отобразить только значение `Electronics`:

```
# Фильтрация по значению (для этого необходимо сделать поле
# полем страницы, а не полем столбца)
category_field = pivot_table.PivotFields('Category')
category_field.Orientation = 3 # поле страницы
category_field.CurrentPage = "Electronics"
```

3. Возможно, вам придется отсортировать строки или столбцы сводной таблицы. В этом примере мы отсортируем поле `Name` в порядке возрастания:

```
# Сортировка строк или столбцов
name_field = pivot_table.PivotFields('Name')
name_field.AutoSort(1, "Name")
```

4. Если исходные данные изменились, то вы можете обновить сводную таблицу, чтобы учесть это изменение.

```
# Определение нового диапазона исходных данных
new_source_data_range = 'Sheet1!A1:C2'

# Обновление свойства SourceData объекта Table сводной таблицы
pivot_table.TableRange2(workbook.Sheets('Sheet1').Range(new_source_data_
range))

# Обновление данных
pivot_table.RefreshTable()
```

5. Завершив работу со сводной таблицей, сохраните изменения и закройте рабочую книгу.

```
workbook.Save()
workbook.Close()
excel.Quit()
```

ПРИМЕЧАНИЕ

Не держите электронную таблицу открытой (в Excel) при обращении или взаимодействии с ней из среды Python, так как это приводит к возникновению ошибок com_errors, которые бывает довольно трудно отладить.

Настроив сводную таблицу, вы можете захотеть усовершенствовать ее, сгруппировав некоторые (или все) категории, чтобы отобразить информацию наилучшим образом. В следующем подразделе мы поговорим о том, как это сделать.

Группировка данных в сводных таблицах

Группировка данных в сводных таблицах помогает получить более глубокое представление об имеющемся у вас массиве информации. Вы можете сгруппировать данные по определенным критериям, таким как диапазоны дат, числовые интервалы или пользовательские категории. В этом подразделе мы поговорим о группировке данных, выполняемой с помощью Python и библиотеки `pywin32`.

Группировка по датам

Одним из распространенных вариантов группировки данных является их агрегирование по диапазонам дат. Например, вы можете сгруппировать данные о продажах по месяцам или кварталам. Для этого в сводной таблице можно выполнить группировку по датам.

Для начала создадим несколько образцов данных.

```
# Создание образцов данных
import pandas as pd
import random
from datetime import datetime, timedelta
import win32com.client as win32
import os
import numpy as np

data = {
    'Date': [datetime(2023, 1, 1) + timedelta(days=i) for i in range(365)],
    'Sales': [random.randint(100, 1000) for _ in range(365)]
}
```

```
df = pd.DataFrame(data)

# Создание объекта ExcelWriter и запись объекта DataFrame на рабочий лист Excel
df.to_excel("GroupingExample.xlsx", sheet_name='Sheet1', index=False)
```

Сохранив данные в листе Excel (по умолчанию они сохраняются в рабочем каталоге Python, но можно указать и другое место), мы можем выполнить уже привычные действия: открыть лист Excel, добавить специальную вкладку для сводной таблицы и создать ее. Шаги, которые были описаны ранее, в данном случае опущены (но доступны на GitHub). Их последовательность выглядит следующим образом.

```
# Подключение к Excel

# Открытие рабочей книги Excel и добавление листа

# Добавление рабочего листа в рабочую книгу для размещения сводной таблицы

# Определение диапазона данных, которые будут использоваться
# в качестве входных для сводной таблицы

# Создание кэша Pivot Cache на основе диапазона данных

# Создание сводной таблицы на новом листе с помощью кэша Pivot Cache

# Добавление поля 'Date' в строки (Rows) и определение переменной date_field,
# как это было сделано с name_field в примере, приведенном выше

# Добавление вычисляемых полей
calculated_field = pivot_table.CalculatedFields().Add("Total Sales", "=SUM(Sales)")

# Группировка по месяцам
date_field.Subtotals = [False]*12
date_field.NumberFormat = 'MMMM YYYY'

# Сортировка строк (Rows)
date_field.AutoSort(1, "Date")
```

В этом примере мы создали сводную таблицу, добавили в строки поле `Date` и вычисляемое поле `Total Sales`. Затем мы указали, что хотим представить даты в формате «месяц год», после чего отсортировали отформатированные даты.

Чтобы добавить в сводную таблицу поле группировки, нужно знать, какие значения можно объединить (обратите внимание, что значения различаются, несмотря на то что вследствие форматирования не отображают конкретных дней):

```
# Подсчет уникальных значений для каждого элемента столбца date в сводной таблице
date_values = pd.DataFrame([item.Value for item in date_field. \
    PivotItems()], columns = ['date'])
unique_values = pd.DataFrame(np.transpose(np.unique(date_values, \
    return_counts=True)), columns=['date', 'count'])
date_values_count = date_values.merge(unique_values).drop_duplicates()
```

```
# Группировка по месяцам
# Настройка свойства GroupOn
date_range = pivot_table_sheet.Range(f"A4:A{starting_row + \
    date_values_count['count'].iloc[0]}")
date_range.Group()

# При желании вы можете использовать описанный выше метод для группировки
# дат, относящихся к другим месяцам

# Примечание: сводная таблица была изменена, теперь вторая группа начинается
# со строки start_row + 2, а не starting_row + 32
```

В результате выполнения приведенного выше кода создается поле группировки `Date2`, в котором дни января группируются в значение `Group1value`, а остальные объединяются в группы, состоящие из одной даты. Используя предыдущий пример, вы можете перебрать остальные уникальные значения дат в формате «месяц год» и так же сгруппировать их. Обратите внимание на то, что теперь поле `Total Sales` вычисляется на основе групп.

В завершение мы меняем формат нового поля группировки на `MMMM YYYY` (месяц год), возвращаем данные в поле `Date` в исходное состояние, чтобы они отображали полную дату, и для облегчения восприятия скрываем подробные сведения о группах. Наконец, мы обновляем сводную таблицу, а затем сохраняем и закрываем файл Excel.

```
# Изменение формата столбца группировки на "месяц год" и возвращение
# данных в поле Date в исходное состояние, предусматривающее
# отображение полной даты
pivot_table.PivotFields('Date2').NumberFormat = 'MMMM YYYY'
date_field.NumberFormat = 'DD MMMM YYYY'

# Сокрытие подробных сведений о сгруппированных значениях
for item in pivot_table.PivotFields('Date2').PivotItems():
    item.ShowDetail = False

# Обновление данных
pivot_table.RefreshTable()

# pivot_table.PivotFields('Date2').Orientation = 2

# Сохранение и закрытие файла
workbook.Save()
workbook.Close()
excel.Quit()
```

Это лишь один из примеров того, как с помощью группировок в сводных таблицах можно выполнять более эффективный анализ данных. Вы можете настроить группировки по своему усмотрению в зависимости от имеющегося у вас набора данных и конкретных целей анализа.

Резюме

В этой главе вы научились использовать функционал сводных таблиц с помощью R и Python. Сводные таблицы — ценные инструменты анализа данных, с помощью которых можно динамически обобщать и изучать большие массивы информации. Приемы, описанные в этой главе, помогут вам раскрыть весь потенциал сводных таблиц, а также автоматизировать процессы их создания, использования и улучшения.

В начале главы мы поговорили о важности сводных таблиц в контексте анализа данных, а затем обсудили процесс установки основных библиотек, необходимых для подготовки вашей среды R или Python к работе с Excel.

Далее мы углубились в процесс создания сводных таблиц с нуля и выяснили, как выбирать источники данных, организовывать строки и столбцы, а также настраивать внешний вид таблицы.

Затем мы обсудили способы, которые позволяют динамически фильтровать, сортировать и обновлять данные, а также изменять сводные таблицы в зависимости от поставленной задачи.

Помимо всего прочего, мы познакомили вас с такими расширенными функциями сводных таблиц, как вычисляемые поля и группировка данных, которые позволяют изучать данные на более глубоком уровне и расширять ваши аналитические возможности.

Таким образом, освоение принципов работы со сводными таблицами в среде R и Python позволило вам получить полный набор навыков, необходимых для эффективного решения задач, связанных с анализом данных. Эти знания помогут вам преобразовывать данные в информативные выводы, оптимизировать рабочий процесс и принимать более обоснованные решения. Способность автоматизировать сводные таблицы и управлять ими с помощью R и Python — ценный актив в современном мире, ориентированном на данные, и теперь вы хорошо подготовлены к тому, чтобы использовать все возможности этих таблиц.

Часть III

Разведочный, статистический анализ данных и анализ временных рядов

В этой части мы поговорим о том, как проводить разведочный анализ данных с помощью R и Python в целях выявления ценных сведений и закономерностей в данных Excel. Вдобавок обсудим основы статистического анализа, в том числе методы линейной и логистической регрессии. Кроме того, поговорим об анализе временных рядов, статистике, создании графиков и методах прогнозирования, позволяющих выявлять тенденции и закономерности в темпоральных данных.

8 Разведочный анализ данных с помощью R и Python

Разведочный анализ данных (exploratory data analysis, EDA) — важнейший этап, с которого специалисты по работе с данными начинают свое исследование. Он предполагает систематическое изучение и визуализацию набора данных — все это позволяет выявить существующие в нем закономерности, тенденции и ценные сведения. EDA выполняется для того, чтобы можно было лучше понять данные, выявить потенциальные проблемы или аномалии, а также обосновать последующие решения, связанные с анализом и моделированием.

Как правило, процесс разведочного анализа начинается с применения ряда методов обобщения данных, таких как вычисление основных статистических показателей (среднего арифметического, медианы и стандартного отклонения), создание частотных распределений, изучение типов данных и недостающих значений. Эти предварительные шаги позволяют получить представление о структуре и качестве набора данных.

Центральную роль в разведочном анализе играет визуализация данных. Специалисты по работе с данными создают различные графики, в том числе гистограммы, диаграммы размаха и рассеяния, а также тепловые карты, чтобы визуализировать распределение и взаимосвязи в данных. Эти визуализации помогают выявить выбросы, асимметрию, корреляции и кластеры, что способствует обнаружению интересных закономерностей, имеющихся в данных.

Изучение категориальных переменных предполагает создание столбчатых, круговых или составных столбчатых диаграмм, с помощью которых можно получить представление о распределении различных категорий и их взаимосвязях. Это особенно полезно при решении таких задач, как сегментация клиентов или анализ рынка.

Кроме того, EDA предполагает оценку взаимосвязей между переменными. Специалисты по работе с данными используют корреляционные матрицы, диаграммы рассеяния и регрессионный анализ для выявления таких связей и зависимостей. Понимание этих связей может помочь в выборе признаков для моделирования и выявлении потенциальных проблем мультиколлинеарности.

В ходе EDA часто выполняются преобразование и очистка данных для решения таких проблем, как наличие выбросов, отсутствующие значения и асимметрия. На основе результатов такого исследования могут приниматься решения относительно восстановления (импутации) данных, масштабирования или кодирования категориальных переменных.

В целом EDA — критически важный этап рабочего процесса специалистов по работе с данными, поскольку он закладывает основу последующего моделирования данных, проверки гипотез и принятия решений. Он позволяет аналитикам делать обоснованный выбор методов предварительной обработки данных, конструирования признаков и моделирования, предоставляя всю информацию о характеристиках и нюансах набора данных. EDA помогает гарантировать, что выводы и решения будут приниматься с учетом сути данных.

В этой главе мы рассмотрим следующие темы:

- анализ данных с помощью R-пакета `skimr`;
- использование R-пакета `GGally`;
- использование R-пакета `DataExplorer`;
- очистка данных Excel в среде Python;
- выполнение разведочного анализа данных с помощью Python.

Технические требования

Файлы с кодом из примеров этой главы доступны по адресу https://github.com/PacktPublishing/Extending-Excel-with-Python-and-R/tree/main/Chapter%208.

В разделах, посвященных языку R, мы будем использовать следующие библиотеки:

- `skimr 2.1.5`;
- `GGally 2.2.0`;
- `DataExplorer 0.8.3`.

Анализ данных с помощью R-пакета skimr

R-пакет `skimr` представляет собой полезный инструмент для получения сводной статистики по переменным, которые могут быть представлены в различных формах, таких как датафреймы и векторы. Этот пакет предоставляет конечному пользователю больше статистических показателей по сравнению с базовой функцией языка R `summary()` и является относительно простым в плане применения и настройки.

Чтобы использовать пакет `skimr`, его нужно установить из репозитория CRAN с помощью команды `install.packages("skimr")`. После установки пакет можно загрузить командой `library(skimr)`. Функция `skim()` используется для обобщения всего набора данных. Например, `skim(iris)` предоставляет сводную статистику по набору данных `iris`. Вывод функции `skim()` отображается горизонтально и предусматривает один раздел для каждого типа переменной и одну строку для каждой из переменных.

Кроме того, в этом пакете есть функция `skim_to_wide()`, которая преобразует вывод функции `skim()` в широкий формат. Это бывает полезно при экспорте сводной статистики в электронную таблицу или другое внешнее приложение.

В целом пакет `skimr` — это полезный инструмент для быстрого и простого получения сводной статистики по переменным в среде R. Он может оказаться особенно полезным при решении задач исследования и очистки данных, поскольку позволяет быстро выявить существующие в них проблемы, обобщить и визуализировать ключевые статистические показатели.

Теперь, когда вы получили общее представление о пакете `skimr`, посмотрим, как он работает. Код на языке R для получения сводной статистики по набору данных `iris` выглядит так:

```
# Пакет skimr
if(!require(skimr)){install.packages("skimr")}
library(skimr)
skim(iris)
```

Посмотрим, что происходит в каждой строке кода.

- Строка `if(!require(skimr)){install.packages("skimr")}` проверяет, установлен ли пакет `skimr`. Если нет, то пакет устанавливается с помощью команды `install.packages("skimr")`. Благодаря этому его можно будет использовать в коде, который будет написан впоследствии.
- Строка `library(skimr)` загружает пакет `skimr` в сеанс R. После загрузки пакета вы можете использовать его функции и возможности.
- Строка `skim(iris)` вызывает функцию `skim()` из пакета `skimr` и применяет ее к набору данных `iris`. Функция `skim()` генерирует сводку по набору данных, в том числе статистику и информацию о каждой переменной (столбце) в этом наборе.

Теперь обсудим ожидаемый результат. После выполнения команды `skim(iris)` вы должны увидеть в консоли R сводку по набору данных `iris`, которая будет содержать статистику и прочую информацию, например:

- *количество значений* для каждой переменной;
- *количество пропущенных значений* для каждой переменной (если таковые имеются);
- *количество уникальных значений* для каждой переменной;
- *среднее значение* для каждой числовой переменной;
- *минимальное значение* для каждой числовой переменной;
- *максимальное значение* для каждой числовой переменной;
- *стандартное отклонение* для каждой числовой переменной;
- прочие статистические показатели.

Результат будет выглядеть примерно так:

```
> skim(iris)
── Data Summary ────────────────────────
                           Values
Name                       iris
Number of rows             150
Number of columns          5
_______________________________
Column type frequency:
  factor                   1
  numeric                  4
_______________________________
Group variables            None

── Variable type: factor ──────────────────────────────────────────────────────
_______________________________
  skim_variable n_missing complete_rate ordered n_unique top_counts
1 Species               0             1 FALSE          3 set: 50, ver: 50,
vir: 50

── Variable type: numeric ─────────────────────────────────────────────────────
_______________________________
  skim_variable n_missing complete_rate mean    sd  p0 p25  p50 p75  p100 hist
1 Sepal.Length          0             1
5.84  0.828  4.3  5.1  5.8   6.4  7.9  ▆▇▇▅▂
2 Sepal.Width           0             1
3.06  0.436  2    2.8  3     3.3  4.4  ▁▆▇▂▁
3 Petal.Length          0             1
3.76  1.77   1    1.6  4.35  5.1  6.9  ▇▁▆▇▂
4 Petal.Width           0             1
1.20  0.762  0.1  0.3  1.3   1.8  2.5  ▇▁▆▅▂
```

Данный вывод предоставляет всю информацию о наборе данных `iris`, помогая быстро понять его структуру и получить ключевые статистические показатели. Для получения этих результатов вы также можете передать функции `skim()` сгруппированный объект `tibble`.

Теперь, когда вы разобрались с тем, как пакет `skimr` можно использовать для изучения данных, можем перейти к рассмотрению пакета `GGally`.

Использование R-пакета GGally

По сути, `GGally` — расширение R-пакета `ggplot2`, чрезвычайно популярного в силу своей элегантности и гибкости. `GGally` дополняет его широчайшим набором функций, которые позволяют визуализировать данные множеством способов.

С помощью `GGally` вы сможете без труда создавать красивые диаграммы рассеяния, гистограммы, столбчатые диаграммы и многое другое. Отличительной чертой этого пакета является то, что он упрощает процесс создания сложных многомерных графиков, экономя ваше время и силы. Хотите исследовать корреляции, визуализировать регрессионные модели или строить кривые выживания? Используйте `GGally`.

Кроме того, с помощью этого пакета вы можете обнаружить скрытые взаимосвязи в ваших данных путем их визуального исследования. Интуитивно понятный синтаксис и дружественный интерфейс делают его доступным как для новичков, так и для опытных специалистов по работе с данными.

Помимо всего прочего, `GGally` упрощает процесс совместной работы. Используя визуализации, создаваемые с его помощью, вы сможете донести ваши выводы до широкой аудитории.

Теперь рассмотрим процесс использования `GGally` на простом примере:

```
if(!require(GGally)){install.packages("GGally")}
if(!require(TidyDensity)){install.packages("TidyDensity")}
library(GGally)
library(TidyDensity)
tidy_normal(.n = 200) |>
    ggpairs(columns = c("y","p","q","dx","dy"))
```

Посмотрим, что происходит в этом коде.

- Строка `if(!require(GGally)){install.packages("GGally")}` проверяет, установлен ли пакет `GGally` в среде R. Если нет, то он устанавливается с помощью команды `install.packages("GGally")`.

- То же самое происходит с пакетом TidyDensity во второй строке кода.
- Строка `library(GGally)`, убедившись в том, что пакет `Ggally` установлен, загружает его в текущий сеанс R.
- Строка `library(TidyDensity)` загружает пакет `TidyDensity`, который используется для создания графиков плотности, позволяющих представить распределение данных в аккуратной и организованной форме.
- Строка `tidy_normal(.n = 200)` генерирует набор, состоящий из 200 случайных точек данных. Предполагается, что их распределение является нормальным (то есть описывается колоколообразной кривой). Для создания этого набора данных используется функция `tidy_normal`.
- В строке `ggpairs(columns = c("y", "p", "q", "dx", "dy"))` функция `ggpairs` из пакета `GGally` применяется к сгенерированному ранее набору данных. Она создает матрицу диаграмм рассеяния, которые отражают зависимость каждой из переменных от всех остальных. Переменные `y`, `p`, `q`, `dx` и `dy` — это столбцы набора данных, которые используются для создания диаграмм рассеяния. Создаваемая матрица визуализирует взаимосвязи между этими столбцами, позволяя исследовать имеющиеся в данных закономерности и корреляции.

Полученный график показан на рис. 8.1.

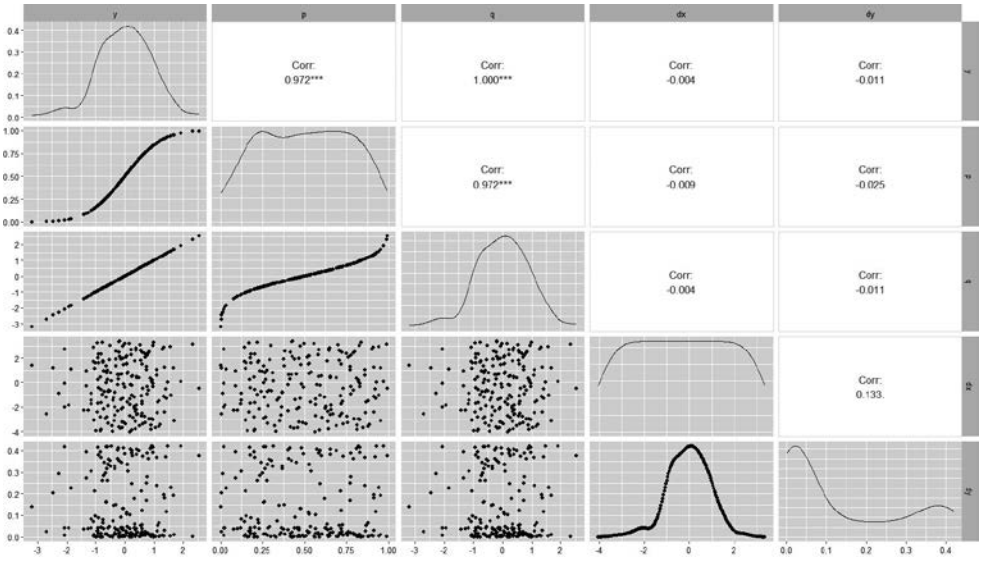


Рис. 8.1. Применение функций GGally к набору из 200 точек, полученному с помощью функции tidy_normal()

Здесь данные были сгенерированы случайным образом, поэтому вы, скорее всего, получите несколько иные результаты.

Теперь, когда мы разобрались с принципом использования пакета `GGally`, обсудим пакет `DataExplorer` и посмотрим, чем он отличается от других.

Использование R-пакета DataExplorer

R-пакет `DataExplorer` был разработан для упрощения большинства задач, связанных с управлением данными и их визуализацией в процессе EDA.

Пакет предоставляет множество функций для решения следующих задач разведочного анализа.

- **Сканирование и анализ переменных.** Пакет может автоматически сканировать и анализировать каждую переменную в наборе данных, определяя ее тип, распределение, выявляя выбросы и недостающие значения.
- **Визуализация данных.** Пакет предоставляет множество функций визуализации, которые помогают аналитикам понять взаимосвязи между переменными и выявить закономерности в данных. В частности, эти функции позволяют строить гистограммы, диаграммы рассеяния и размаха, тепловые карты и корреляционные матрицы.
- **Преобразование данных.** Пакет предоставляет функции для преобразования данных, например категориальных переменных в числовые, восстановления пропущенных значений и масштабирования числовых переменных.
- **Понимание общей структуры набора данных.** Пакет можно использовать для определения различных типов переменных в наборе данных, изучения их распределений и взаимосвязей.
- **Выявление выбросов и пропущенных значений.** Пакет может помочь аналитикам выявить такие проблемы, как выбросы и недостающие значения в данных, которые необходимо решить перед созданием прогностических моделей.
- **Выработка гипотез.** Пакет помогает аналитикам формулировать гипотезы о данных, выявляя закономерности и взаимосвязи между переменными.
- **Подготовка к моделированию.** Пакет помогает выполнить предварительную обработку данных, необходимую для дальнейшего моделирования и анализа.

Рассмотрим несколько примеров использования пакета `DataExplorer`.

```
install.packages("DataExplorer")
library(DataExplorer)
```

```
library(TidyDensity)
library(dplyr)

df <- tidy_normal(.n = 200)
df |>
  introduce() |>
  glimpse()
```

Сначала мы проверяем, установлен ли пакет `DataExplorer`. Если нет, то мы его устанавливаем. Затем мы загружаем пакет `DataExplorer`, а также пакеты `TidyDensity` и `dplyr`.

Далее мы создаем нормально распределенный набор данных, содержащий 200 наблюдений. Для этого используем функцию `tidy_normal()`, которая является весьма удобным способом создания таких наборов данных в среде R. Как правило, из всего вывода функции `tidy_normal()` берется только столбец y.

Получив набор данных, мы используем функцию `introduce()` из пакета `DataExplorer` для создания сводки, содержащей информацию о количестве наблюдений и типах переменных.

Наконец, воспользуемся функцией `glimpse()` из пакета `dplyr`, чтобы отобразить данные в транспонированном виде. Это позволяет выполнить быстрый обзор данных и убедиться в том, что они выглядят ожидаемым образом.

Таким образом, приведенный выше код является простым и быстрым способом исследования нормально распределенного набора данных в среде R. Он отлично подходит в качестве первичного этапа анализа как для студентов и новичков, так и для опытных специалистов по работе с данными. Результат его выполнения выглядит следующим образом:

```
> df |>
+   introduce() |>
+   glimpse()
Rows: 1
Columns: 9
$ rows                <int> 200
$ columns             <int> 7
$ discrete_columns    <int> 1
$ continuous_columns  <int> 6
$ all_missing_columns <int> 0
$ total_missing_values <int> 0
$ complete_rows       <int> 200
$ total_observations  <int> 1400
$ memory_usage        <dbl> 12344
```

Теперь рассмотрим функцию `plot_intro()` и результат ее применения к тем же данным, полученный с помощью простого вызова `df |> plot_intro()` (рис. 8.2).

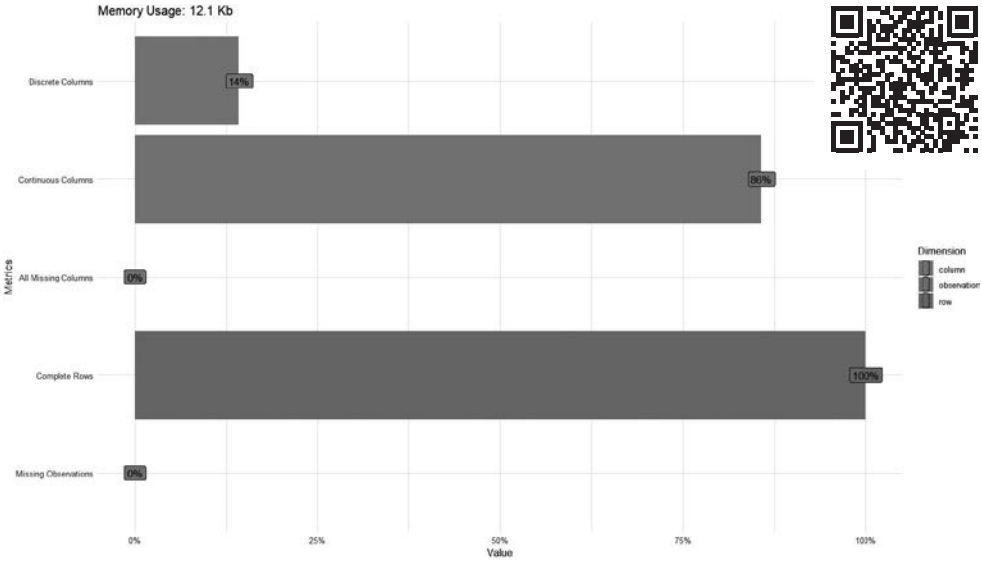


Рис. 8.2. Вывод функции plot_intro()

Наконец, посмотрим на результат применения функции `plot_qq()` (рис. 8.3).

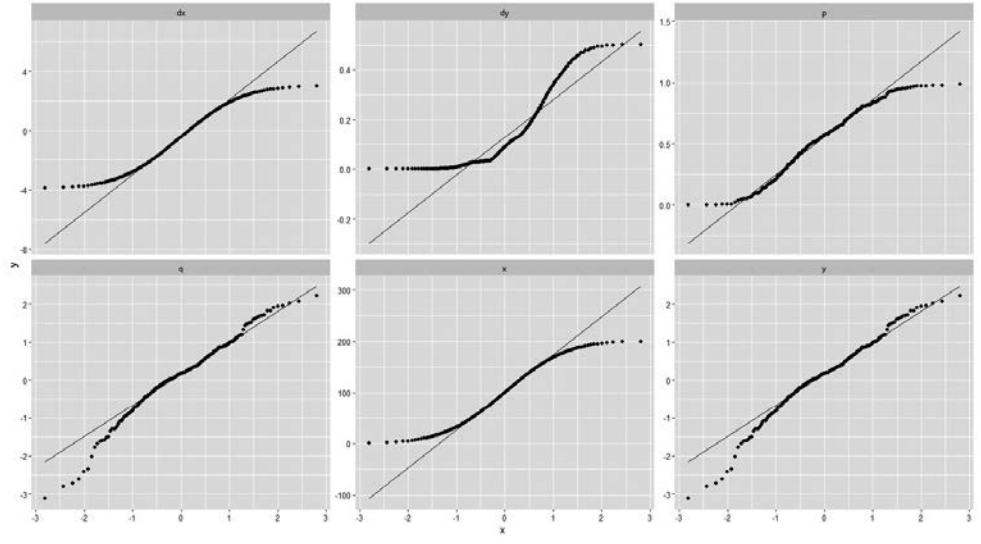


Рис. 8.3. Вывод функции plot_qq() со всеми данными

Теперь рассмотрим *квантильный график* (Q-Q, квантиль-квантиль) только с двумя переменными, столбцами *q* и *y*. Код для его создания выглядит так:

```
df[c("q","y")] |>
  plot_qq()
```

А сам график показан на рис. 8.4.

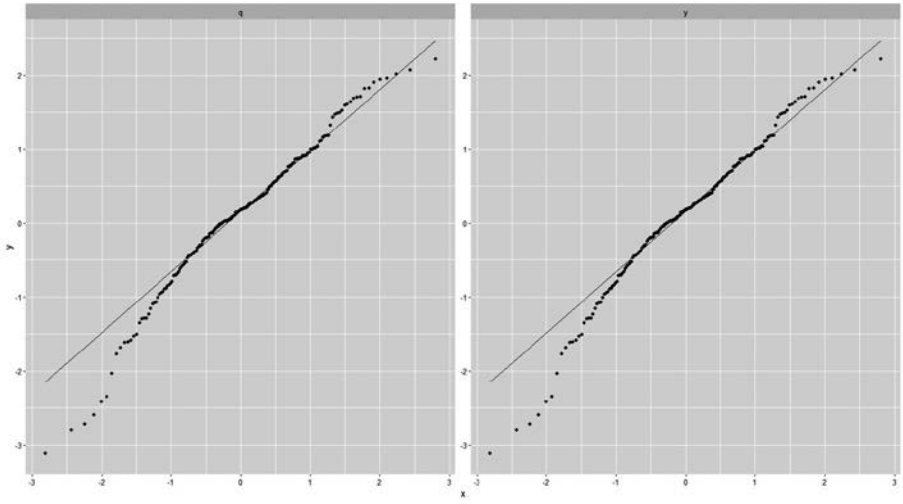


Рис. 8.4. Вывод функции plot_qq() только со столбцами y и q

Если вам не очень понятно, что такое квантильный график, то вы можете найти дополнительную информацию в Интернете. Еще один вопрос, который мы не обсуждали, связан с обработкой отсутствующих значений в среде R. Для сбора, очистки и выполнения других операций над данными используется множество функций, таких как `all()`, `any()`, `is.na()` и `na.omit()`. Их подробное описание вы также можете найти в Интернете.

Теперь, когда мы рассмотрели несколько функций из различных R-пакетов, пришло время обсудить их эквиваленты в Python.

Очистка данных Excel в среде Python

Очистка данных — важный этап. С его помощью вы можете:

- подготовить данные Excel к разведочному анализу в среде Python;
- выявить и устранить различные проблемы, связанные с качеством данных, которые могут повлиять на точность и надежность результатов анализа;
- убедиться, что ваши данные имеют правильный формат и не содержат ошибок.

Приступим к очистке данных в нашем примере. Начнем с генерации грязных данных.

```
import pandas as pd
import numpy as np

# Создание датафрейма с отсутствующими значениями, дубликатами
# и смешанными типами
data = {
    'ID': [1, 2, 3, 4, 5, 6],
    'Name': ['Alice', 'Bob', 'Charlie', 'Alice', 'Eva', 'Eva'],
    'Age': [25, np.nan, 30, 28, 22, 23],
    'Salary': ['$50,000', '$60,000', 'Missing', '$65,000', '$55,000',
    '$75,000']
}
df = pd.DataFrame(data)

# Внедрение пропущенных значений
df.loc[1, 'Age'] = np.nan
df.loc[3, 'Salary'] = np.nan

# Добавление дубликатов
df = pd.concat([df, df.iloc[1:3]],
ignore_index=True)

# Сохранение образца данных
# в текущем рабочем каталоге
df.to_excel('dirty_data.xlsx')
```

Созданный датафрейм выглядит следующим образом (рис. 8.5).

```
>>> df
   ID     Name   Age   Salary
0   1    Alice  25.0  $50,000
1   2      Bob   NaN  $60,000
2   3  Charlie  30.0  Missing
3   4    Alice  28.0      NaN
4   5      Eva  22.0  $55,000
5   6      Eva  23.0  $75,000
6   2      Bob   NaN  $60,000
7   3  Charlie  30.0  Missing
>>>
```

Рис. 8.5. Датафрейм с отсутствующими значениями, дубликатами и смешанными типами

Получив грязные данные, мы можем приступить к их очистке, которая предполагает обработку отсутствующих значений, удаление дубликатов и преобразование типов данных. Посмотрим, как это делается.

Обработка отсутствующих значений

Мы начинаем с выявления ячеек или столбцов с отсутствующими данными. В Python такие данные обычно бывают представлены значениями *NaN* (сокращение от Not a Number, то есть «не число») или *None*.

В зависимости от контекста вам может понадобиться заменить отсутствующие значения. К распространенным методам замены относятся:

- замена недостающих числовых значений средним арифметическим, медианой или модой (то есть наиболее часто встречающейся категорией). Учтите, что

это искусственно уменьшит значение стандартной ошибки, например, в случае выполнения регрессии. Помните об этом при очистке данных, которая выполняется в целях моделирования;

- замена отсутствующих категориальных данных модой;
- использование прямого или обратного заполнения для распространения предыдущего или следующего действительного значения;
- восстановление пропущенных значений путем интерполяции с учетом существующих в данных тенденций.

Python предлагает несколько решений, позволяющих сделать замену эффективно и статистически надежно, начиная от базовых методов библиотеки `pandas` и заканчивая специализированными пакетами, предлагающими гораздо более эффективные методы импутации (восстановления) данных, такие как `fancyimpute`. Однако импутацию никогда не следует применять вслепую, поскольку отсутствующие данные тоже могут содержать важную информацию, а их необдуманное восстановление способно исказить результаты анализа.

Важно различать три типа отсутствующих значений.

- *Полностью случайный пропуск* (missing completely at random, MCAR).
  - В данном случае отсутствие значений является совершенно случайным и никак не связано с наблюдаемыми или ненаблюдаемыми переменными.
  - Вероятность отсутствия точки данных одинакова для всех наблюдений.
  - Пропущенные и непропущенные значения не имеют систематических различий.
  - Пример: опрос, в ходе которого респонденты случайно пропустили некоторые вопросы.
- *Случайный пропуск* (missing at random, MAR).
  - Отсутствие значений зависит от наблюдаемых, а не от ненаблюдаемых данных.
  - Вероятность отсутствия точки данных связана с другими наблюдаемыми переменными в наборе данных.
  - После учета других наблюдаемых переменных пропуски данных оказываются случайными.
  - Пример: в ходе опроса мужчины могут менее охотно сообщать о своих доходах, чем женщины. В этом случае вероятность пропуска данных о доходах зависит от пола.
- *Неслучайный пропуск* (missing not at random, MNAR).
  - Отсутствие значений зависит от ненаблюдаемых данных или от самих пропущенных значений.
  - Вероятность отсутствия точки данных связана с отсутствующими значениями или другими ненаблюдаемыми переменными.

- Пропуски не оказываются случайными даже после учета наблюдаемых переменных.
- Пример: в ходе опроса люди с высоким доходом могут менее охотно сообщать о своем достатке.

Рассмотрим вариант, когда в строке или столбце отсутствует значительная часть данных. В таком случае вы можете вообще удалить эту строку или столбец. Однако будьте осторожны во избежание существенной потери информации.

```
import pandas as pd
import numpy as np

# Загрузка данных Excel в датафрейм pandas
df = pd.read_excel('dirty_data.xlsx', index_col=0)

# Обработка отсутствующих значений

# Определение отсутствующих значений
missing_values = df.isnull().sum()

# Замена отсутствующих значений средним арифметическим (для числовых столбцов)
df['Age'].fillna(df['Age'].mean(), inplace=True)

# Замена отсутствующих значений модой (для категориальных столбцов)
df['Salary'].fillna(df['Salary'].mode()[0], inplace=True)

# Прямое или обратное заполнение отсутствующих значений

# Данная строка является заполнителем, демонстрирующим одну из возможностей
# df['ColumnWithMissingValues'].fillna(method='ffill', inplace=True)

# Интерполяция недостающих значений исходя из тенденций в данных

# Данная строка является заполнителем, демонстрирующим одну из возможностей
# df['NumericColumn'].interpolate(method='linear', inplace=True)
# Удаление строк или столбцов с отсутствующими данными
df.dropna(axis=0, inplace=True) # Удаление строк с отсутствующими данными
df.dropna(axis=1, inplace=True) # Удаление столбцов с отсутствующими данными

df.to_excel('cleaned_data.xlsx')
```

Обработка дубликатов

Библиотеки Python, такие как `pandas`, предоставляют функции для обнаружения и обработки дубликатов. Для их выявления можно использовать метод `duplicated()`. После обнаружения дубликатов вы можете удалить их с помощью метода `drop_duplicates()`. Будьте осторожны при удалении, особенно в тех случаях, когда наличие повторяющихся строк является ожидаемым.

```
# Обработка дубликатов

# Обнаружение и отображение дублирующихся строк
duplicate_rows = df[df.duplicated()]
print("Duplicate Rows:")
print(duplicate_rows)

# Удаление дублирующихся строк
df.drop_duplicates(inplace=True)
```

Преобразование типов данных

Несмотря на высокую степень автоматизации этого процесса, чтение данных из Excel может привести к неправильному определению их типов. В Python типы данных часто назначаются автоматически при их чтении из Excel с помощью таких библиотек, как `pandas`. Однако по завершении необходимо убедиться в том, что каждому столбцу присвоен правильный тип данных (например, число, текст, дата и т. д.).

Последствия неправильного определения типов данных варьируются от потребления слишком большого объема памяти до реальных семантических ошибок. Например, если логические значения хранятся в виде чисел с плавающей запятой, то будут обрабатываться корректно, но занимать гораздо больше памяти (64 байта вместо одного бита), а попытка преобразовать строку в число с плавающей запятой может вообще привести к сбою в работе кода. Чтобы снизить этот риск, сначала определите типы загружаемых данных, а затем соответствующим образом преобразуйте их. Для преобразования типов данных в Python можно использовать метод `astype()` из библиотеки `pandas`. Например, чтобы преобразовать столбец в числовой тип данных, вы можете использовать `df['Column Name'] = df['Column Name'].astype(float)`.

Вот пример того, как это можно сделать:

```
# Контроль преобразования типов данных

# Проверка типов данных
print(df.dtypes)

# Преобразование столбца в другой тип данных (например, в число с плавающей
точкой)
df.loc[df['Salary']=='Missing', 'Salary'] = np.NaN
df.loc[:, 'Salary'] = df['Salary'].str.replace("$", "")
df.loc[:, 'Salary'] = df['Salary'].str.replace(",", "")
df['Salary'] = df['Salary'].astype(float)
print(df)

# Теперь, когда столбец Salary является числовым, мы можем заполнить
# недостающие значения средним арифметическим
df['Salary'].fillna(df['Salary'].mean(), inplace=True)
```

Проблемы, характерные для данных Excel

Среди таких проблем можно назвать следующие.

- **Объединенные ячейки.** Такие ячейки в файлах Excel могут привести к возникновению нерегулярностей в наборе данных при его импорте в среду Python. В связи с этим перед импортом рекомендуется отменить объединение ячеек. Если это невозможно, рассмотрите вариант предварительной обработки этих ячеек в среде Python.
- **Пустые ячейки смещают данные.** Пустые ячейки в файле Excel могут нарушить выравнивание данных при их импорте в Python. Чтобы решить эту проблему, вы можете выполнить следующее:
  - использовать параметр `pandas na_values` при чтении файлов Excel, чтобы указать значения, которые должны рассматриваться как отсутствующие (NaN) во время импорта;
  - вручную скорректировать файл Excel, чтобы удалить ненужные пустые ячейки перед импортом.

Решив эти проблемы, вы сделаете данные чистыми и пригодными к разведочному анализу, в результате которого получите более точные выводы.

Выполнение разведочного анализа данных с помощью Python

Далее мы рассмотрим конкретные методы исследования и визуализации, которые помогут вам лучше понять анализируемый набор данных. Начнем с подробного обсуждения методов изучения распределений и взаимосвязей между данными.

Сводная статистика

В первую очередь нужно получить сводную статистику по набору данных. Речь идет о вычислении основных показателей: для числовых признаков — среднее арифметическое, медиана, стандартное отклонение и процентили; для категориальных признаков — частоты и проценты.

Получение сводной статистики служит нескольким целям.

- **Получение представления о данных** — выявление типичных значений и разброса числовых данных. В случае с категориальными данными статистика показывает распределение категорий.

- **Выявление выбросов** — на них могут указывать необычно высокие или низкие средние значения.
- **Оценка качества данных** — выявление недостающих данных, которые могут быть представлены значениями NaN в наборе данных.
- **Формулирование первоначальных выводов** — получение общего представления о структуре данных, на основе которого могут приниматься решения, связанные с их дальнейшим анализом и визуализацией.

Далее более подробно рассмотрим показатели для числовых и категориальных признаков.

Числовые признаки

Применительно к ним имеет смысл вычислить следующие статистические показатели.

- **Среднее арифметическое.** Это мера центральной тенденции, которая вычисляется путем деления суммы всех числовых значений на их общее количество и дает представление о типичном значении данных.
- **Медиана.** Это значение, находящееся точно в середине набора данных, отсортированных в порядке возрастания или убывания. Это весьма надежный показатель центральной тенденции, на который в меньшей степени влияют экстремальные значения (выбросы).
- **Стандартное отклонение.** Определяет степень разброса или дисперсии данных. Более высокое стандартное отклонение соответствует большему разбросу данных.
- **Процентили.** Это конкретные значения, ниже которых находится определенный процент наблюдений. Например, 25-й процентиль (Q1) соответствует значению, ниже которого находятся 25 % точек данных. Процентили помогают определить характеристики распределения данных.

Категориальные признаки

Категориальные признаки предусматривают собственный набор статистических показателей.

- **Частота.** В случае с категориальными признаками имеет смысл рассчитать частоту встречаемости каждой уникальной категории в наборе данных.
- **Проценты.** Выражение частот в процентах от общего числа наблюдений дает представление об относительной распространенности каждой категории.

Получение сводной статистики в Python

Это можно сделать с помощью библиотек наподобие pandas. При работе с числовыми признаками можно использовать такие функции, как .mean(), .median(), .std() и .describe(). При работе с категориальными признаками можно применять .value_counts() и пользовательские функции для вычисления частот и процентов.

Ниже приведен пример вычисления сводной статистики с помощью Python и pandas для числовых и категориальных признаков. В демонстрационных целях создадим образец датафрейма:

```
import pandas as pd
import random

# Создание образца датафрейма
data = {
    'Age': [random.randint(18, 60) for _ in range(100)],
    'Gender': ['Male', 'Female'] * 50,
    'Income': [random.randint(20000, 100000) for _ in range(100)],
    'Region': ['North', 'South', 'East', 'West'] * 25
}
df = pd.DataFrame(data)

# Вычисление сводной статистики для числовых признаков
numerical_summary = df.describe()

# Вычисление частот и процентов для категориальных признаков
categorical_summary = df['Gender'].value_counts(normalize=True)

print("Summary Statistics for Numerical Features:")
print(numerical_summary)

print("\nFrequency Counts and Percentages for Categorical Features (Gender):")
print(categorical_summary)
```

В этом примере мы создали образец DataFrame, df, со столбцами Age, Gender, Income и Region. Мы использовали функцию df.describe() для расчета таких статистических показателей, как среднее арифметическое, стандартное отклонение, минимальное значение, максимальное значение, квартили и т. д., для числовых признаков (Age и Gender). Затем мы использовали фрагмент df['Gender'].value_counts(normalize=True) для вычисления частот и процентов для категориального признака Gender. Параметр normalize=True выражает результаты подсчетов в процентах. Если для него задано значение False, то команда value_counts() возвращает серию необработанных результатов подсчетов всех уникальных значений в столбце Gender.

Вы можете адаптировать этот код к своему набору данных Excel, загрузив данные в pandas DataFrame, а затем применив описанные выше функции вычисления сводной статистики.

Распределение данных

Прежде чем углубляться в изучение данных, необходимо определить их распределение, то есть выяснить, как организованы значения в наборе. Это поможет вам принимать более взвешенные решения на этапах статистического анализа, прогнозирования и моделирования. Рассмотрим несколько ключевых понятий.

- При **нормальном распределении** данные располагаются симметрично относительно среднего значения, образуя колоколообразную кривую. К таким данным очень легко применять многие статистические методы. Проверить нормальность распределения можно с помощью таких визуализаций, как гистограммы и квантильные графики, а также статистические тесты наподобие теста Шапиро — Уилка.
- **Перекос** определяет асимметрию распределения данных. Положительный перекос означает наличие длинного хвоста на правой стороне распределения, а отрицательный — наличие такого хвоста на левой. Выявить перекос очень важно, поскольку он может повлиять на достоверность результатов некоторых статистических тестов.
- **Эксцесс** — показатель «тяжелохвостости» или «пикообразности» распределения. Высокое значение коэффициента эксцесса указывает на тяжелые хвосты, а низкое — на легкие. Выявление эксцесса помогает выбирать подходящие статистические модели.
- Данные могут иметь и **другие распределения**, в частности экспоненциальное, логнормальное, равномерное или распределение Пуассона. Например, результаты подсчетов часто соответствуют распределению Пуассона, а количество осадков имеет логнормальное распределение.

Для анализа распределений данных с помощью Python вы можете использовать такие визуализации, как гистограммы, графики плотности ядра и диаграммы размаха. А статистические тесты и библиотеки наподобие SciPy помогут вам выявить и количественно оценить отклонения от нормальности.

Далее мы рассмотрим несколько фрагментов кода, который генерирует данные с логнормальным распределением, выполняет тест Шапиро — Уилка, строит квантильный график с нормальным и логнормальным распределением, а также рассчитывает такие показатели, как перекос и эксцесс.

Сначала создадим образцы данных:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
import statsmodels.api as sm
```

```
# Генерирование образцов данных с логнормальным распределением
np.random.seed(0)
data = np.random.lognormal(mean=0, sigma=1, size=1000)

# Создание датафрейма pandas
df = pd.DataFrame({'Data': data})

# Создание гистограммы для получения наглядного представления о распределении
into the distribution:

# Создание гистограммы на основе данных
plt.hist(data, bins=30, color='skyblue', edgecolor='black')
plt.title('Histogram of Data')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()
```

На полученной гистограмме виден явный перекос, что говорит скорее о логнормальном, нежели о нормальном распределении данных (что неудивительно, если учесть способ их генерации) (рис. 8.6).

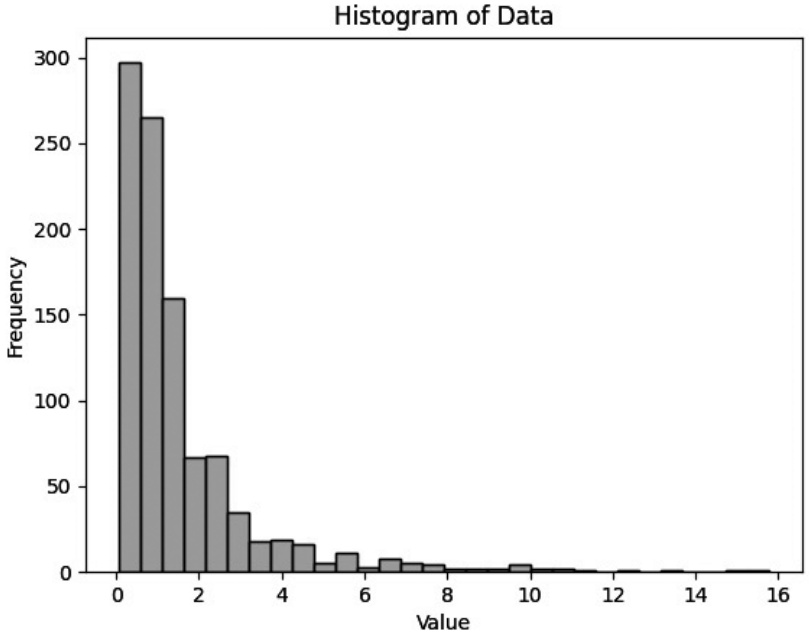


Рис. 8.6. Гистограмма, отражающая распределение данных

Далее мы можем провести базовый статистический анализ, чтобы подтвердить наши предположения относительно распределения данных. Мы начнем с выполнения теста Шапиро — Уилка на нормальность, затем создадим квантильный график

с нормальным распределением и, наконец, квантильный график с логнормальным распределением.

```
# Выполнение теста Шапиро – Уилка на нормальность
shapiro_stat, shapiro_p = stats.shapiro(data)
is_normal = shapiro_p > 0.05 # Проверка, нормально ли распределение данных
print(f'Shapiro-Wilk p-value: {shapiro_p}')
print(f'Is data normally distributed? {is_normal}')

# Создание квантильного графика с нормальным распределением
sm.qqplot(data, line='s', color='skyblue')
plt.title('Q-Q Plot (Normal)')
plt.xlabel('Theoretical Quantiles')
plt.ylabel('Sample Quantiles')
plt.show()

# Создание квантильного графика с логнормальным распределением
log_data = np.log(data)
sm.qqplot(log_data, line='s', color='skyblue')
plt.title('Q-Q Plot (Lognormal)')
plt.xlabel('Theoretical Quantiles')
plt.ylabel('Sample Quantiles')
plt.show()
```

Первый квантильный график показывает, что нормальное распределение очень плохо соотносится с данными (рис. 8.7).

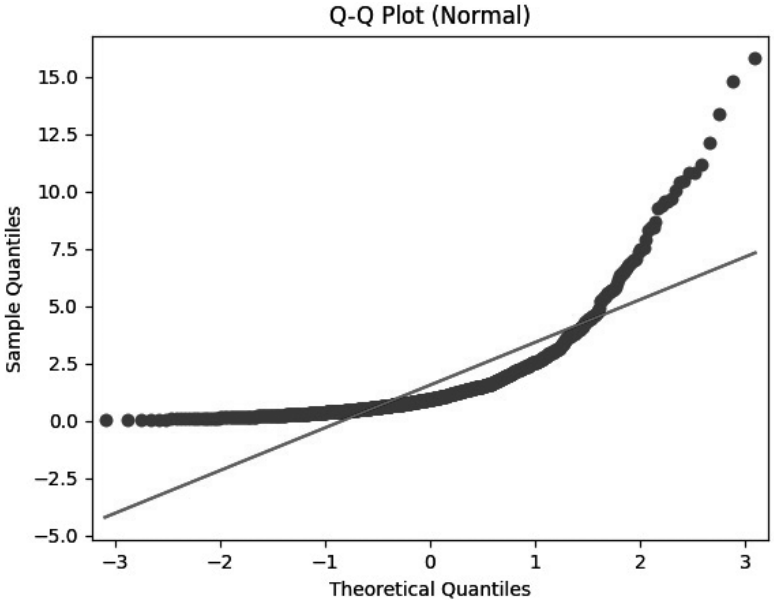


Рис. 8.7. Квантильный график для нормального распределения

Следующий квантильный график для логнормального распределения, напротив, демонстрирует почти идеальное соответствие (рис. 8.8).

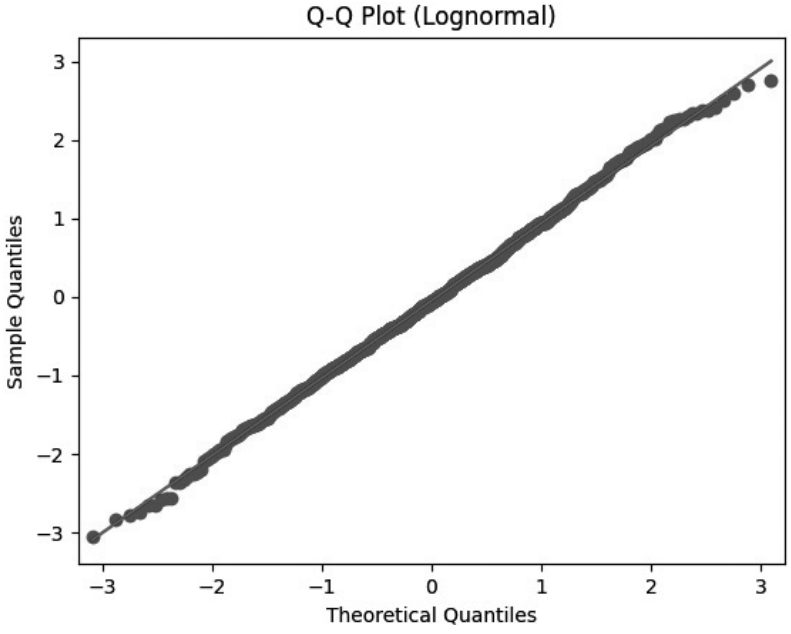


Рис. 8.8. Квантильный график для логнормального распределения

Наконец, вычисляем показатели перекоса и эксцесса, чтобы убедиться в том, что все делаем правильно, и выводим результаты на экран.

```
# Вычисление показателей перекоса и эксцесса
skewness = stats.skew(data)
kurtosis = stats.kurtosis(data)

print(f'Skewness: {skewness}')
print(f'Kurtosis: {kurtosis}')
```

На основании гистограммы, результатов теста Шапиро — Уилка, отвергающих гипотезу о нормальном распределении, двух квантильных графиков, а также показателей перекоса и эксцесса мы можем с уверенностью сказать, что данные действительно подчиняются логнормальному распределению.

Теперь пришло время переключиться с анализа одной переменной на изучение взаимосвязей между несколькими переменными.

Взаимосвязи между переменными

Взаимосвязи между переменными можно исследовать с помощью диаграмм рассеяния и корреляционных матриц. Выявление сильных корреляций или зависимостей между признаками может помочь в их отборе и конструировании на более поздних этапах анализа. В этом подразделе мы рассмотрим методы исследования таких взаимосвязей с помощью Python.

Мы будем использовать следующие библиотеки и сгенерированный набор данных:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import ppscore as pps

# Генерация тестовых данных с тремя переменными
np.random.seed(0)
data = {
    'Feature1': np.random.randn(100),
    'Feature2': np.random.randn(100) * 2,
}

# Создание линейной целевой переменной (Target) на основе Feature1
# и нелинейной функции Feature2
data['Target'] = data['Feature1'] * 2 + np.sin(data['Feature2']) + \
    np.random.randn(100) * 0.5

# Создание датафрейма
df = pd.DataFrame(data)
```

Вы можете адаптировать этот код для анализа взаимосвязей между переменными в своих данных Excel, используя методы, описанные в предыдущих главах.

Тепловая карта корреляции

Один из наиболее распространенных способов оценки взаимосвязей — создание тепловой карты корреляции. Эта карта дает наглядное представление о зависимостях между числовыми переменными в вашем наборе данных. Используя такие библиотеки, как `seaborn`, вы можете создать информативную тепловую карту, на которой степень корреляции обозначена цветом, что позволяет легко определить сильные и слабые взаимосвязи между переменными.

Посмотрим, как это сделать:

```
# Вычисление и создание тепловой карты корреляции
corr_matrix = df.corr()
```

```
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", \
            linewidths=.5)
plt.title('Correlation Heatmap')
plt.show()
```

Код создает тепловую карту корреляции, визуализирующую взаимосвязи между переменными. Вот как она выглядит (рис. 8.9).

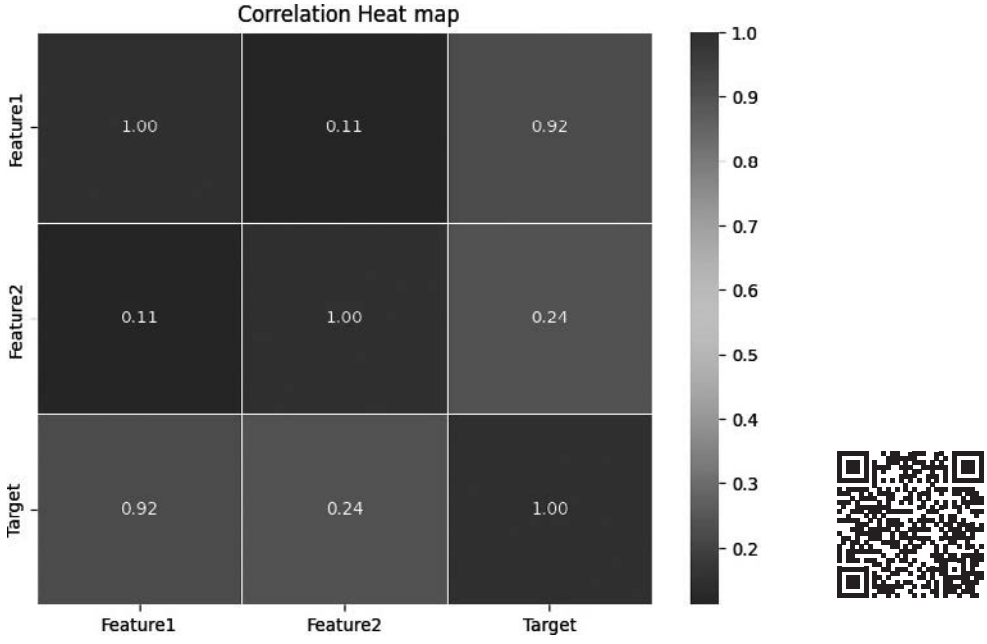


Рис. 8.9. Тепловая карта корреляции

Оценка предсказательной силы

Помимо традиционных показателей корреляции, существует так называемая оценка предсказательной силы (predictive power score, PPS), которая дает представление о нелинейных взаимосвязях между переменными. PPS оценивает, насколько хорошо одна переменная может предсказывать другую.

Далее мы продемонстрируем способ вычисления и визуализации матриц PPS с помощью библиотеки `ppscore`, которые позволят вам оценить прогностический потенциал ваших данных.

```
# Вычисление показателя PPS
plt.figure(figsize=(8, 6))
matrix_df = pps.matrix(df)[['x', 'y', 'ppscore']].pivot(columns='x', \
    index='y', values='ppscore')
```

```
sns.heatmap(matrix_df, vmin=0, vmax=1, cmap="Blues", linewidths=0.5, annot=True)
plt.title("Predictive Power Score (PPS) Heatmap")
plt.show()

# Дополнительные сведения
correlation_target = df['Feature1'].corr(df['Target'])
pps_target = pps.score(df, 'Feature1', 'Target') ['ppscore']

print(f'Correlation between Feature1 and Target: {correlation_target:.2f}')
print(f'Predictive Power Score (PPS) between Feature1 and Target: {pps_target:.2f}')
```

В этом фрагменте кода мы вычисляем матрицу PPS, которая измеряет предсказательную силу каждого признака в отношении целевой переменной (`Target`). В конце мы вычисляем PPS и корреляцию между признаком `Feature1` и целевой переменной. Итоговая тепловая карта PPS (рис. 8.10) помогает глубже понять взаимосвязи между переменными.

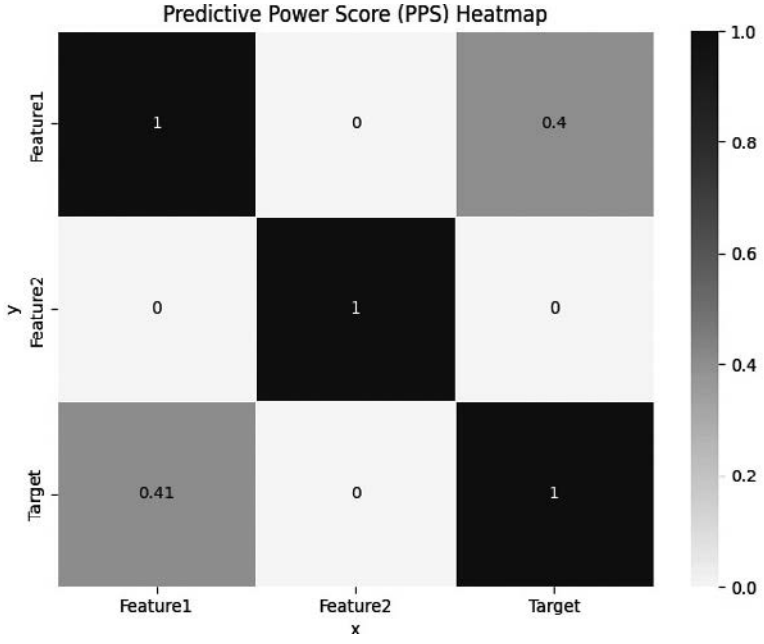


Рис. 8.10. Тепловая карта PPS

Диаграммы рассеяния

Диаграммы рассеяния представляют собой еще один ценный инструмент для визуализации взаимосвязей. Они позволяют изучить зависимость между двумя числовыми переменными путем нанесения точек данных на двумерную плоскость. Способ создания диаграмм рассеяния был описан в главе 6.

Визуализация ключевых атрибутов

Визуализация — мощный инструмент, позволяющий получить представление о данных. Вы можете создавать различные графики и диаграммы для визуализации ключевых атрибутов набора данных. Для исследования числовых данных можно использовать гистограммы, диаграммы размаха и рассеяния, благодаря которым можно выявить выбросы и потенциальные корреляции, а также определить тип распределения. Категориальные данные можно исследовать с помощью столбчатых и круговых диаграмм, отображающих частотные распределения и пропорции. Эти визуализации позволяют получить сведения, которые могут отсутствовать в сводной статистике. Методы создания этих визуализаций были рассмотрены в главе 6.

Резюме

В этой главе мы обсудили два важнейших процесса: очистку данных и их разведочный анализ с помощью R и Python, сделав акцент на данных Excel.

Очистка данных — фундаментальный этап их исследования. Мы рассмотрели способы обработки отсутствующих значений, в том числе восстановление, удаление и интерполяцию. Кроме того, мы уделили внимание работе с дубликатами, так как данные Excel часто приходят из разных источников, а значит, нередко избыточны. Среди прочего, мы указали на необходимость правильного назначения типов данных для предотвращения искажения результатов анализа.

Разговор о разведочном анализе данных с помощью Python мы начали с обсуждения сводной статистики. При изучении числовых признаков такие показатели, как среднее арифметическое, медиана, стандартное отклонение и процентили, дают первоначальное представление о центральных тенденциях и изменчивости данных. Затем мы обсудили распределение данных, которое влияет на принятие решений, касающихся последующего анализа и моделирования. Наконец, мы поговорили о том, как выявлять взаимосвязи между переменными, используя диаграммы рассеяния и корреляционные матрицы, помогающие определять корреляции и зависимости между признаками. Освоив все эти приемы, вы сможете обеспечить качество данных и с их помощью принимать обоснованные решения.

9 Статистический анализ: линейная и логистическая регрессия

В этой главе мы подробно обсудим реализацию таких важнейших статистических методов анализа, как линейная и логистическая регрессия, с помощью фреймворка `tidymodels`, базового R и Python. Каким бы ни был ваш опыт работы с данными, изложенный далее материал поможет вам получить глубокое понимание *линейной* и *логистической регрессии*, а также способов их реализации в среде R и Python. Однако, несмотря на принципиальную возможность выполнения линейной и логистической регрессии, проблема заключается в том, что линейную можно провести только на одной серии несгруппированных данных, а выполнение логистической может оказаться весьма сложным и потребовать добавления внешних надстроек. Кроме того, ее можно применять только к несгруппированным или невложенным данным. В R и Python таких ограничений нет.

В этой главе мы рассмотрим следующие темы:

- линейная регрессия;
- логистическая регрессия;
- инструменты для реализации линейной и логистической регрессии;
- выполнение линейной регрессии в среде R;
- выполнение логистической регрессии в среде R;
- выполнение линейной регрессии в среде Python с использованием данных Excel;
- выполнение логистической регрессии в среде Python с использованием данных Excel.

Технические требования

Файлы с кодом из примеров этой главы доступны по адресу https://github.com/PacktPublishing/Extending-Excel-with-Python-and-R/tree/main/Chapter9.

Для изучения материала этой главы вам понадобятся следующие R-пакеты:

- `readxl 1.4.3`;
- `performance 0.10.8`;
- `tidymodels 1.1.1`;
- `purrr 1.0.2`.

Сначала мы познакомимся с понятиями линейной и логистической регрессии, а затем перейдем к способам их выполнения в среде R и Python.

Линейная регрессия

Линейная регрессия — фундаментальный метод статистического анализа, используемый для моделирования взаимосвязи между зависимой переменной (обычно обозначаемой как Y) и одной или несколькими независимыми переменными (часто обозначаемыми как X). Цель заключается в нахождении такого линейного уравнения, которое наиболее точно описывает влияние изменения независимых переменных на зависимую переменную. Многим из вас данный метод, вероятно, известен как *метод наименьших квадратов* (ordinary least squares, OLS).

Проще говоря, линейная регрессия помогает предсказать непрерывный числовой результат на основе одного или нескольких входных признаков. Чтобы это сработало, множество предположений должны оказаться истинными. Дополнительную информацию об этом вы можете найти в Интернете. В текущей главе мы познакомимся как с простой (одна независимая переменная), так и с множественной (несколько независимых переменных) линейной регрессией.

Логистическая регрессия

Логистическая регрессия — еще один важнейший метод статистического анализа, который в основном используется для решения задач двоичной (бинарной) классификации. Вместо того чтобы предсказывать непрерывные результаты, логистическая регрессия прогнозирует вероятность наступления события, которая обычно выражается в виде ответа «да» или «нет». Этот метод особенно полезен в сценариях, где нам нужно спрогнозировать, например, уйдет клиент или нет, является ли письмо спамом или нет и т. д. Логистическая регрессия моделирует взаимосвязь между независимыми переменными и логарифмом отношения шансов бинарного исхода.

Инструменты для реализации линейной и логистической регрессии

Мы рассмотрим два подхода к реализации линейной и логистической регрессии в среде R. Сначала мы используем базовый R, с помощью которого вы сможете понять основные концепции и функции. Затем обсудим фреймворк `tidymodels`, который обеспечивает современный подход к моделированию и машинному обучению на языке R, а также последовательный и эффективный способ создания, настройки и оценки моделей, что делает его ценным инструментом для специалистов по анализу данных.

В разделах, посвященных Python, вы познакомитесь с такими известными библиотеками, как `sklearn` и `statsmodels`. Библиотека `sklearn`, или `scikit-learn`, предоставляет множество простых и эффективных инструментов для прогностического анализа данных, которые доступны всем и могут использоваться в различных контекстах. Библиотека `statsmodels` в большей степени ориентирована на статистические модели и проверку гипотез. Благодаря комбинации этих библиотек Python вы сможете реализовывать линейную и логистическую регрессию, решая задачи, связанные как с машинным обучением, так и со статистическим анализом.

В этой главе вы найдете пошаговые инструкции, примеры кода и практические рекомендации. Изучив все это, вы сможете применять методы линейной и логистической регрессии в рамках собственных проектов по анализу данных.

Выполнение линейной регрессии в среде R

В этом разделе мы выполним линейную регрессию в среде R с помощью как базового функционала этого языка, так и фреймворка `tidymodels`. Вы узнаете о том, как применить этот метод к набору данных, содержащему разные группы. Если вы научитесь выполнять линейную регрессию на сгруппированных данных, то легко сможете выполнить ее и на одной группе, поскольку в этом случае вам не придется группировать данные и выполнять операции над их отдельными группами.

Выполнение линейной регрессии с помощью базового функционала R

В первом примере мы используем базовую функцию R `lm()`, чтобы проанализировать набор данных `iris` с помощью метода линейной регрессии.

Мы разобьем код на фрагменты и обсудим происходящее на каждом этапе. Сначала используем команду `library` для загрузки необходимых пакетов в нашу среду разработки:

```
library(readxl)
```

Здесь мы загружаем библиотеку `readxl`, которая обычно используется для чтения данных из файлов Excel. Указанный путь (`path`) предполагает, что у вас есть папка chapter1, в которой содержится файл с данными, имеющий название `iris_data.xlsx`:

```
df <- read_xlsx(
  path = "chapter1/iris_data.xlsx",
  sheet = "iris"
)
head(df)
```

Здесь мы считываем данные из файла Excel `iris_data.xlsx`, расположенного в папке chapter1. А если точнее, считываем лист `iris` из данного файла. Для этого используется функция `read_xlsx`. Полученные данные хранятся в переменной `df`. Функция `head(df)` выводит первые несколько строк этого датафрейма (`df`), позволяя нам увидеть, как он выглядит:

```
iris_split <- split(df, df$species)
```

Этот код разбивает набор данных `df` на несколько подмножеств на основе уникальных значений в столбце `species`. В результате получается список датафреймов, каждый из которых содержит только строки, соответствующие определенному виду ирисов.

Теперь нам нужно определить зависимые и независимые переменные, а также объект `formula`:

```
dependent_variable <- "petal_length"
independent_variables <- c("petal_width", "sepal_length", "sepal_width")
f_x <- formula(
  paste(
        dependent_variable,
        "~",
        paste(independent_variables, collapse = " + ")
  )
)
```

Здесь мы определяем переменные, необходимые для выполнения линейной регрессии. Зависимая переменная (`dependent_variable`), то есть та, которую мы хотим предсказать, — `petal_length`. Независимые переменные, то есть те, которые мы будем использовать для предсказания зависимой переменной, — `petal_width`, `sepal_length` и `sepal_width`.

Затем код создает формулу `f_x`, которая представляет модель линейной регрессии. По сути, она говорит о том, что мы хотим предсказать длину лепестка (`petal_length`) на основе других перечисленных переменных, разделенных знаком плюс.

```
perform_linear_regression <- function(data) {
  lm_model <- lm(f_x, data = data)
  return(lm_model)
}
```

В этой части мы определяем пользовательскую функцию R `perform_linear_regression`. Она принимает один аргумент `data`, который представляет собой датафрейм. Внутри нее мы используем функцию `lm`, позволяющую выполнять линейную регрессию с помощью определенной ранее формулы `f_x` и предоставленного датафрейма. Полученная линейная модель сохраняется в `lm_model` и возвращается в качестве вывода функции:

```
results <- lapply(iris_split, perform_linear_regression)
```

Здесь мы применяем функцию `perform_linear_regression` к каждому подмножеству набора данных `iris` с помощью функции `lapply`. Это означает, что мы выполняем линейную регрессию отдельно для каждого вида ирисов и сохраняем результаты в списке `results`:

```
lapply(results, summary)
```

Здесь снова используется `lapply`, но на сей раз мы применяем функцию `summary` к каждой модели линейной регрессии в списке `results`. Эта функция предоставляет статистическую информацию о модели линейной регрессии, в частности коэффициенты и значения R-квадрата:

```
par(mfrow = c(2,2))
lapply(results, plot)
par(mfrow = c(1, 1))
```

Эти строки кода используются для создания набора из четырех графиков, визуализирующих производительность модели. Сначала мы организуем эти графики в виде сетки 2 × 2 с помощью фрагмента `par(mfrow = c(2,2))`. Затем мы используем функцию `lapply` для создания графиков каждой модели линейной регрессии в списке `results`. Наконец, мы возвращаем область создания графиков в исходное состояние с помощью фрагмента `par(mfrow = c(1, 1))`:

```
lm_models <- lapply(
    iris_split,
    function(df) lm(f_x, data = df)
)
```

Эта часть выполняет тот же линейный регрессионный анализ, что и предыдущая, только объединяет этапы создания и обобщения линейных моделей в более сжатую форму с помощью анонимных функций. Сначала к каждому подмножеству видов в `iris_split` применяется функция `lm`, а созданный список линейных моделей сохраняется в `lm_models`. Затем мы получаем сводки по каждой из этих линейных моделей с помощью функции `lapply`.

Итак, представленный выше код на языке R считывает данные об ирисах из файла Excel, выполняет линейную регрессию для каждого из видов, обобщает результаты и создает визуализации для оценки эффективности модели. Он позволяет

получить подробный анализ взаимосвязи между зависимой переменной (`petal_length`) и независимыми переменными (`petal_width`, `sepal_length` и `sepal_width`) для каждого вида ирисов.

Выполнение линейной регрессии с помощью tidymodels и purrr

Теперь, когда мы разобрались с тем, как выполнить простую линейную регрессию на наборе данных `iris` в среде R, сделаем то же самое с помощью фреймворка `tidymodels`:

```
f_x <- formula(paste("petal_width", "~", "petal_length + sepal_width + \
sepal_length"))
```

В этом блоке определяется формула для модели линейной регрессии. Функция `formula()` принимает два аргумента: переменную отклика и предикторные переменные. Переменная отклика — это переменная, которую мы хотим предсказать, а предикторные переменные — это переменные, которые, по нашему мнению, могут в этом помочь. В данном случае переменной отклика является `petal_width`, а предикторными переменными — `petal_length`, `sepal_width` и `sepal_length`:

```
library(dplyr)
library(tidyr)
library(purrr)
library(tidymodels)
nested_lm <- df |>
  nest(data = -species) |>
  mutate(split = map(data, ~ initial_split(., prop = 8/10)),
         train = map(split, ~ training(.)),
         test = map(split, ~ testing(.)),
         fit = map(train, ~ lm(f_x, data = .)),
         pred = map2(.x = fit, .y = test, ~ predict(object = .x, newdata = .y)))
```

Этот блок создает вложенную модель линейной регрессии с помощью функции `nest()` из пакета `tidyr`. Функция группирует данные по заданной переменной, которой в данном случае является `species`.

Для каждой группы функция `nest()` создает список, который содержит данные, относящиеся к этой группе. Затем функция `mutate()` применяется для добавления новых столбцов во вложенный датафрейм.

Мы используем функцию `split()`, чтобы случайным образом разбить данные в каждой группе на обучающий и тестовый наборы. Затем мы выбираем эти наборы с помощью функций `training()` и `testing()` соответственно. Используя функции `map()` и `map2()`, мы можем выполнить итерацию по вектору или списку либо по двум векторам или спискам и применить к ним функцию.

Для подгонки линейной регрессионной модели к обучающим данным в каждой группе мы используем функцию `lm()`. Затем запускаем функцию `predict()`, чтобы спрогнозировать значения переменной отклика для тестовых данных в каждой группе с помощью подогнанной модели линейной регрессии:

```
nested_lm |>
  select(species, pred) |>
  unnest(pred)
```

Этот блок кода выбирает столбцы `species` и `pred` из вложенного датафрейма и применяет к столбцу `pred` функцию `unnest()`, которая преобразует вложенный датафрейм в обычный датафрейм, содержащий одну строку для каждого наблюдения.

Итоговый датафрейм представляет собой вложенную модель линейной регрессии, предусматривающую по одной подогнанной линейной регрессионной модели для каждого вида.

Рассмотрим пример. Мы будем использовать созданную ранее формулу `f_x` вместе с переменной `df`, которую создали в самом начале. В коде ниже показано, как с помощью вложенной модели линейной регрессии можно прогнозировать ширину лепестка нового цветка ириса:

```
library(dplyr)
library(tidyr)
library(purrr)
library(tidymodels)

# Создание вложенной модели линейной регрессии
nested_lm <- df |>
  nest(data = -species) |>
  mutate(split = map(data, ~ initial_split(., prop = 8/10)),
         train = map(split, ~ training(.)),
         test = map(split, ~ testing(.)),
         fit = map(train, ~ lm(f_x, data = .)),
         pred = map2(.x = fit, .y = test, ~ predict(object = .x, newdata = .y)))

# Предсказание ширины лепестка для нового цветка ириса
new_iris <- data.frame(sepal_length = 5.2, sepal_width = 2.7, petal_length = 3.5)

# Предсказание ширины лепестка
predicted_petal_width <- predict(nested_lm[[1]]$fit, newdata = new_iris))

# Вывод предсказанной ширины лепестка на экран
print(predicted_petal_width)
```

Результат выполнения этого кода:

```
1.45
```

Прогнозируемая ширина лепестка составляет 1,45 см.

Выполнение логистической регрессии в среде R

Как и в случае с линейной регрессией, в этом разделе мы выполним логистическую регрессию с помощью базового R и фреймворка `tidymodels`. Нам предстоит решить простую задачу бинарной классификации, позволяющую предсказать выживаемость пассажиров «Титаника».

Выполнение логистической регрессии с помощью базового функционала R

Начнем с реализации логистической регрессии с помощью базового функционала R, чтобы смоделировать ответ `Survived` (выжил) на основе набора данных `Titanic`.

Ниже приведены код, выполняющий это моделирование, а также пояснения к нему.

```
library(tidyverse)
df <- Titanic |>
       as.data.frame() |>
       uncount(Freq)
```

Этот блок кода начинает с загрузки библиотеки `tidyverse`, которая содержит различные инструменты для работы с данными и их визуализации. Затем он создает датафрейм под названием `df`. Для этого он берет набор данных `Titanic` (предполагается, что он доступен в вашей среде) и выполняет над ним три операции с помощью оператора `|>`. Функция `as.data.frame()` преобразует набор данных в датафрейм, а `uncount(Freq)` создает дубликаты строк в наборе данных, количество которых соответствует значениям в столбце `Freq`. Это довольно часто делается для расширения обобщенных данных.

```
set.seed(123)
train_index <- sample(nrow(df), floor(nrow(df) * 0.8), replace = FALSE)
train <- df[train_index, ]
test <- df[-train_index, ]
```

Этот фрагмент разбивает данные на обучающий и тестовый наборы, что является обычной практикой в процессе машинного обучения.

- Строка `set.seed(123)` задает зерно (значение инициализации) генератора случайных чисел для обеспечения воспроизводимости, то есть для того, чтобы результаты случайных операций всегда оказывались одинаковыми.
- Строка `sample(nrow(df), floor(nrow(df) * 0.8), replace = FALSE)` случайным образом выбирает 80 % строк датафрейма `df` (для обучающего набора) без замены и сохраняет их индексы в `train_index`.

- Строка `train <- df[train_index, ]` создает обучающий набор, выбирая строки из `df` с помощью индексов `train_index`.
- Строка `test <- df[-train_index, ]` создает тестовый набор, выбирая строки из `df`, которые не относятся к обучающему набору.

Затем мы создаем модель:

```
model <- glm(Survived ~ Sex + Age + Class, data = train, family = "binomial")
```

Теперь обсудим код, создающий модель.

- Этот блок кода обучает модель логистической регрессии с помощью функции `glm`.
- Модель учится предсказывать переменную `Survived` на основе переменных `Sex`, `Age` и `Class`, присутствующих в обучающих данных. Переменная `Age` в данном случае является дискретной.
- Аргумент `family = "binomial"` указывает на то, что это задача бинарной классификации, результатом которой является либо `Yes` (Да), либо `No` (Нет). Информация о том, как выбрать подходящее семейство, доступна по адресу https://stats.stackexchange.com/a/303592/35448.

Теперь зададим предсказания модели и переменную отклика:

```
predictions <- predict(model, newdata = test, type = "response")
pred_resp <- ifelse(predictions <= 0.5, "No", "Yes")
```

Посмотрим, какие действия выполняются в этом коде.

- В данном случае мы используем обученную модель для получения прогнозов на тестовом наборе.
- Фрагмент `predict(model, newdata = test, type = "response")` вычисляет вероятность выживания для каждого пассажира в тестовом наборе.
- Фрагмент `ifelse(predictions <= 0.5, "No", "Yes")` преобразует эти вероятности в бинарные предсказания: `"No"`, если вероятность меньше или равна `0.5`, и `"Yes"`, если вероятность превышает `0.5`. Так выглядит обычный подход, однако вам может потребоваться скорректировать его с учетом особенностей вашего проекта.

Теперь рассмотрим переменную `accuracy`:

```
accuracy <- mean(pred_resp == test$Survived)
```

Эта строка кода выполняет следующие действия:

- определяет точность предсказаний модели, сравнивая `pred_resp` (прогнозы модели) с фактическим значением выживаемости в тестовом наборе (`test$Survived`);

- вычисляет среднее арифметическое результирующих логических значений, где `TRUE` означает правильное предсказание, а `FALSE` — неправильное.

Рассмотрим оставшуюся часть кода:

```
print(accuracy)
table(pred_resp, test$Survived)
```

В результате его выполнения мы получим:

- точность модели на тестовом наборе;
- матрицу ошибок, которая показывает количество правильных и неправильных предсказаний. Более подробная информация о матрицах ошибок доступна здесь: https://www.v7labs.com/blog/confusion-matrix-guide.

Таким образом, приведенный выше код загружает набор данных, разбивает его на обучающий и тестовый наборы, обучает модель логистической регрессии для прогнозирования выживаемости, оценивает точность модели и отображает результаты. Это простой пример решения задачи бинарной классификации с помощью методов машинного обучения.

Выполнение логистической регрессии с помощью tidymodels

В этом подразделе мы используем фреймворк `tidymodels` для выполнения логистической регрессии на наборе данных `Titanic`. Мы уже выполняли ее с помощью базового функционала R, поэтому сразу перейдем к делу.

```
library(tidymodels)
library(healthyR.ai)
```

Код загружает две библиотеки, которые понадобятся для проведения анализа: `tidymodels` и `healthyR.ai`. Библиотека `tidymodels` предоставляет общий интерфейс для многих алгоритмов машинного обучения, а `healthyR.ai` — набор инструментов для оценки производительности моделей.

```
df <- Titanic |>
    as_tibble() |>
    uncount(n) |>
    mutate(across(where(is.character), as.factor))
```

Этот код преобразует набор данных `Titanic` в `tibble` — структуру данных, совместимую с `tidymodels`. Кроме того, он применяет функцию `uncount()` к столбцу `n`, который содержит количество вхождений каждой строки в набор данных. Наконец, он преобразует все символьные переменные в наборе данных в факторы.

```
# Настройка зерна генератора случайных чисел для воспроизводимости результатов
set.seed(123)

# Разбиение данных на обучающий и тестовый наборы
split <- initial_split(df, prop = 0.8)
train <- training(split)
test <- testing(split)
```

Этот код разбивает набор данных `df` на обучающий и тестовый наборы. Обучающий служит для обучения модели, а тестовый — для оценки ее производительности на данных, с которыми она ранее не сталкивалась. Для разбиения используется функция `initial_split()` библиотеки `tidymodels`. Аргумент `prop` задает долю данных, которые будут использованы для обучения. В этом примере мы взяли 80 % данных для обучения и 20 % — для тестирования:

```
# Создание рецепта предварительной обработки данных
recipe <- recipe(Survived ~ Sex + Age + Class, data = train)

# Указание логистической регрессии в качестве модели
log_reg <- logistic_reg() |> set_engine("glm", family = "binomial")

# Объединение рецепта и модели в рабочий процесс
workflow <- workflow() %>% add_recipe(recipe) %>% add_model(log_reg)

# Обучение модели логистической регрессии
fit <- fit(workflow, data = train)
```

Этот код обучает модель логистической регрессии для прогнозирования выживаемости пассажиров «Титаника». Функция `recipe()` из `tidymodels` используется для предварительной обработки данных. Функция `logistic_reg()` из `tidymodels` применяется для выбора модели логистической регрессии. Функция `workflow()` из этой же библиотеки объединяет рецепт и модель в рабочий процесс. Наконец, функция `fit()` из `tidymodels` используется для обучения модели на обучающих данных.

```
# Прогнозирование на тестовом наборе
predictions <- predict(fit, new_data = test) |> bind_cols(test) |>
    select(Class:Survived, .pred_class)

# Более эффективный метод
pred_fit_tbl <- fit |> augment(new_data = test)
```

Этот код предсказывает вероятность выживания для каждого пассажира, указанного в тестовом наборе. Для прогнозирования используется функция `predict()` из библиотеки `tidymodels`. Аргумент `new_data` задает данные, на которых мы собираемся делать предсказания (в нашем примере это тестовый набор). Функция `bind_cols()` применяется для привязки прогнозов к данным тестового набора.

Мы применяем функцию `select()` для выбора столбцов, которые хотим сохранить. Объект `pred_fit_tbl` представляет собой экземпляр `tibble`, который содержит предсказания модели, а также истинные значения выживаемости. Этот объект будет использоваться для оценки производительности модели.

```
# Показатели точности, по которым будет оцениваться модель,
# из пакета healthyR.ai
perf <- hai_default_classification_metric_set()

# Вычисление показателей точности
perf(pred_fit_tbl, truth = Survived, estimate = .pred_class)

# Отображение матрицы ошибок
predictions |> conf_mat(truth = Survived, estimate = .pred_class)
```

Данный блок кода оценивает производительность модели на тестовом наборе. Для этого используется функция `hai_default_classification_metric_set()` из пакета `healthyR.ai`, создающая набор показателей успешности классификации по умолчанию, к которым относятся точность, прецизионность, полнота и оценка F1 (подробнее об этих показателях мы поговорим чуть позже).

Функция `perf()` используется для вычисления показателя точности на тестовом наборе. Объект `pred_fit_tbl` представляет собой датафрейм, содержащий предсказания модели, а также истинные значения выживаемости. Аргументы `truth` и `estimate` задают столбцы датафрейма, содержащие истинные и предсказанные значения соответственно.

Функция `conf_mat()` используется для отображения матрицы ошибок, показывающей количество правильно и неправильно предсказанных моделью результатов.

Наконец, функции `tidy()` и `glance()` из пакета `broom` можно использовать для упорядочения и обобщения подогнанной модели. Функция `tidy()` преобразует объект модели в экземпляр `tibble` — структуру данных, с которой гораздо проще работать. Функция `glance()` выводит краткую сводку, содержащую коэффициенты, стандартные ошибки и p-значения для всех переменных в модели.

Вот простое объяснение каждого из показателей, которые вычисляются в приведенном выше блоке кода.

- *Точность* — доля правильно предсказанных моделью результатов.
- *Прецизионность* — доля правильно предсказанных моделью положительных результатов.
- *Полнота* — доля правильно предсказанных моделью истинно положительных результатов.
- *Оценка F1* — среднее гармоническое значение прецизионности и полноты. Данная оценка является хорошим показателем общей эффективности модели.

Матрица ошибок — полезный инструмент, позволяющий получить представление о производительности модели. При оценке идеальной модели все наблюдения оказались бы на диагонали матрицы. Однако в реальности ни одна модель не является идеальной, так что некоторые наблюдения будут предсказаны неверно.

Наконец, мы оцениваем качество модели путем построения *кривой ROC* (кривой рабочей характеристики приемника; receiver operating characteristic). Подробнее об этом типе кривых можно прочитать здесь: https://www.tmwr.org/performance. Вот код, который создает кривую ROC:

```
roc_curve(
  pred_fit_tbl, truth = Survived, .pred_Yes,
  event_level = "second"
) |>
  autoplot()
```

А так выглядит результат его выполнения (рис. 9.1).

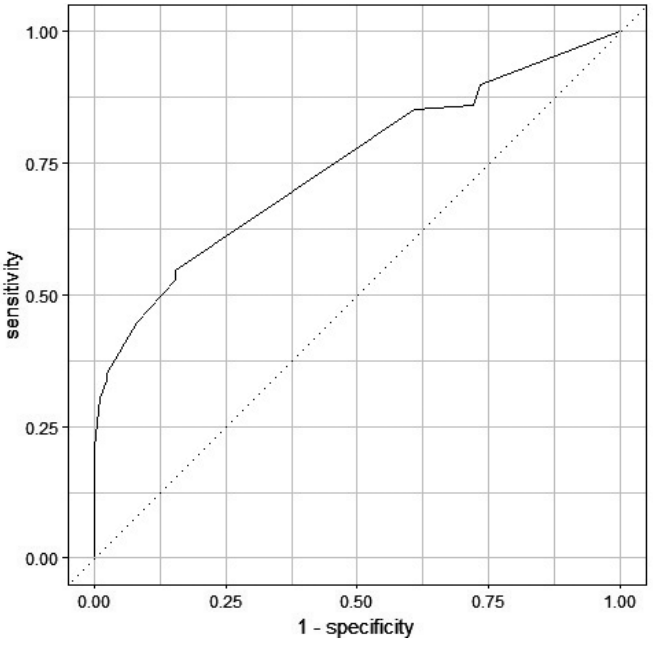


Рис. 9.1. ROC-кривая для модели логистической регрессии

Итак, вы научились выполнять линейную и логистическую регрессию с помощью базового функционала R и фреймворка `tidymodels`. Для этого вы использовали наборы данных `Titanic` и `iris`. Теперь пришло время сделать то же самое с помощью Python.

Выполнение линейной регрессии в среде Python с использованием данных Excel

Линейную регрессию в среде Python можно реализовать с помощью таких библиотек, как `pandas`, `scikit-learn`, `statsmodels` и `matplotlib`. Ниже приводится соответствующий код и описываются необходимые действия.

1. Сначала мы импортируем нужные библиотеки:

```
# Импорт необходимых библиотек
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import statsmodels.api as sm
from statsmodels.graphics.regressionplots import plot_regress_exog
from statsmodels.graphics.gofplots import qqplot
```

2. Затем создаем файл Excel с тестовыми данными. Разумеется, в реальном сценарии они вам не понадобятся — вы пропустите этот шаг и после загрузки необходимых библиотек загрузите данные из Excel (см. следующий шаг).

```
# Шаг 0: генерация выборки данных и их сохранение в файле Excel
np.random.seed(0)
n_samples = 100
X = np.random.rand(n_samples, 2) # Два признака
y = 2 * X[:, 0] + 3 * X[:, 1] + np.random.randn(n_samples)

# Линейная зависимость с шумом
# Создание датафрейма pandas
data = {'Feature1': X[:, 0], 'Feature2': X[:, 1], 'Target': y}
df = pd.DataFrame(data)

# Сохранение данных в Excel
df.to_excel("linear_regression_input.xlsx")
```

3. На этом этапе мы импортируем данные из файла Excel с тестовыми данными и подготавливаем их к анализу с помощью инструментов, описанных в предыдущей главе.

```
# Шаг 1: импорт данных Excel в датафрейм pandas
excel_file = "linear_regression_input.xlsx"
df = pd.read_excel(excel_file)

# Шаг 2: изучение данных
# Применение инструментов для выполнения EDA, описанных в предыдущей главе

# Шаг 3: подготовка данных (при необходимости)
# Применение инструментов для очистки данных, описанных в предыдущей главе
```

4. Теперь мы готовы к проведению анализа. Разбейте данные на обучающий и тестовый наборы, а затем подгоните линейную модель, основанную на *методе наименьших квадратов*, к обучающим данным.

```
# Шаг 4: разбиение данных на обучающий и тестовый наборы
X = df[['Feature1', 'Feature2']] # Независимые переменные
y = df['Target'] # Зависимая переменная

# Разбиение данных на обучающий и тестовый наборы с помощью
# фиксированного зерна генератора случайных чисел для обеспечения
# воспроизводимости результатов
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, \
    random_state=42)

# Шаг 5: подгонка модели линейной регрессии
# Прибавление константы (значения точки пересечения)
# к независимым переменным
X_train = sm.add_constant(X_train)
X_test = sm.add_constant(X_test)

# Подгонка линейной модели
model = sm.OLS(y_train, X_train).fit()
```

   Обратите внимание: выполнение импутации в рамках процесса очистки данных до разбиения их набора на тестовую и обучающую выборки может привести к загрязнению тестового набора данными из обучающего набора. Помните об этом при выполнении очистки и подготовки данных.

5. Теперь нужно оценить обученную модель на тестовых данных:

```
# Шаг 6: оценка модели
y_pred = model.predict(X_test)

# Отображение сводной статистики
print(model.summary())
```

   В результате выполнения этого кода на экране отобразится сводная статистика, дающая представление о взаимосвязях в наборе данных (рис. 9.2).

Интерпретация результатов модели выходит за рамки этой книги, однако следующие несколько подсказок помогут вам разобраться в них.

- **Коэффициенты**, связанные с каждой независимой переменной (предиктором) в модели, говорят о силе и направлении связи. Положительный коэффициент указывает на положительную корреляцию, при которой увеличение предиктора сопровождается увеличением целевой переменной. И наоборот, отрицательный коэффициент означает отрицательную корреляцию.

```
>>> # Print the model summary
>>> print(model.summary())
                            OLS Regression Results
==============================================================================
Dep. Variable:                 Target   R-squared:                       0.521
Model:                            OLS   Adj. R-squared:                  0.508
Method:                 Least Squares   F-statistic:                     41.84
Date:                Wed, 14 Feb 2024   Prob (F-statistic):           5.02e-13
Time:                        10:19:13   Log-Likelihood:                -108.46
No. Observations:                  80   AIC:                             222.9
Df Residuals:                      77   BIC:                             230.1
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0449      0.296      0.152      0.880      -0.544       0.634
Feature1       1.5628      0.392      3.988      0.000       0.783       2.343
Feature2       3.0963      0.369      8.395      0.000       2.362       3.831
==============================================================================
Omnibus:                        0.261   Durbin-Watson:                   2.275
Prob(Omnibus):                  0.878   Jarque-Bera (JB):                0.319
Skew:                           0.130   Prob(JB):                        0.853
Kurtosis:                       2.833   Cond. No.                         5.57
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
>>>
```

Рис. 9.2. Сводная статистика с результатами подогнанной модели

- **Точка пересечения.** Представляет собой прогнозируемое значение целевой переменной, когда все предикторные переменные равны нулю. Это значение очень важно учитывать в контексте анализа.
- **R-квадрат (R2).** Оценивает, насколько хорошо модель соответствует фактическим данным, и показывает долю дисперсии целевой переменной, объясняемой предикторами. Чем ближе значение R-квадрата к 1, тем лучше модель соответствует фактическим данным. Учтите, что добавление дополнительных переменных всегда увеличивает этот показатель. «Слишком хорошее» соответствие может говорить о «переобучении» модели, которого мы стремимся избежать. Для выбора модели вы можете использовать такие критерии, как Ср-критерий Мэллоу, AIC, BIC и скорректированный R-квадрат, которые штрафуют модель за слишком большое количество параметров, используемых для ее подгонки.
- **P-значения**, связанные с коэффициентами, помогают определить статистическую значимость каждого предиктора. Более низкие p-значения говорят

о большей значимости (то есть служат более веским доказательством для отклонения нулевой гипотезы). Если p-значение меньше заданного уровня значимости (например, 0,05), то можно сделать вывод о том, что предиктор оказывает статистически значимое влияние на целевую переменную. Однако имейте в виду, что использовать одни лишь p-значения не стоит (чтобы понять, почему, ознакомьтесь с содержанием дебатов по поводу p-хакинга и смежными темами статистической науки).

- **Остатки** (разница между наблюдаемыми и предсказанными результатами). Их изучение позволяет оценить производительность модели. В идеале остатки должны быть случайными и не содержать явных закономерностей. Закономерности в остатках могут указывать на неправильную спецификацию модели.
- **Доверительный интервал.** Это диапазон, в который с заданной вероятностью попадает истинное значение параметра генеральной совокупности. Более широкие интервалы соответствуют большей неопределенности.
- **F-статистика** (или критерий Фишера). Позволяет оценить общую значимость модели. Малое значение этого показателя говорит о том, что модель неэффективна в объяснении вариации зависимой переменной, и наоборот.
- **Скорректированный R-квадрат.** Это версия R-квадрата, скорректированная с учетом количества предикторов в модели. Помогает определить, улучшает ли добавление дополнительных предикторов качество модели.

Внимательно изучив эти показатели, вы сможете получить представление о том, насколько хорошо линейная модель соответствует вашим данным, о значимости предикторных переменных и общем качестве модели. Эта информация помогает принимать обоснованные решения и получать значимые выводы из результатов анализа.

После обучения модели и оценки ее соответствия данным мы можем визуализировать ее результаты, чтобы упростить их интерпретацию. Следующий код создает диаграмму рассеяния, отражающую взаимосвязь между предсказанными и наблюдаемыми значениями:

```
plt.scatter(X_test['Feature1'], y_test, color='blue', label='Actual')
plt.scatter(X_test['Feature1'], y_pred, color='red', label='Predicted')
plt.xlabel('Feature1')
plt.ylabel('Target')
plt.title('Linear Regression Prediction')
plt.legend()
plt.show()
```

Эта диаграмма выглядит следующим образом (рис. 9.3).

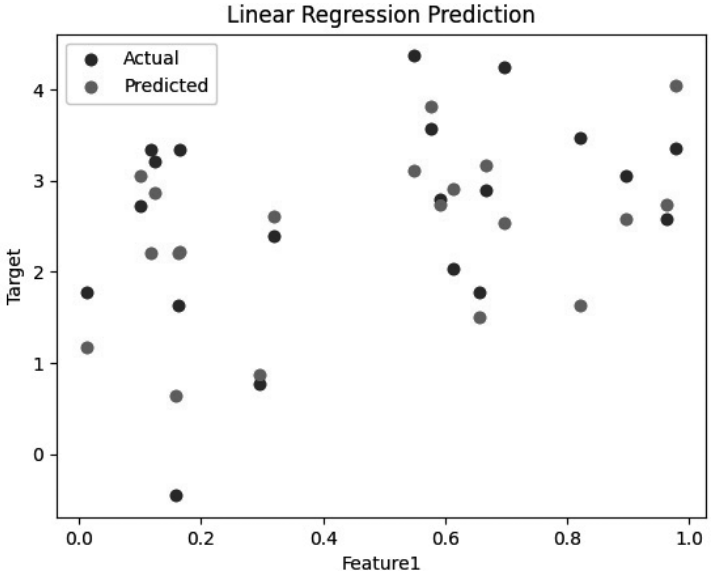


Рис. 9.3. График, отражающий взаимосвязь между прогнозами модели линейной регрессии и фактическими значениями

Кроме того, мы можем создать диагностические диаграммы и визуализации, в частности, графики остатков и квантильные графики, которые помогают выявить такие потенциальные проблемы в модели, как гетероскедастичность или выбросы.

```
# При необходимости укажите 'TkAgg' в качестве бэкенда перед генерацией
# графиков – закомментируйте эту строку, если вы работаете в WSL
# или другой неинтерактивной среде
plt.switch_backend('TkAgg')

# Остатки
fig, ax = plt.subplots(figsize=(12, 8))
plot_regress_exog(model, "Feature1", fig=fig)
plt.show()

# Создание квантильного графика
qqplot(model.resid, line="s")
plt.show()
```

Полученные графики выглядят следующим образом (рис. 9.4, 9.5).

К слову, библиотека `scikit-learn` имеет встроенную линейную модель, но не предусматривает такую удобную сводную статистику, которую мы использовали в коде, показанном выше.

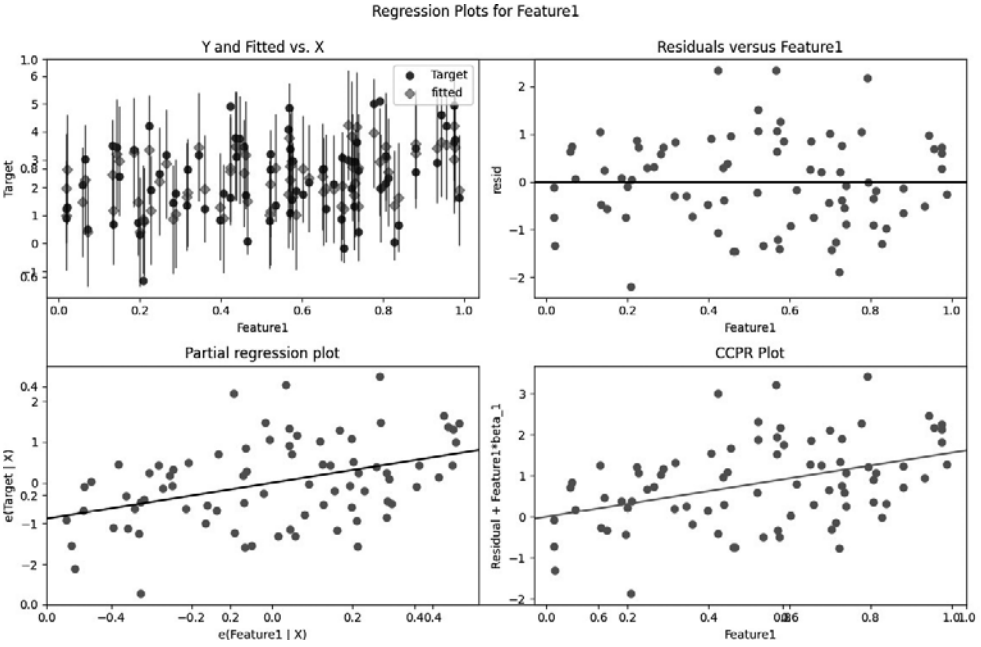


Рис. 9.4. График остатков

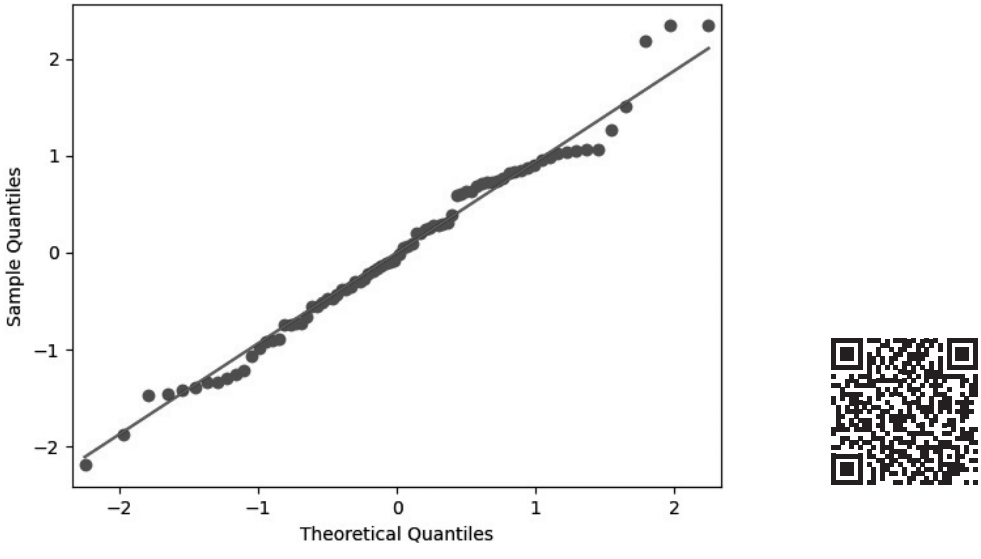


Рис. 9.5. Квантильный график остатков

Итак, теперь вы знаете, как выглядит базовый процесс выполнения линейной регрессии с использованием Python и данных Excel. Далее мы поговорим о выполнении логистической регрессии.

Выполнение логистической регрессии в среде Python с использованием данных Excel

В следующем коде мы генерируем случайную выборку данных с двумя признаками (`Feature1` и `Feature2`) и бинарной целевой переменной (`Target`) на основе простого условия. Мы выполняем логистическую регрессию, оцениваем модель с помощью показателя точности, матрицы ошибок и классификационного отчета, визуализируем результаты бинарной классификации и интерпретируем коэффициенты.

Ниже приведен пример кода с пошаговым разбором выполняемых действий.

1. Как обычно, мы начинаем с импорта библиотек:

```
# Импорт библиотек
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,
  classification_report
```

   В этом примере мы будем использовать другой набор данных:

```
# Шаг 0: генерация выборки данных
np.random.seed(0)
n_samples = 100
X = np.random.rand(n_samples, 2) # Два признака

# Бинарная классификация на основе условия
y = (X[:, 0] + X[:, 1] > 1).astype(int)

# Создание датафрейма pandas
data = {'Feature1': X[:, 0], 'Feature2': X[:, 1], 'Target': y}
df = pd.DataFrame(data)

df.to_excel("logistic_regression_input.xlsx")
```

2. Теперь мы можем прочитать данные Excel и подготовить их к этапу моделирования:

```
# Шаг 1: импорт данных Excel в датафрейм pandas
excel_file = "logistic_regression_input.xlsx"
df = pd.read_excel(excel_file)

# Шаг 2: разбиение данных на обучающий и тестовый наборы
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=42)
```

3. Далее мы можем создать и обучить модель. На этот раз мы будем использовать библиотеку `scikit-learn`.

```
# Шаг 3: создание и обучение модели логистической регрессии
model = LogisticRegression()
model.fit(X_train, y_train)
```

4. После обучения модели мы можем визуализировать ее результаты:

```
# Шаг 4: визуализация

# Визуализация результатов бинарной классификации
plt.scatter(X_test[y_test == 1][:, 0],
    X_test[y_test == 1][:, 1], color='blue',
    label='Class 1 (Actual)')
plt.scatter(X_test[y_test == 0][:, 0],
    X_test[y_test == 0][:, 1], color='red',
    label='Class 0 (Actual)')
plt.xlabel('Feature1')
plt.ylabel('Feature2')
plt.title('Logistic Regression Prediction')
plt.legend()
plt.show()
```

5. Поскольку в данном случае мы используем логистическую, а не линейную регрессию для бинарной классификации, нам нужны другие показатели для оценки качества модели.

```
# Шаг 5: оценка модели и интерпретация ее результатов
y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)
```

Результат работы модели выглядит следующим образом (рис. 9.6).

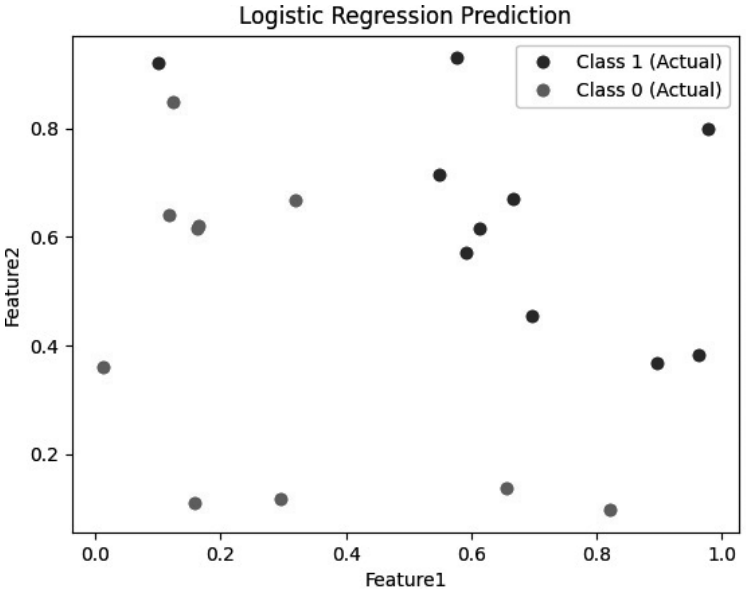


Рис. 9.6. График прогнозов модели логистической регрессии

Сводная статистика показана на рис. 9.7.

```
>>> print("Accuracy:", accuracy)
Accuracy: 0.9
>>> print("Confusion Matrix:\n", conf_matrix)
Confusion Matrix:
 [[ 8  2]
 [ 0 10]]
>>> print("Classification Report:\n", class_report)
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.80      0.89        10
           1       0.83      1.00      0.91        10

    accuracy                           0.90        20
   macro avg       0.92      0.90      0.90        20
weighted avg       0.92      0.90      0.90        20

>>>
```

Рис. 9.7. Сводная статистика модели

Интерпретацию сводной статистики можно начать с анализа следующих элементов.

- `Accuracy` (точность) — фундаментальный показатель, представляющий собой отношение правильно предсказанных моделью результатов к их общему количеству. Несмотря на свою простоту, этот показатель может ввести в заблуждение, если в наборе данных существует дисбаланс классов.
- `Confusion Matrix` (матрица ошибок) предоставляет более подробные сведения, разбивая предсказания модели на четыре категории: истинно положительные, истинно отрицательные, ложноположительные и ложноотрицательные. Эта матрица дает четкое представление о том, насколько правильно модель классифицирует положительные и отрицательные экземпляры.
- `Classification Report` (классификационный отчет) предоставляет исчерпывающую информацию и содержит такие показатели, как прецизионность (`precision`), полнота (`recall`), оценка F1 (`f1-score`) и поддержка (`support`) для обоих классов. Прецизионность отображает количество предсказанных положительных результатов, которые на самом деле оказались положительными. Полнота показывает долю правильно предсказанных истинно положительных результатов. Оценка F1 — среднее гармоническое значение прецизионности и полноты. Поддержка показывает количество экземпляров каждого класса. Совокупность этих показателей позволяет более точно оценить эффективность модели в плане решения задач бинарной классификации.

Вы можете использовать приведенный выше пример кода и выборку данных для тестирования и проведения экспериментов с моделью логистической регрессии.

Обратите внимание: вопреки распространенному (но неверному) утверждению, логистическую регрессию можно использовать и для решения задач регрессии. Классификатором ее делает произвольная точка отсечения, задаваемая для классификации наблюдений. В одних случаях вы можете использовать необработанный результат регрессии (например, если хотите предсказать вероятность принадлежности точки данных к классу, а не только более вероятный класс). В других же можете поэкспериментировать с точкой отсечения (например, если некие уже имеющиеся знания о предметной области предполагают, что 50 % не является подходящей точкой отсечения).

Итак, логистическая регрессия — относительно простая модель, обладающая множеством преимуществ. Она отличается универсальностью и высокой производительностью, легко настраивается и интерпретируется. Чаще всего она помогает решать задачи классификации с использованием точки отсечения, задаваемой на основе имеющихся знаний о предметной области, однако по своей сути остается методом регрессии, с помощью которого можно предсказывать вероятность принадлежности объектов к тем или иным классам.

Резюме

В этой главе мы обсудили способы выполнения линейной и логистической регрессии на данных Excel.

Линейная регрессия — фундаментальный статистический метод, позволяющий моделировать взаимосвязи между зависимыми и независимыми переменными. Мы обсудили ее варианты применения и допущения, а также рассмотрели процесс загрузки данных из Excel, их подготовки к анализу и обучения моделей линейной регрессии с помощью базового функционала R и фреймворка `tidymodels`, а также с помощью библиотек Python `scikit-learn` и `statsmodels`.

Изучив примеры кода, вы узнали о том, как проводить регрессионный анализ, оценивать точность модели и генерировать сводную статистику и показатели, позволяющие интерпретировать результаты модели. Кроме того, вы получили представление о том, как создавать диагностические диаграммы: графики остатков и квантильные графики, которые помогают выявлять такие проблемы, как гетероскедастичность и выбросы.

Логистическая регрессия — мощный инструмент прогнозирования вероятности принадлежности объектов к тому или иному классу и решения задач бинарной классификации. Мы обсудили ее важность и варианты применения, а также поговорили о том, как готовить данные, обучать модели и интерпретировать показатели. На практических примерах мы рассмотрели процесс создания моделей логистической регрессии с помощью R-фреймворка `tidymodels` и библиотеки Python `scikit-learn`.

К этому моменту вы уже должны хорошо разбираться в теории и практике применения линейной и логистической регрессии и уметь использовать эти методы для эффективного анализа данных.

В следующей главе мы поговорим об анализе временных рядов и применении соответствующих методов к данным Excel.

10 Анализ временных рядов: статистика, графики и прогнозирование

В сфере математического анализа, особенно при изучении данных и тенденций, графики временных рядов играют ключевую роль. Временной ряд — это графическое представление точек данных, собранных за несколько последовательных временных интервалов. Этот инструмент используется в различных областях, таких как экономика, финансы, экология и социальные науки, для анализа закономерностей, тенденций и динамики изменения данных во времени.

Типичный график временного ряда состоит из двух основных компонентов: оси времени и оси значений. Первая отображает ход времени, которое может измеряться в различных единицах: секундах, минутах, часах, днях, месяцах или годах. А вторая представляет значения изучаемой переменной, которой может быть что угодно, начиная с цены акции и температуры и заканчивая численностью населения и объемом продаж.

Процесс создания графика временного ряда состоит из следующих этапов.

- **Сбор данных.** Собирайте их с определенной периодичностью. Точная фиксация закономерностей требует, чтобы временные интервалы были постоянными.
- **Представление данных.** Нанесите собранные точки данных на график, сопоставив каждую точку с соответствующей отметкой на оси времени.
- **Масштабирование оси.** Задайте подходящий масштаб оси времени и оси значений, чтобы гарантировать точное отображение закономерностей.
- **Соединение точек данных.** В зависимости от контекста точки данных можно соединять линиями, кривыми или столбиками. Соединение точек упрощает выявление тенденций.

Как мы уже сказали, основная функция графика временного ряда заключается в анализе тенденций, закономерностей и аномалий в данных. Результатами такого анализа могут быть:

- **выявленные тенденции** — постепенное увеличение или уменьшение значений с течением времени. Могут повлиять на принятие решения;

- **сезонные колебания** — например, скачки объемов продаж в праздничные дни можно выявить путем анализа регулярных колебаний данных, наблюдаемых в течение определенных периодов;
- **циклические закономерности** — повторяющиеся, но нерегулярные колебания, которые могут зависеть от таких факторов, как экономические циклы или изменения в окружающей среде;
- **волатильность и выбросы** — внезапные скачки или провалы на графике привлекают внимание к событиям или факторам, влияющим на измеряемую переменную.

Кроме того, графики временных рядов служат основой для прогнозирования и выполнения прогностического анализа. Математические модели можно применять к историческим данным, чтобы делать предсказания относительно будущих тенденций и значений. Важно отметить: когда мы говорим о предсказаниях, то имеем в виду прогнозирование того, что может произойти, если все условия останутся прежними. Для этой цели обычно используются такие методы, как скользящие средние, экспоненциальное сглаживание и модели *авторегрессионного интегрированного скользящего среднего* (autoregressive integrated moving average, ARIMA).

В сфере математического анализа график временного ряда помогает понять динамику изменения данных во времени. Визуальное представление точек данных и тенденций позволяет исследователям, аналитикам и лицам, принимающим решения, извлекать ценные сведения, выявлять закономерности и делать обоснованные прогнозы. Графики временных рядов, рассматриваемые сквозь призму математического анализа, дают возможность системно подходить к изучению темпоральных данных и способствуют принятию более взвешенных решений.

В этой главе мы рассмотрим следующие темы:

- генерация случайных объектов временного ряда в среде R;
- создание временных рядов с помощью R;
- автоматический подбор параметров модели ARIMA с помощью библиотеки `healthyR.ts`;
- моделирование броуновского движения с помощью библиотеки `healthyR.ts`;
- анализ временных рядов в Python: статистика, графики и прогнозирование;
- создание временных рядов: простые диаграммы и графики АКФ/ЧАКФ;
- статистический анализ данных временных рядов;
- подходы к прогнозному моделированию;
- прогнозирование временных рядов с помощью модели глубокого обучения LSTM.

Технические требования

Файлы с кодом из примеров этой главы доступны по адресу https://github.com/PacktPublishing/Extending-Excel-with-Python-and-R/tree/main/Chapter%2010.

В данной главе мы будем использовать следующие пакеты:

- `healthyR.ts`;
- `forecast`;
- `timetk`;
- `Modeltime`;
- `prophet` (для Python);
- `keras`;
- `tensorflow`.

Генерация случайных объектов временного ряда в среде R

Начнем с создания объектов временных рядов с помощью базового функционала R. Основной класс объектов временных рядов в R — `ts`. Преобразовать объект в объект этого класса можно либо путем непосредственного применения функции `ts()`, либо с помощью вызова функции `as.ts()` на таком объекте, как вектор.

В этом разделе мы сгенерируем несколько случайных объектов временного ряда с помощью базового функционала R, который предусматривает различные распределения. Мы используем случайное нормальное распределение, вызвав функцию `rnorm()`, принимающую три параметра:

- `n` — количество точек, которое необходимо сгенерировать;
- `mean` — среднее значение распределения, которое по умолчанию равно 0;
- `sd` — стандартное отклонение распределения, которое по умолчанию равно 1.

Сгенерируем наш первый случайный вектор, который назовем `x`:

```
# Генерация случайного временного ряда
# Настройка зерна генератора случайных чисел для воспроизводимости результатов
set.seed(123)

# Генерация случайных точек с использованием нормального распределения
# со средним значением, равным 0, и стандартным отклонением, равным 1
n <- 25
x <- rnorm(n)
head(x)
[1] -0.56047565 -0.23017749 1.55870831 0.07050839 0.12928774 1.71506499
```

В этом коде выполняются следующие действия.

- Строка `set.seed(123)` гарантирует генерацию одинаковых случайных чисел при каждом запуске кода. Задав значение зерна генератора случайных чисел (в данном случае `123`), вы можете быть уверены в том, что он всегда будет генерировать одни и те же числа, что обеспечит воспроизводимость результатов анализа.
- В строке `n <- 25` мы определяем переменную `n` и задаем ее значение равным `25`. Эта переменная обозначает количество точек данных, которое должно содержаться в нашем случайном временном ряду.
- Строка `x <- rnorm(n)` отвечает за фактическую генерацию данных. Мы создаем новую переменную `x` и используем функцию `rnorm()` для генерации случайных чисел. Эти числа берутся из нормального (или гауссова) распределения, которое часто называется колоколообразной кривой. Значение `n` задает количество генерируемых случайных точек данных, которое в данном случае равно 25.
- В строке `head(x)` мы используем функцию `head()`, чтобы вывести на экран несколько первых значений переменной `x`. Это позволяет увидеть, как выглядят сгенерированные данные, то есть получить представление о наборе данных, не отображая его целиком.

Итак, приведенный выше код задает зерно генератора случайных чисел для обеспечения воспроизводимости результатов, указывает нужное количество точек данных (`25`) и генерирует эти точки данных из нормального распределения, сохраняя их в переменной `x`. Затем с помощью функции `head()` он выводит на экран несколько первых значений `x`. Подобный код часто используется в ходе анализа данных и при решении статистических задач, когда вам требуются случайные данные для работы или моделирования реальных сценариев.

Теперь преобразуем этот вектор `x` в объект временного ряда с помощью функции `ts()`:

```
# Преобразование x в объект ts
ts_obj <- ts(x)
```

Проверим класс только что созданного объекта:

```
class(ts_obj)
[1] "ts"
```

То же самое мы должны сделать с его структурой и атрибутами:

```
str(ts_obj)
  Time-Series [1:25] from 1 to 25: -0.5605 -0.2302 1.5587 0.0705 0.1293 ...
attributes(ts_obj)
$tsp
```

```
[1] 1 25 1
$class
[1] "ts"
```

Итак, что именно произошло?

- Строка `ts_obj <- ts(x)` создает объект временного ряда (`ts_obj`) из вектора или серии данных `x`. Этот шаг преобразует `x` в формат временного ряда.
- Строка `class(ts_obj)` проверяет и отображает класс `ts_obj`. Данная функция должна возвратить `ts`, указав на то, что `ts_obj` действительно является объектом временного ряда.
- Строка `str(ts_obj)` отображает структуру `ts_obj`, предоставляя информацию о временном ряде. В данном случае она показывает, что временной ряд имеет 25 точек данных в диапазоне от 1 до 25, а также отображает сами значения.
- Строка `attributes(ts_obj)` показывает атрибуты объекта временного ряда. В данном случае отображается временной интервал (`tsp`) со значениями `1 25 1`, которые означают, что временной ряд начинается с периода 1 и заканчивается на периоде 25, а частота регистрации значений равна 1.

Итак, по сути, этот код берет вектор `x`, преобразует его в объект временного ряда, а затем предоставляет информацию о его классе и структуре.

Теперь визуализируем полученный временной ряд с использованием функции `plot`. Мы можем сделать это с помощью вызова `plot(ts_obj)` (рис. 10.1).

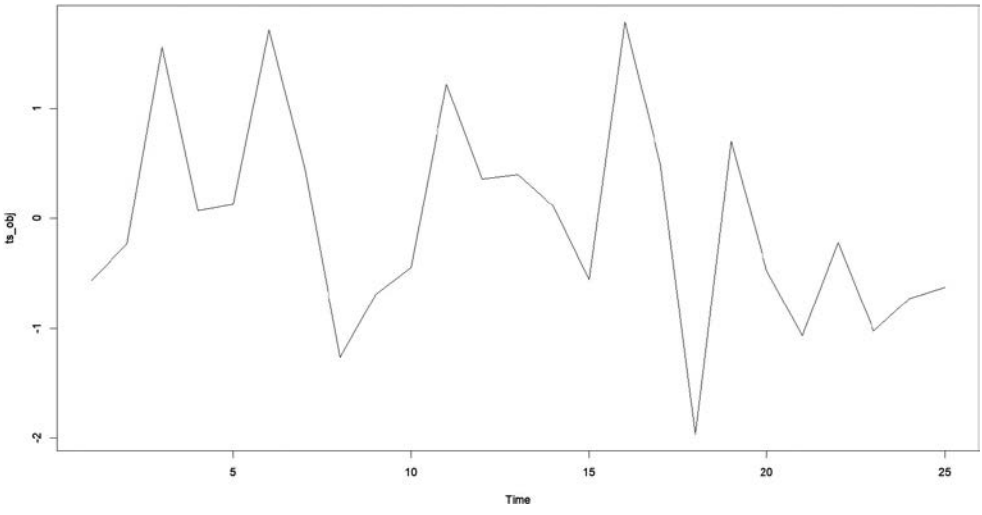


Рис. 10.1. График временного ряда, созданного с помощью функции rnorm(25)

Теперь, когда мы разобрались с тем, как преобразовать вектор в объект временного ряда, поговорим об изменении его параметров.

Изменение параметров временного ряда

В этом подразделе мы преобразуем вектор, созданный функцией `rnorm()`, в различные объекты временного ряда с разными параметрами начала, окончания и частоты. Это можно сделать с помощью базового функционала R.

Сначала взглянем на несколько примеров кода, а затем разберем их.

```
# Изменение начальной даты временного ряда
ts(x, start = 1980)
ts(x, start = c(1980, 05))
ts(x, start = 1980, frequency = 12)
ts(x, start = 1980, frequency = 12/3)

# Изменение даты окончания временного ряда
ts(x, end = 2023)
ts(x, end = 2023, frequency = 12)
ts(x, end = 2023, frequency = 12/3)
```

```
      Qtr1        Qtr2        Qtr3        Qtr4
2017 -0.56047565 -0.23017749  1.55870831  0.07050839
2018  0.12928774  1.71506499  0.46091621 -1.26506123
2019 -0.68685285 -0.44566197  1.22408180  0.35981383
2020  0.40077145  0.11068272 -0.55584113  1.78691314
2021  0.49785048 -1.96661716  0.70135590 -0.47279141
2022 -1.06782371 -0.21797491 -1.02600445 -0.72889123
2023 -0.62503927
```

Ниже приведены пояснения к каждому из предыдущих примеров.

- Строка `ts(x, start = 1980)` создает временной ряд с началом в 1980 году. Точные день и месяц не указаны, поэтому в качестве начальной даты по умолчанию используется 1 января 1980 года.
- Строка `ts(x, start = c(1980, 05))` явно указывает май 1980 года в качестве начальной даты временного ряда (сначала указывается год, а затем месяц).
- Строка `ts(x, start = 1980, frequency = 12)` создает временной ряд с началом в 1980 году и частотой фиксации точек данных в один месяц.
- Строка `ts(x, start = 1980, frequency = 12/3)` создает временной ряд с началом в `1980` году и частотой фиксации точек данных, равной 4. Это означает, что точки данных находятся на расстоянии одного квартала (трех месяцев) друг от друга.

- Строка `ts(x, end = 2023)` создает временной ряд с датой окончания в `2023` году. Начальная дата здесь не указана, поэтому в ее качестве по умолчанию используется начало сгенерированного временного ряда.
- Строка `ts(x, end = 2023, frequency = 12)` указывает в качестве даты окончания `2023` год, а в качестве частоты — 1 месяц.
- Строка `ts(x, end = 2023, frequency = 12/3)` указывает в качестве даты окончания `2023` год, а в качестве частоты — значение 4. Это означает, что точки данных находятся на расстоянии одного квартала (трех месяцев) друг от друга.

В последнем фрагменте предыдущего блока кода показан результат последнего изменения.

Эти примеры демонстрируют, как можно управлять такими параметрами временного ряда, как начальная и конечная точки и частота фиксации наблюдений. Теперь, когда у нас есть сгенерированные данные, создадим их график.

Создание временных рядов с помощью R

В этом разделе мы рассмотрим способы создания временных рядов, а также некоторых диагностических диаграмм, в частности графиков *автокорреляционной функции* (АКФ) и *частичной автокорреляционной функции* (ЧАКФ). Для начала считаем набор данных `AirPassengers` с помощью пакета `readxl`:

```
# Чтение файла airpassengers.xlsx и преобразование в объект ts
# с началом в 1949 г.
ap_ts <- read_xlsx("./Chapter 10/airpassengers.xlsx") |>
    ts(start = 1949, frequency = 12)

# Создание графика объекта ts
plot(ap_ts)
```

В результате получается следующий график (рис. 10.2).

На графике отчетливо видны тренд и сезонные колебания. Взяв за основу это наблюдение, мы можем выполнить декомпозицию временного ряда, то есть разложить его на следующие четыре компонента:

- наблюдаемые данные;
- тренд;
- сезонная компонента;
- случайная компонента.

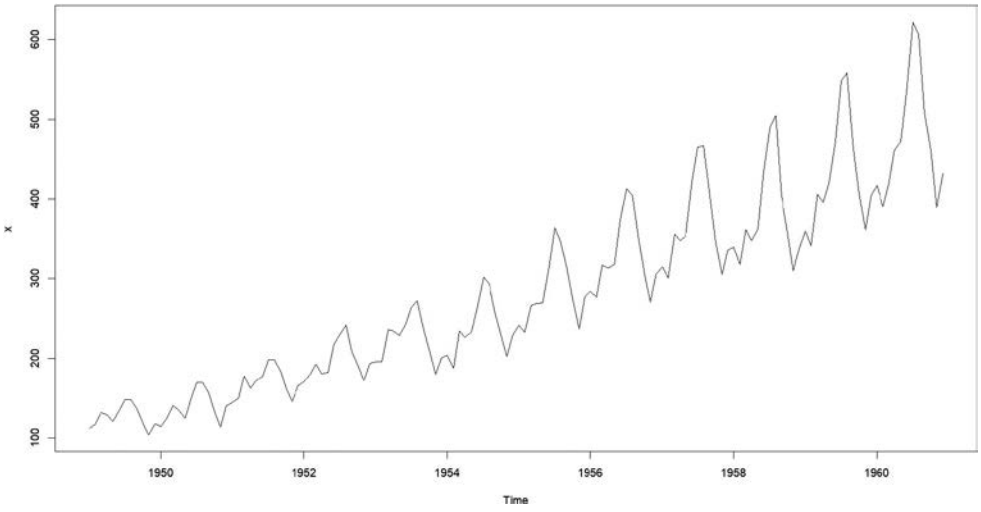


Рис. 10.2. Визуализация набора данных AirPassengers в виде временного ряда

Визуализируем результаты этой декомпозиции. Вот необходимый для этого код:

```
plot(decompose(ap_ts))
```

Получившиеся графики показаны на рис. 10.3.

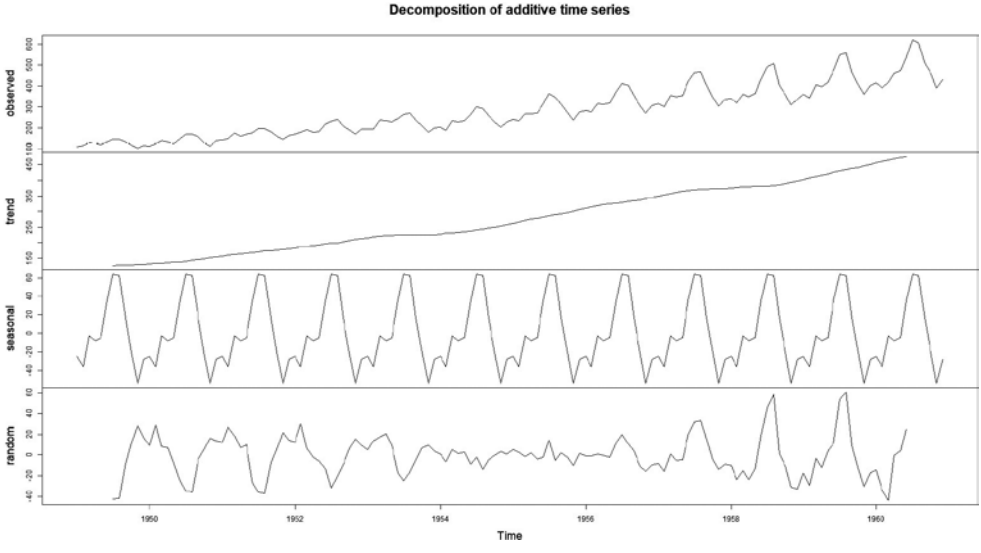


Рис. 10.3. Результат декомпозиции временного ряда

Выполнив декомпозицию, мы можем приступить к созданию и анализу графиков АКФ и ЧАКФ.

Создание и анализ графиков АКФ и ЧАКФ в среде R

График АКФ — это график автокорреляционной функции, которая отражает взаимосвязь текущего и предыдущего наблюдений. Она показывает, насколько текущее наблюдение коррелирует с теми же значениями временного ряда, взятыми с различными сдвигами во времени (лагами). Так, при наличии сезонного спроса на пиво вы заметите сильную корреляцию между текущим наблюдением и наблюдением, зафиксированным в предыдущий аналогичный сезонный период.

Вывод функции `acf()` выглядит так:

```
acf(ap_ts)
```

Создаваемый ею график показан на рис. 10.4.

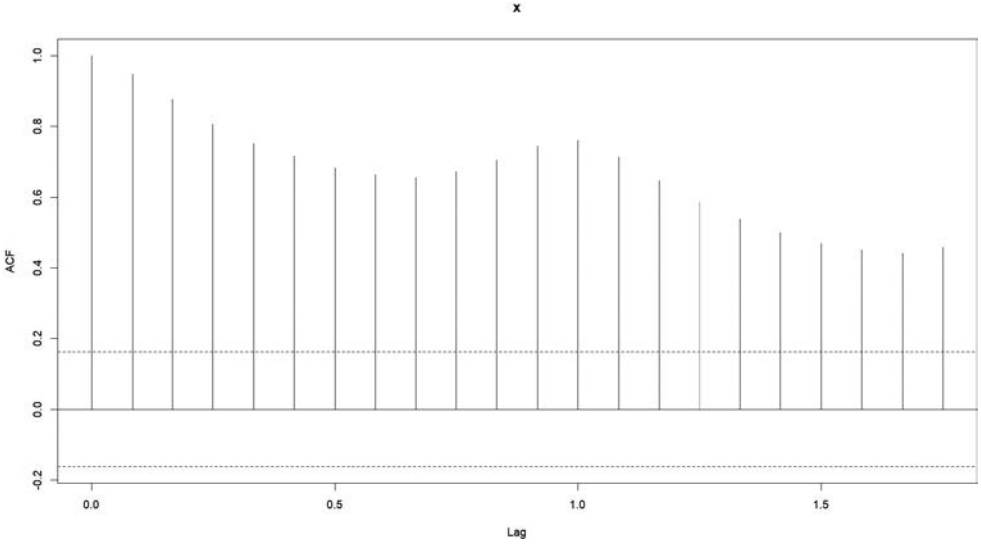


Рис. 10.4. График АКФ для набора данных AirPassengers

Здесь мы можем заметить сильную взаимосвязь последовательных точек, поскольку в самих данных присутствует восходящий тренд. Однако мы также видим на графике пики и долины, свидетельствующие о сезонных корреляциях.

Теперь посмотрим на график ЧАКФ, который тоже генерируется функцией `acf()` при задании типа `"partial"`.

Частичная автокорреляция отражает взаимосвязь наблюдения *t* и некоего наблюдения *t-n* при исключении влияния на эту взаимосвязь промежуточных наблюдений. Теперь посмотрим на график ЧАКФ, созданный для того же временного ряда с помощью функции `acf(ap_ts, type = "partial")` (рис. 10.5).

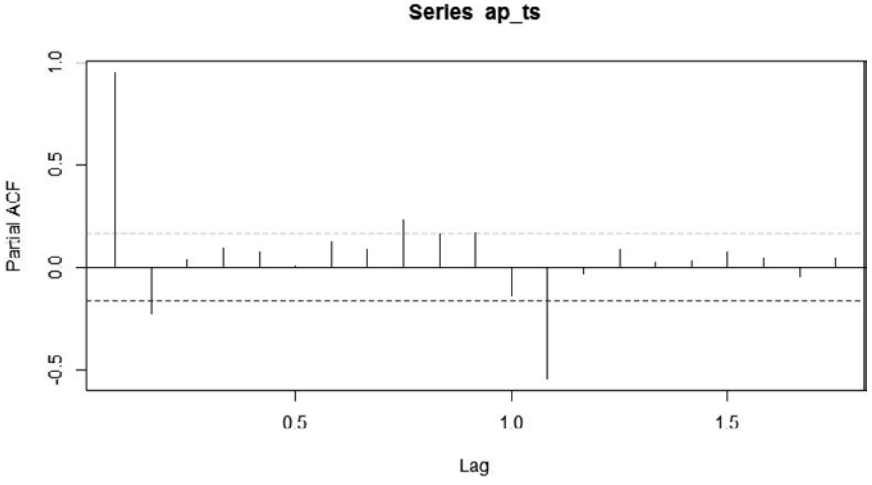


Рис. 10.5. График ЧАКФ для набора данных AirPassengers

Итак, мы рассмотрели способы создания графиков АКФ и ЧАКФ в среде R и кратко обсудили, что они собой представляют. Теперь поговорим о моделировании временного ряда с помощью библиотеки `healthyR.ts`.

Автоматический подбор параметров модели ARIMA с помощью библиотеки healthyR.ts

Временные ряды, как и любой другой набор данных, можно моделировать. Для этого существует огромное количество как старых, так и новых методов. В этом разделе мы обсудим способ создания модели ARIMA, а точнее, автоматического подбора ее параметров с помощью библиотеки `healthyR.ts` в среде R. Модели ARIMA используются для описания автокорреляций в данных.

Мы реализуем рабочий процесс, который завершается созданием и подгонкой настроенной модели с помощью функции `ts_auto_arima()`. Эта модель требует того, чтобы наши данные представляли собой объект `tibble`. В этом примере мы будем использовать набор данных `AirPassengers`.

Итак, преобразуем уже загруженный набор данных `AirPassengers` в тиббл:

```
library(healthyR.ts)
library(dplyr)
library(timetk)
ap_tbl <- ts_to_tbl(ap_ts) |>
  select(-index)
> class(ap_tbl)
[1] "tbl_df"      "tbl"          "data.frame"
```

Здесь мы загрузили библиотеки `healthyR.ts` и `dplyr`, а затем преобразовали уже существующий объект временного ряда `ap_ts` в тиббл с помощью функции `ts_to_tbl()` из библиотеки `healthyR.ts`. Далее нам нужно разбить данные на обучающий и тестовый наборы с помощью функции `time_series_split()` из библиотеки `timetk`:

```
# Разбиение данных временного ряда
splits <- time_series_split(
  data = ap_tbl
  , date_var = date_col
  , assess = 12
  , skip = 3
  , cumulative = TRUE
)
> splits
<Analysis/Assess/Total>
<132/12/144>
```

После разбиения данных мы можем выполнить основную функцию. Она является шаблонной в том смысле, что выполняет большинство действий автоматически. Однако вы всегда можете сделать с моделью все, что пожелаете. Эти шаблонные функции из библиотеки `healthyR.ts` не являются чем-то незаменимым.

Выполним эту функцию и посмотрим на результаты:

```
Library(modeltime)
ts_auto_arima <- ts_auto_arima(
  .data = ap_tbl,
  .num_cores = 10,
  .date_col = date_col,
  .value_col = x,
  .rsamp_obj = splits,
  .formula = x ~ .,
  .grid_size = 20,
  .cv_slice_limit = 5,
  .tune = TRUE
)
```

В этом фрагменте кода мы предоставили тиббл с данными и указали количество ядер, которые хотим использовать, — в нашем случае `10`. Затем указали столбец, в котором хранится дата, — `date_col`. Столбцом значений является `x` из предоставленного нами тиббла. Далее следует наш объект повторной выборки, `splits`, а затем формула, передаваемая в функцию рецепта внутри функции `ts_auto_arima()`. В данном случае это формула, сопоставляющая значение `x` с датой. В качестве размера настроечной сетки мы задаем значение `20` и ограничиваем количество создаваемых срезов данных пятью (`.cv_slice_limit = 5`). Для параметра `.tune` мы задаем значение `TRUE`, которое дает функции команду приступить к настройке модели. Из-за этой настройки возврат данных может занять несколько секунд/минут.

Посмотрим на выходные данные. Первое, на что мы обращаем внимание, — это информация о рецепте:

```
> ts_auto_arima$recipe_info
$recipe_call
recipe(.data = ap_tbl, .date_col = date_col, .value_col = x,
    .formula = x ~ ., .rsamp_obj = splits, .tune = TRUE, .grid_size = 20,
    .num_cores = 10, .cv_slice_limit = 5)

$recipe_syntax
[1] "ts_arima_recipe <-"
[2] "\n  recipe(.data = ap_tbl, .date_col = date_col, .value_col = x,
.formula = x ~ \n    ., .rsamp_obj = splits, .tune = TRUE, .grid_size = 20,
.num_cores = 10, \n     .cv_slice_limit = 5)"

$rec_obj

── Recipe ──────────────────────────────────────────────────────────────
─────────────────────────

── Inputs
Number of variables by role
outcome:   1
predictor: 1
```

Итак, из вывода очевидно, что у нас есть один результат (`outcome`) и одна предикторная переменная (`predictor`). Это все, что нам нужно для создания модели Auto ARIMA без экзогенных регрессоров.

Теперь изучим информацию о модели:

```
ts_auto_arima
> ts_auto_arima$model_info
$model_spec
ARIMA Regression Model Specification (regression)

Main Arguments:
  seasonal_period = tune::tune()
  non_seasonal_ar = tune::tune()
  non_seasonal_differences = tune::tune()
```

```
  non_seasonal_ma = tune::tune()
  seasonal_ar = tune::tune()
  seasonal_differences = tune::tune()
  seasonal_ma = tune::tune()

Computational engine: arima
```

Здесь мы видим, что для всех параметров модели задано значение `tune::tune()`, что позволяет прогнать модель через настроечную сетку:

```
$wflw
══ Workflow
Preprocessor: Recipe
Model: arima_reg()

── Preprocessor
0 Recipe Steps

── Model
ARIMA Regression Model Specification (regression)

Main Arguments:
  seasonal_period = tune::tune()
  non_seasonal_ar = tune::tune()
  non_seasonal_differences = tune::tune()
  non_seasonal_ma = tune::tune()
  seasonal_ar = tune::tune()
  seasonal_differences = tune::tune()
  seasonal_ma = tune::tune()

Computational engine: arima
```

Созданный далее объект рабочего процесса показывает, что рецепт не предусматривает никаких шагов, поскольку мы не выполняем никаких преобразований.

```
$fitted_wflw
…
── Model
Series: outcome
ARIMA(4,1,2)(1,0,1)[12]

Coefficients:
         ar1     ar2     ar3      ar4     ma1      ma2     sar1     sma1
      -0.221  0.9020  0.0894  -0.2144  0.0477  -0.9523  0.9695  -0.0869
s.e.   0.092  0.0996  0.0958   0.0875  0.0367   0.0365  0.0143   0.0927

sigma^2 = 99.46:  log likelihood = -497.36
AIC=1012.72   AICc=1014.21   BIC=1038.6

$was_tuned
[1] "tuned"
```

Подогнанный объект рабочего процесса показывает, что в качестве лучшей модели была выбрана `ARIMA(4,1,2)(1,0,1)[12]`. Вдобавок он сообщает нам ее коэффициенты и значение AIC.

Теперь изучим возвращаемый объект `model_calibration`:

```
> ts_auto_arima$model_calibration
$plot

$calibration_tbl
# Таблица модельного времени
# Тиббл: 1 × 5
  .model_id .model .model_desc.type .calibration_data
      <int> <list>      <chr>                          <chr> <list>
          1 1 <workflow> ARIMA(4,1,2)(1,0,1)[12] Test  <tibble [12 × 4]>

$model_accuracy
# Тиббл: 1 × 9
  .model_id .model_desc.type     mae  mape  mase smape  rmse   rsq
      <int> <chr>                                <chr> <dbl> <dbl>  <dbl> <dbl>
<dbl> <dbl>
1           1 ARIMA(4,1,2)(1,0,1)[12] Test     16.2  3.35              0.335
3.35  19.5  0.960
```

Получившийся в результате график показан на рис. 10.6.

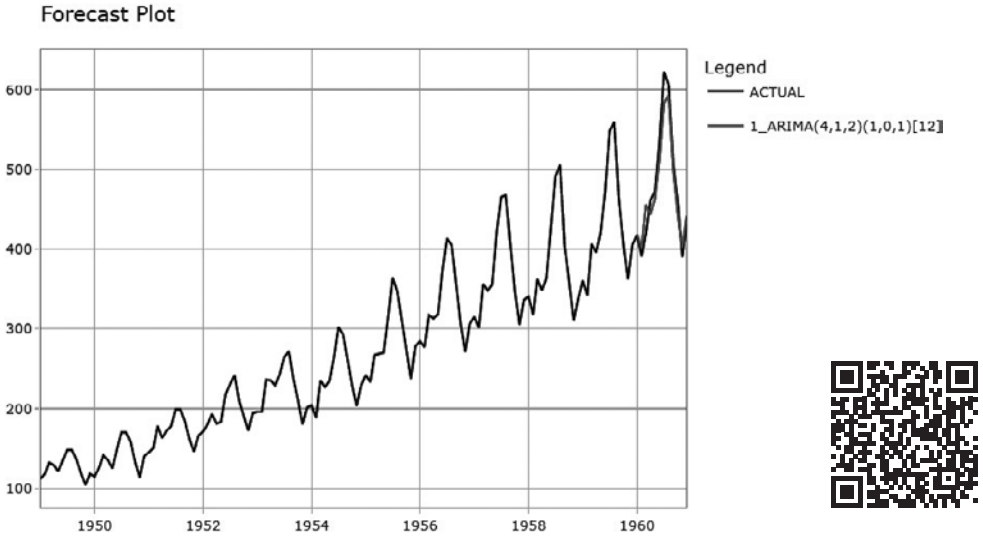


Рис. 10.6. График калибровки модели Auto ARIMA

Наконец, рассмотрим возвращаемый объект `tuned_model`:

```
> ts_auto_arima$tuned_info
$tuning_grid
# Тиббл: 20 × 7
   seasonal_period non_seasonal_ar non_seasonal_differences non_seasonal_ma
seasonal_ar
   <chr>          <int>           <int>          <int>       <int>
 1 weekly           3               0              1           2
 2 yearly           5               1              4           0
# i еще 2 переменные: seasonal_differences <int>, seasonal_ma <int>

$tscv
# План перекрестной проверки временных рядов
# Тиббл: 5 × 2
  Splits               id
  <list>               <chr>
1 <split [120/12]> Slice1
2 <split [117/12]> Slice2
3 <split [114/12]> Slice3
4 <split [111/12]> Slice4
5 <split [108/12]> Slice5

$tuned_results
# Результаты настройки
# Н/д
# Тиббл: 5 × 4
  splits               id      .metrics                 .notes
  <list>               <chr>  <list>                    <list>
1 <split [120/12]> Slice1 <tibble [120 × 11]> <tibble [1 × 3]>
2 …
$grid_size
[1] 20
$best_metric
[1] "rmse"

$best_result_set
# Тиббл: 1 × 13
  seasonal_period non_seasonal_ar non_seasonal_differences non_seasonal_ma
seasonal_ar
  <chr>           <int>           <int>          <int>          <int>
1 yearly             4               1              2              1
# i еще 8 переменных: seasonal_differences <int>, seasonal_ma <int>, .metric
<chr>,
# .estimator <chr>, mean <dbl>, n <int>, std_err <dbl>, .config <chr>
$tuning_grid_plot
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
$plotly_grid_plot
```

По этому коду видно, что в данной части возвращаемого функцией результата содержится много информации. Ее должно хватить для принятия обоснованного решения о том, целесообразно ли продолжать настройку модели.

Последний график, который возвращается как в виде статического объекта `ggplot2`, так и в виде объекта `plotly`, показан на рис. 10.7.

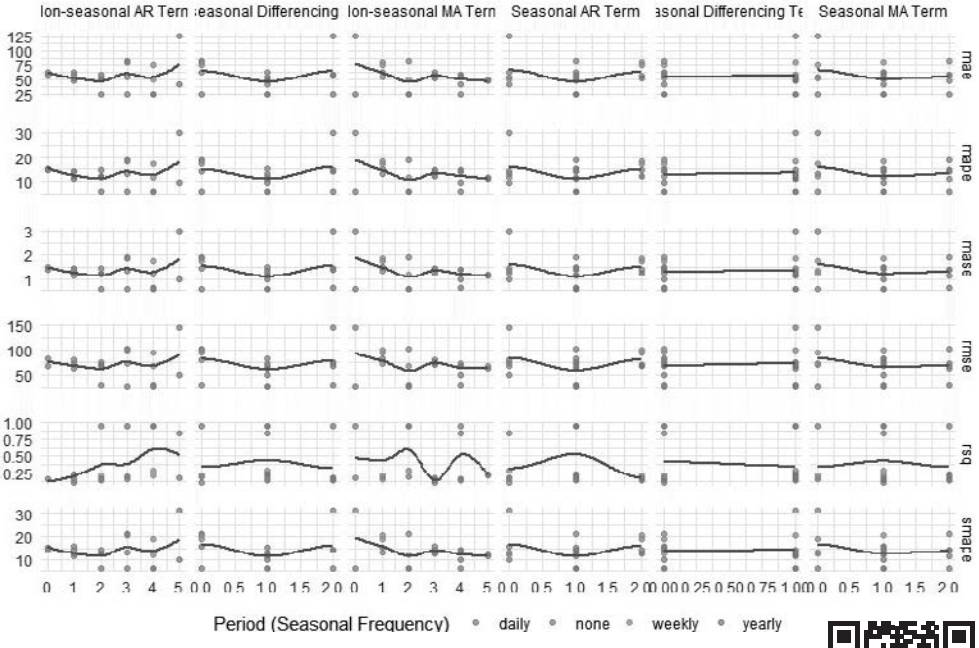


Рис. 10.7. Настроечная сетка модели Auto ARIMA

Теперь рассмотрим концепцию броуновского движения.

Моделирование броуновского движения с помощью библиотеки healthyR.ts

Последним графиком временного ряда, который мы рассмотрим, будет визуализация *броуновского движения*, также известного как винеровский процесс. Это фундаментальная концепция в сфере финансов и математики, которая описывает случайное движение частиц в жидкости. В сфере финансов оно часто используется для моделирования динамики цен на такие финансовые инструменты, как акции, товары и валюты.

Ниже перечислены некоторые ключевые характеристики броуновского движения.

- **Случайность.** Будущая величина и направление движения в любой момент времени не могут быть с уверенностью предсказаны.
- **Непрерывная траектория.** Это означает, что цена актива изменяется плавно, без резких скачков и разрывов.
- **Независимые приращения.** Изменения цены актива, происходящие в рамках неперекрывающихся временных интервалов, не зависят друг от друга. Иными словами, движение цены в одном интервале не влияет на движение цены в другом.
- **Нормальное (гауссово) распределение.** Приращения броуновского движения (изменения цены) нормально распределены, то есть подчиняются закону Гаусса. Это соответствует представлению о том, что на финансовых рынках малые изменения цены происходят чаще, чем большие.
- **Постоянная дисперсия.** Дисперсия ценовых приращений не меняется с течением времени. Это свойство иногда называют *гомоскедастичностью*.

С математической точки зрения изменение цены актива $S(t)$ с течением времени t может быть описано с помощью стохастического дифференциального уравнения:

$$dS(t) = \mu S(t)dt + \sigma S(t)dW(t),$$

где:

$dS(t)$ — бесконечно малое изменение цены актива за небольшой промежуток времени, dt;

μ — скорость дрейфа или ожидаемая средняя доходность актива;

σ — волатильность актива, представляющая собой стандартное отклонение изменений его цены;

$dW(t)$ — винеровский процесс, представляющий случайное приращение.

Броуновское движение лежит в основе многих финансовых моделей, в частности модели ценообразования опционов Блэка — Шоулза и модели динамики процентных ставок Васичека. Оно помогает понять и оценить поведение финансовых инструментов, отражая присущие им случайность и волатильность. Однако важно отметить, что поведение реальных финансовых инструментов может отклоняться от идеального броуновского движения под влиянием таких факторов, как настроение участников рынка, новости и другие внешние воздействия.

Мы можем легко смоделировать броуновское движение и создать его график с помощью библиотеки `healthyR.ts`. Необходимый для этого код выглядит так:

```
library(healthyR.ts)
ts_brownian_motion() |>
  ts_brownian_motion_plot(t, y)
```

Его вывод показан на рис. 10.8 (поскольку речь идет о случайном процессе, ваш результат, скорее всего, будет несколько иным).

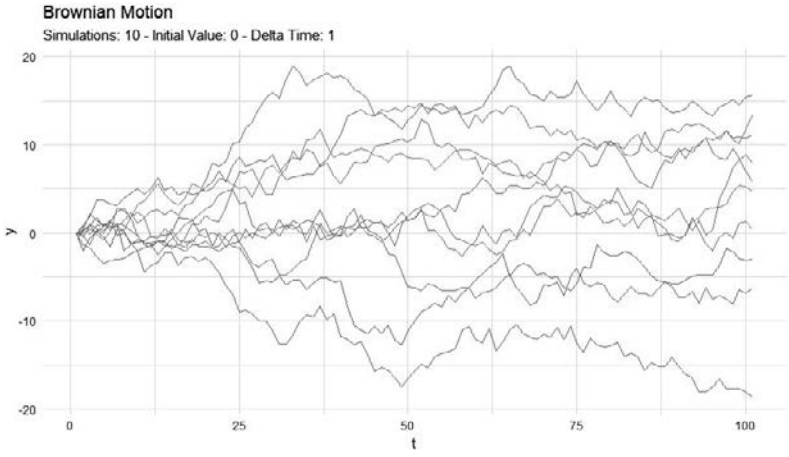


Рис. 10.8. Моделирование броуновского движения с помощью библиотеки healthyR.ts

Теперь, когда мы разобрались с анализом временных рядов в среде R, посмотрим, как это делается в Python.

Анализ временных рядов в Python: статистика, графики и прогнозирование

Прежде чем приступать к анализу временных рядов, необходимо подготовить данные для работы. В этом разделе мы рассмотрим процесс создания фиктивных данных временного ряда, их сохранения в файл Excel и последующего считывания в `pandas`. Они послужат основой для дальнейшего анализа.

Как обычно, начинаем с загрузки необходимых библиотек:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Теперь нужно создать образцы данных и сохранить их в Excel, чтобы иметь возможность использовать их в остальных разделах этой главы:

```
# Создание диапазона дат
date_rng = pd.date_range(start='2022-01-01', end='2023-12-31', freq='D')

# Создание тренда
trend = 0.05 * np.arange(len(date_rng))
```

```
# Создание сезонной компоненты (цикличности)
seasonal = 2.5 * np.sin(2 * np.pi * np.arange(len(date_rng)) / 365)

# Добавление случайного шума
noise = np.random.normal(0, 0.5, len(date_rng))

# Объединение всех компонент для создания временного ряда
time_series = trend + seasonal + noise

# Создание датафрейма
df = pd.DataFrame({'Date': date_rng, 'Value': time_series})

# Сохранение данных в файл Excel
df.to_excel('time_series_data.xlsx', index=False)
```

Наконец, мы должны загрузить данные из Excel и получить представление о том, как они выглядят:

```
# Считывание данных обратно в pandas
loaded_df = pd.read_excel('time_series_data.xlsx')

# Отображение нескольких первых строк
print(loaded_df.head())
```

В этом примере мы генерируем синтетический временной ряд с линейным трендом, синусоидальной сезонной компонентой и случайным шумом. Такой набор данных хорошо отображает реальные временные ряды, закономерности в которых часто состоят из комбинации этих элементов. Затем мы сохраняем сгенерированный набор данных в файл Excel, откуда его при необходимости можно считать обратно в среду Python для анализа.

Создание временных рядов: простые диаграммы и графики АКФ/ЧАКФ

Визуализация данных временных рядов позволяет понять лежащие в их основе закономерности и тенденции. В этом разделе мы рассмотрим различные графики временных рядов и способы их создания с помощью Python. Эти визуализации помогут получить представление о сезонности, тенденциях и автокорреляции в данных временного ряда.

Начнем с загрузки необходимых библиотек:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```

Теперь нужно загрузить данные из Excel и убедиться в корректности преобразования информации о дате:

```
# Загрузка данных временного ряда (замените 'time_series_data.xlsx' своим файлом)
data = pd.read_excel('time_series_data.xlsx')

# Преобразование столбца 'Date' в формат даты и времени
# и назначение в качестве индекса
data['Date'] = pd.to_datetime(data['Date'])
data.set_index('Date', inplace=True)
```

Теперь мы можем создать простой график временного ряда, чтобы получить первичное представление о наших данных:

```
# Создание временного ряда
plt.figure(figsize=(12, 6))
plt.plot(data['Value'])
plt.title('Time Series Plot')
plt.xlabel('Date')
plt.ylabel('Value')
plt.grid(True)
plt.show()
```

Получившийся график показан на рис. 10.9.

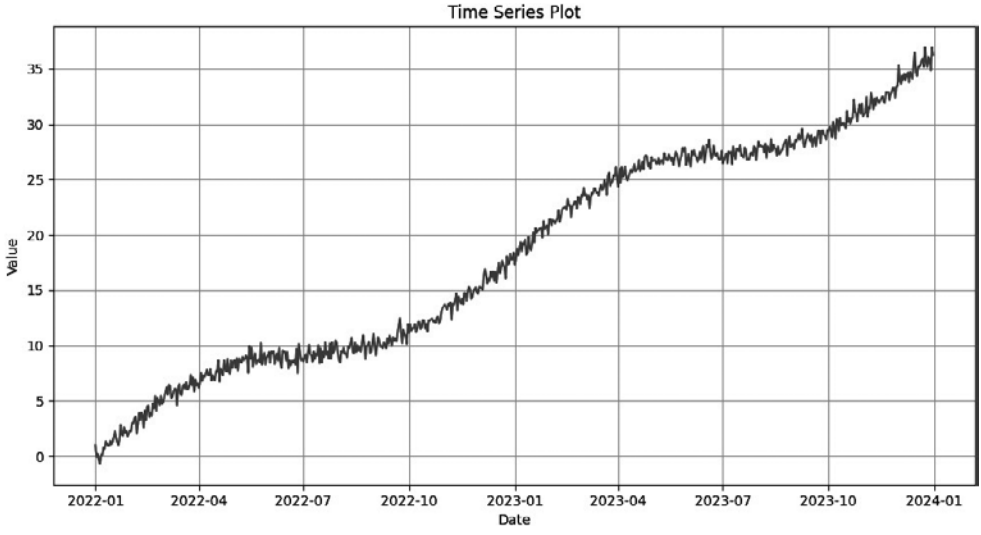


Рис. 10.9. График временного ряда

Создадим более сложные графики АКФ и ЧАКФ с помощью следующего кода:

```
# Построение графиков АКФ и ЧАКФ
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 8))

# График АКФ
plot_acf(data['Value'], lags=10, ax=ax1)
ax1.set_title('Autocorrelation Function (ACF)')

# График ЧАКФ
plot_pacf(data['Value'], lags=40, ax=ax2)
ax2.set_title('Partial Autocorrelation Function (PACF)')

plt.tight_layout()
plt.show()
```

Получившиеся графики показаны на рис. 10.10.

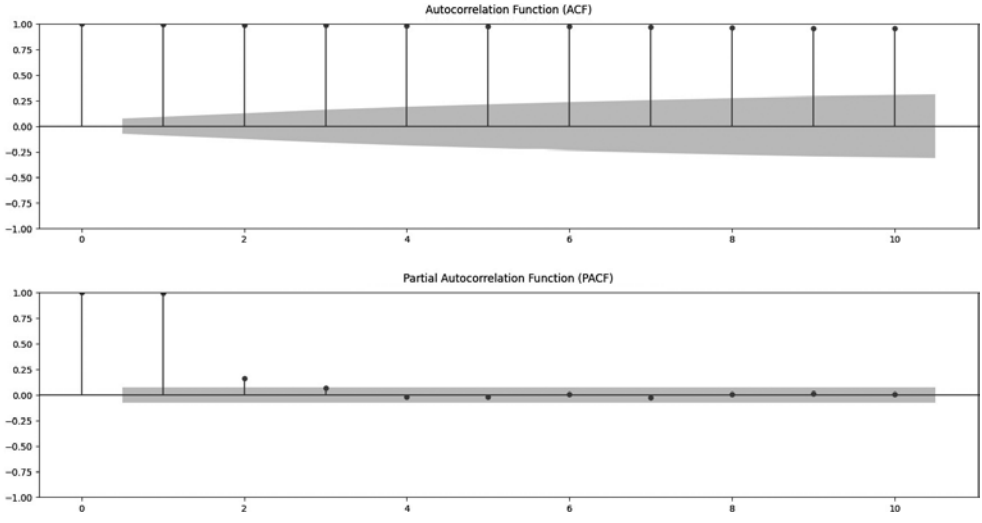


Рис. 10.10. Графики АКФ и ЧАКФ для лагов 1 и 2

Здесь мы начали с загрузки данных временного ряда из файла Excel и преобразования столбца `Date` в индекс, имеющий формат даты и времени. Затем создали график временного ряда для визуализации динамики данных во времени. Кроме того, мы создали графики АКФ и ЧАКФ, позволяющие изучать автокорреляцию во временном ряду. Эти графики ценны тем, что помогают определить потенциальные значения лагов для создания моделей временных рядов.

Теперь поговорим о том, как использовать графики АКФ и ЧАКФ для выявления сезонности, тенденций и автокорреляции в данных временных рядов.

График автокорреляционной функции (АКФ)

График АКФ показывает корреляцию между временным рядом и его сдвинутыми во времени версиями. Другими словами, он определяет, насколько хорошо текущее значение временного ряда коррелирует с его прошлыми значениями.

Пики на определенных лагах на графике АКФ указывают на потенциальную сезонность или закономерности в данных. Например, значительный пик на лаге 7 во временном ряду с дневной частотой указывает на недельную сезонность, а пик на лаге 12 во временном ряду с месячной частотой может говорить о годовой сезонности.

Анализируя график АКФ, вы можете определить значения лагов, при которых корреляции значительно отклоняются от нуля. Эти лаги могут дать представление о сезонности в данных. Кроме того, выявление экспоненциального затухания функции АКФ может свидетельствовать о наличии авторегрессионной составляющей в модели временного ряда.

График частичной автокорреляционной функции (ЧАКФ)

График ЧАКФ измеряет прямую взаимосвязь между временным рядом и его сдвинутыми во времени версиями, исключая влияние промежуточных лагов.

Пики на определенных лагах на графике ЧАКФ указывают на количество сдвинутых значений, которые непосредственно влияют на текущее значение временного ряда. Например, значительный пик на лаге 1 и отсутствие значительных пиков после него указывают на *процесс авторегрессии* первого порядка.

График ЧАКФ помогает определить порядок авторегрессии (p) в моделях ARIMA. Он также позволяет выявить резкие изменения во временном ряду, указывающие на структурные разрывы или сдвиги.

Анализируя эти графики в сочетании с данными временного ряда, вы можете получить ценную информацию о наличии сезонности, тенденций и автокорреляции, что поможет вам выбрать подходящие модели временных рядов и сделать более точные прогнозы.

На рис. 10.10 мы видим, что на графике ЧАКФ только лаг 1 показывает сильную автокорреляцию, в то время как график АКФ отображает высокую автокорреляцию на всех исследуемых лагах вследствие особенностей использованного способа генерации случайных данных.

Получив представление об этих графиках, мы можем перейти к статистическому анализу.

Статистический анализ данных временных рядов

Методы статистического анализа Python позволяют получить более полное представление о временных рядах. В этом разделе мы обсудим тест ADF и декомпозицию временного ряда.

Тест ADF

Тест ADF (Augmented Dickey-Fuller test, расширенный тест Дики — Фуллера) — статистический метод, который используется для оценки *стационарности* временного ряда. Стационарность является фундаментальной концепцией в анализе временных рядов, поскольку упрощает процесс моделирования. Стационарный временной ряд обладает такими статистическими свойствами, как постоянное среднее значение и дисперсия. Нестационарные данные, напротив, демонстрируют тенденции или сезонность, что усложняет моделирование и точное прогнозирование.

В рамках теста ADF нулевая гипотеза (H0) предполагает, что данные не являются стационарными, а альтернативная (H1) — что данные стационарны. Проанализировав p-значение, полученное в результате проведения теста, вы можете решить, можно ли отвергнуть нулевую гипотезу. Низкое p-значение (обычно менее 0,05) указывает на то, что данные стационарны, а высокое говорит об обратном. Поэтому при проведении теста ADF p-значение меньше 0,05 — показатель стационарности временного ряда.

Вот пример кода, который реализует тест ADF в Python с помощью библиотеки `statsmodels`:

```
from statsmodels.tsa.stattools import adfuller
import pandas as pd

# Считывание данных обратно в pandas
df = pd.read_excel('time_series_data.xlsx')

# Расширенный тест Дики – Фуллера
adf_result = adfuller(df['Value'])
print("\nAugmented Dickey-Fuller Test:")
print(f"ADF Statistic: {adf_result[0]}")
print(f"P-value: {adf_result[1]}")
print("Null Hypothesis (H0): Data is non-stationary")
print("Alternative Hypothesis (H1): Data is stationary")

if adf_result[1] <= 0.05:
    print("Result: Reject the null hypothesis. Data is stationary.")
else:
    print("Result: Failed to reject the null hypothesis. Data \
        is non-stationary.")
```

Нам не удалось отвергнуть нулевую гипотезу, и это неудивительно — сгенерированные нами данные временного ряда имеют четкий линейный тренд и, следовательно, не являются стационарными.

Чтобы получить более полное представление о данных, мы можем проанализировать полученный вывод (рис. 10.11).

```
>>> # Print ADF test results
>>> print("Test Statistic:", adf_result[0])
Test Statistic: -0.24536819384067887
>>> print("p-value:", adf_result[1])
p-value: 0.9328933413659218
>>> print("Lags Used:", adf_result[2])
Lags Used: 9
>>> print("Number of Observations Used for the ADF Regression:", adf_result[3])
Number of Observations Used for the ADF Regression: 720
>>> print("Critical Values:")
Critical Values:
>>> for key, value in adf_result[4].items():
...     print(f"  {key}: {value}")
...
  1%: -3.439464954327953
  5%: -2.8655625802683473
  10%: -2.5689120852623457
>>> print("Maximized Information Criterion (IC):", adf_result[5])
Maximized Information Criterion (IC): 1208.8292254446185
>>>
```

Рис. 10.11. Результаты теста ADF

Как видите, результат теста ADF, возвращенный `adfuller`, содержит несколько компонентов.

- Первое значение в кортеже (в нашем случае `-0.24536819384067887`) — *тестовая статистика*, которая используется для оценки нулевой гипотезы о наличии единичного корня в наборе данных временного ряда. Чем более отрицательным (или менее положительным) является ее значение, тем больше оснований для отклонения нулевой гипотезы.
- Второе значение (в нашем случае `0.9328933413659218`) — связанное с тестовой статистикой *p-значение*, которое отражает вероятность ее наблюдения при условии истинности нулевой гипотезы. Небольшое p-значение (обычно менее 0,05) — повод отклонить нулевую гипотезу в пользу гипотезы стационарности.
- Третье значение (в нашем случае `9`) показывает *количество лагов*, используемых в регрессии при проведении теста ADF.
- Четвертое значение (в нашем случае `720`) показывает *количество наблюдений, использованных в регрессии ADF, и критических значений.*

- Следующий компонент — *словарь*, содержащий критические значения для различных доверительных уровней (1, 5 и 10 %). Эти критические значения используются для определения значимости тестовой статистики.
- Последнее значение (в нашем случае `1208.8292254446185`) — *максимизированный информационный критерий* (maximized information criterion, IC), который показывает, насколько хорошо модель подогнана к данным. Чем ниже значение IC, тем выше степень этого соответствия.

Как правило, интерпретация результатов теста ADF сводится к изучению тестовой статистики и p-значения. Чем более отрицательным (или менее положительным) является значение тестовой статистики и чем меньше p-значение (обычно менее 0,05), тем больше оснований для того, чтобы отвергнуть нулевую гипотезу о наличии единичного корня и сделать вывод о стационарности временного ряда.

Декомпозиция временного ряда

Декомпозиция временного ряда — метод разложения данных временного ряда на такие ключевые компоненты, как тренд, сезонность и остатки. Эти компоненты позволяют получить ценную информацию о закономерностях, лежащих в основе временного ряда, что облегчает дальнейшее прогнозирование.

- *Тренд* — основное долгосрочное движение или тенденция в данных. Отражает общее направление изменения значений, то есть их увеличение или уменьшение с течением времени.
- *Сезонность* — закономерности в данных, повторяющиеся через фиксированные промежутки времени. В зависимости от данных речь может идти о ежедневных, еженедельных, ежемесячных или ежегодных закономерностях. Выявлять сезонность очень важно, поскольку это помогает обнаруживать периодические тенденции, которые следует учитывать при прогнозировании.
- *Остатки* — нерегулярные или случайные компоненты данных временного ряда. Это то, что остается после исключения тренда и сезонности. Анализ остатков помогает выявить любые дополнительные закономерности или необычные явления в данных.

В нашем коде мы выполняем декомпозицию временного ряда, чтобы разложить его на составляющие, и визуализируем каждую из них. Это позволяет понять структуру данных, что облегчает выявление закономерностей, которые могут повлиять на дальнейшее прогнозирование.

Посмотрим, как декомпозиция временных рядов выполняется в Python:

```
from statsmodels.tsa.seasonal import seasonal_decompose
import matplotlib.pyplot as plt

# Декомпозиция временных рядов
decomposition = seasonal_decompose(df['Value'], model='additive', period=365)
```

```
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

# Создание графиков для различных компонентов
plt.figure(figsize=(12, 8))
plt.subplot(411)
plt.plot(df['Date'], df['Value'], label='Original')
plt.legend(loc='best')
plt.subplot(412)
plt.plot(df['Date'], trend, label='Trend')
plt.legend(loc='best')
plt.subplot(413)
plt.plot(df['Date'], seasonal, label='Seasonal')
plt.legend(loc='best')
plt.subplot(414)
plt.plot(df['Date'], residual, label='Residual')
plt.legend(loc='best')
plt.suptitle("Time Series Decomposition")
plt.show()
```

Полученный график не требует пояснений и отражает искусственный способ генерации данных (рис. 10.12).

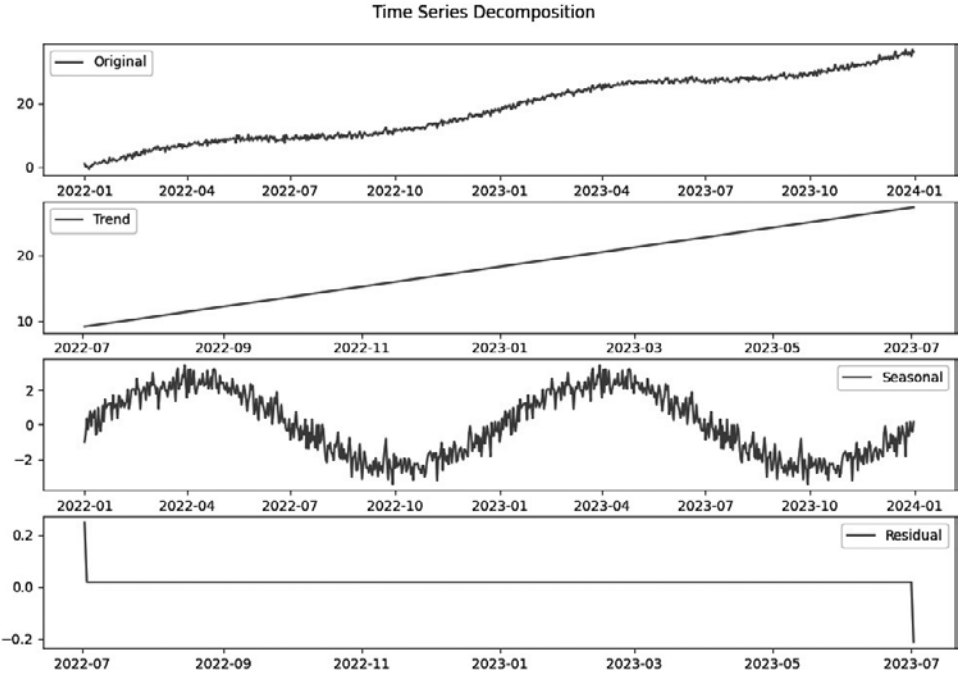


Рис. 10.12. Результат декомпозиции временного ряда

Итак, в этом разделе мы рассмотрели основные статистические показатели, помогающие разобраться в данных временных рядов, а также методы, позволяющие решать наиболее типичные задачи моделирования, в частности, связанные с декомпозицией и интерпретацией компонентов. Теперь, когда вы получили более глубокое представление о временных рядах благодаря созданию графиков и статистическому анализу, мы можем перейти к самому важному этапу — прогнозированию.

Подходы к прогнозному моделированию

В этом разделе мы поговорим о реализации методов прогнозного моделирования с помощью двух мощных библиотек Python — `statsmodels` и `prophet`. Они предоставляют множество инструментов для прогнозирования временных рядов, позволяя вам принимать взвешенные решения и делать прогнозы на основе имеющихся данных.

Прогнозирование с помощью библиотеки statsmodels

Популярная библиотека Python `statsmodels` содержит широкий спектр статистических инструментов, в том числе таких, с помощью которых можно выполнять анализ временных рядов. Для целей прогнозирования она предлагает функцию создания моделей ARIMA, которые являются основным инструментом анализа временных рядов, позволяющим выявлять и моделировать сложные закономерности в данных.

Создание модели ARIMA с помощью `statsmodels` предполагает выбор подходящего порядка дифференцирования, авторегрессионных компонентов и скользящего среднего — все это позволяет выявлять основные закономерности в данных. Когда модель будет создана, можно переходить к прогнозированию и оценке ее эффективности.

Наконец, важно отметить: имеющийся у нас временной ряд не является стационарным, поэтому мы должны смоделировать изменения в нем.

Рассмотрим рабочий процесс, реализованный в коде.

1. Сначала мы импортируем необходимые библиотеки и загружаем данные:

```
# Импорт необходимых библиотек
import pandas as pd
import numpy as np
import statsmodels.api as sm
from scipy.stats import norm
import matplotlib.pyplot as plt

# Загрузка данных временного ряда (замените название файла на свое)
time_series_data = pd.read_excel('time_series_data.xlsx') ['Value']
```

2. Затем выполняем проверку на стационарность и решаем, хотим смоделировать сам временной ряд или его разности:

```
# Выполнение расширенного теста Дики — Фуллера
# для проверки на стационарность
result = sm.tsa.adfuller(time_series_data, autolag='AIC')

# Если p-значение превышает пороговый уровень (например, 0,05),
# то выполняется дифференцирование, чтобы сделать данные стационарными
if result[1] > 0.05:
        differenced_data = np.diff(time_series_data, n=1)
else:
        differenced_data = time_series_data
```

3. После дифференцирования данных (которое мы выполнили, так как наш временной ряд не являлся стационарным) мы можем создать модель и подогнать ее:

```
# Создание модели ARIMA
order = (1, 1, 1) # Значения, основанные на результате анализа АКФ и ЧАКФ
model = sm.tsa.ARIMA(differenced_data, order=order)

# Подгонка модели ARIMA
model_fit = model.fit()
```

4. Вместо того чтобы основывать свое решение исключительно на графиках, мы могли бы выполнить гипернастройку параметров порядка. Однако в данном примере это не потребуется. Гипернастройка предполагает настройку параметров, определяющих модель, например настройку порядка модели ARIMA, а не параметров, подгоняемых под данные.

5. Теперь мы можем создать прогноз с помощью обученной модели.

   Обратите внимание: мы смоделировали дифференцированные данные, поэтому необходимо преобразовать прогнозы обратно в фактический временной ряд:

```
# Создание прогнозов
forecast_steps = 50 # При необходимости измените количество шагов
forecast = model_fit.forecast(steps=forecast_steps)

# Если p-значение превышает пороговый уровень (например, 0,05),
# то выполняется дифференцирование, чтобы сделать данные стационарными
if result[1] > 0.05:
    # Модель обучена на дифференцированных данных, поэтому
    # прогнозные значения должны прибавляться к последней точке данных
    cumsum_forecasts = np.cumsum(forecast)
```

```
    # Прибавьте эту кумулятивную сумму к последнему наблюдаемому
    # значению в исходных данных
    real_forecasts = cumsum_forecasts + time_series_
data[len(time_series_data)-1]

else:
    real_forecasts = forecast
```

6. Далее мы должны вычислить основные статистические показатели и использовать их для расчета доверительного интервала:

```
# Получение параметров модели ARIMA
params = model_fit.params
p, d, q = order
resid = model_fit.resid

# Вычисление стандартных ошибок
stderr = np.std(resid)

# Расчет доверительных интервалов
z_score = norm.ppf(0.975) # Для 95%-ного доверительного интервала
conf_int = np.column_stack((real_forecasts - z_score * stderr,
    real_forecasts + z_score * stderr))

# Разделение прогнозов на точечные прогнозы и доверительные интервалы
point_forecasts = real_forecasts # Точечные прогнозы
forecast_stderr = stderr         # Стандартные ошибки прогнозов
lower_bound = conf_int[:, 0]     # Нижняя граница доверительного интервала
upper_bound = conf_int[:, 1]     # Верхняя граница доверительного интервала
```

7. Наконец, мы должны создать график прогноза вместе с исходным временным рядом, чтобы визуально оценить эффективность модели:

```
# Визуализация исходного временного ряда и прогнозов
plt.figure(figsize=(12, 6))
plt.plot(time_series_data, label='Original Time Series', color='blue')
plt.plot(range(len(time_series_data),
    len(time_series_data) + forecast_steps),
    real_forecasts, label='Forecast', color='red')
plt.fill_between(range(len(time_series_data),
    len(time_series_data) + forecast_steps),
    lower_bound, upper_bound, color='pink', alpha=0.5)
plt.xlabel('Time Steps')
plt.ylabel('Value')
plt.title('ARIMA Time Series Forecast')
plt.legend()
plt.show()
```

Полученный график (рис. 10.13) позволяет нам сделать определенные выводы.

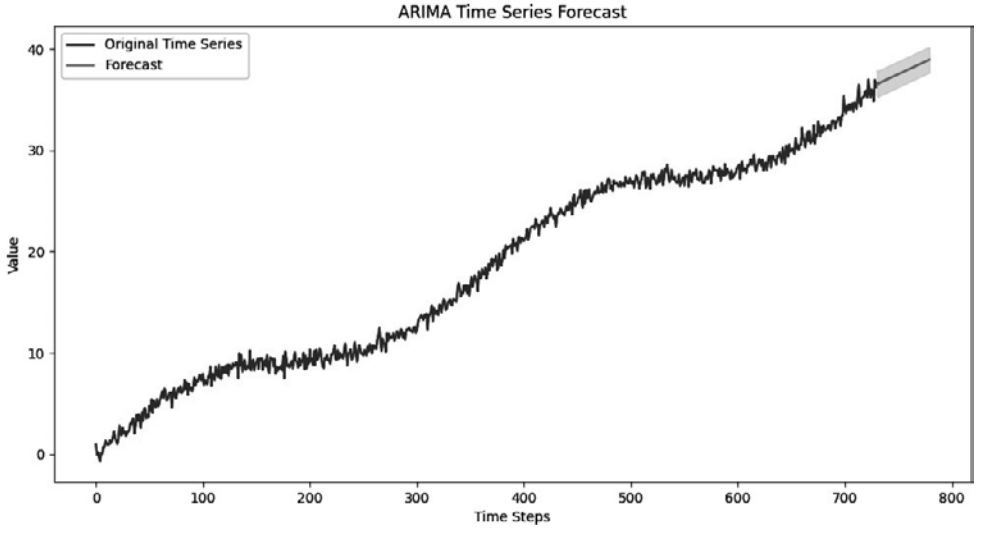


Рис. 10.13. Временной ряд и прогноз, полученный с помощью statsmodels

Мы видим, что, несмотря на довольно эффективное отражение тренда и сезонности, модель является слишком упрощенной и не может в полной мере продемонстрировать природу временного ряда.

Чтобы это исправить, можем использовать более сложную модель из библиотеки `prophet`.

Прогнозирование временных рядов с помощью библиотеки prophet

Библиотека `prophet` была разработана специально для прогнозирования временных рядов. Она известна простотой использования и широким выбором моделей, предназначенных для самых разных сценариев.

Эта библиотека предоставляет интуитивно понятный способ моделирования временных рядов, а также инструменты для оптимизации гиперпараметров и оценки прогнозов.

Код, который мы будем использовать, заметно более простой.

1. Сначала мы должны загрузить необходимые библиотеки, считать данные из Excel и подготовить их, создав датафрейм со столбцами, которые ожидает получить модель `prophet`:

```
# Импорт необходимых библиотек
import pandas as pd
from prophet import Prophet
from prophet.plot import plot

# Загрузка данных временного ряда (замените название файла своим)
time_series_data = pd.read_excel('time_series_data.xlsx')

# Создание датафрейма со столбцами 'ds' и 'y'.
df = pd.DataFrame({'ds': time_series_data['Date'], 'y':
    time_series_data['Value']})
```

2. Теперь мы должны настроить модель, используя знания о предметной области (в нашем случае мы исходим из знаний о том, как именно были сгенерированы данные):

```
# Инициализация и подгонка модели Prophet без недельной сезонности
model = Prophet(weekly_seasonality=False)

# Добавление сезонной составляющей на основе знаний о предметной области
# (в нашем случае мы исходим из знаний о том, как именно
# были сгенерированы данные)
model.add_seasonality(name='custom_season', period=365, fourier_order=5)

# Подгонка настроенной модели
model.fit(df)
```

3. Наконец, мы можем создать прогноз и его график:

```
# Создание датафрейма для будущих дат
forecast_steps = 150 # При необходимости измените количество шагов
future = model.make_future_dataframe(periods=forecast_steps, freq='D')

# Прогнозирование
forecast = model.predict(future)

# Создание графика прогноза
fig = model.plot(forecast)

fig.show()

# Создание графика составляющих прогноза:
# тренда, годовой и недельной сезонности
fig2 = model.plot_components(forecast)
fig2.show()
```

Полученный график демонстрирует гораздо более качественный прогноз с более узким доверительным интервалом по сравнению с тем, который был создан с помощью библиотеки `statsmodel`. На нем явно видны тренд и сезонность (рис. 10.14).

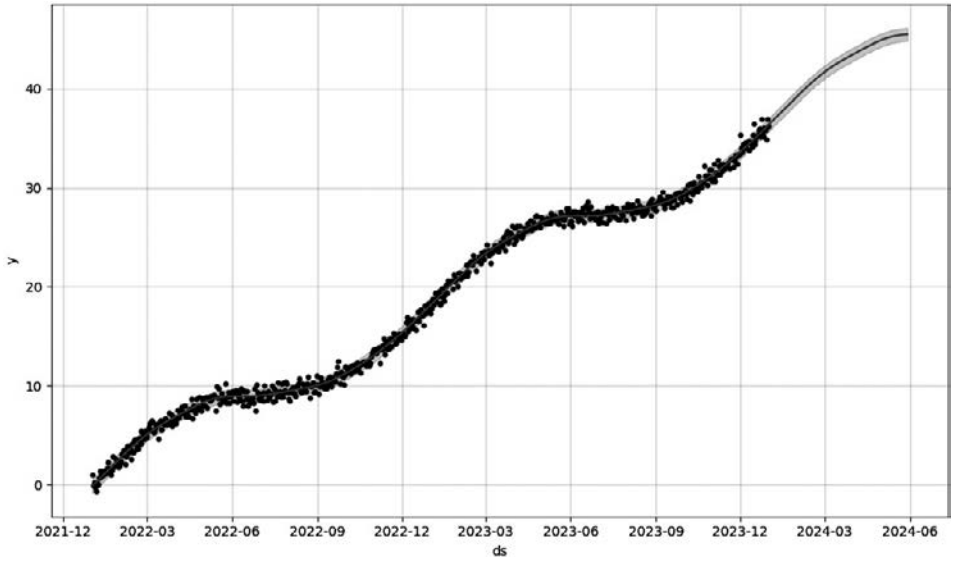


Рис. 10.14. Временной ряд и прогноз, созданный с помощью библиотеки prophet

Чтобы дополнительно оценить соответствие модели данным, рассмотрим ее компоненты (рис. 10.15).

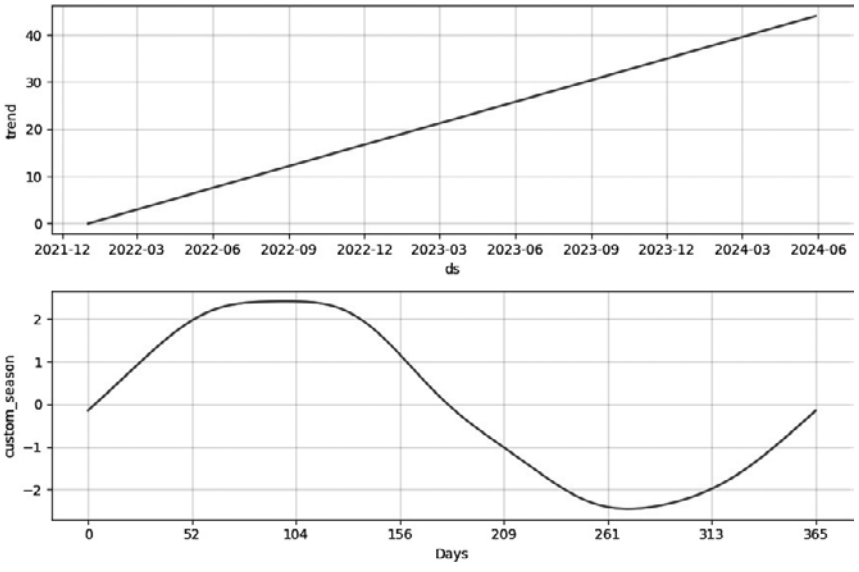


Рис. 10.15. Компоненты модели временного ряда, подогнанной библиотекой prophet

Как видите, библиотека `prophet` подгоняет модель очень простым способом, который не требует угадывания начальных параметров. Чтобы улучшить исходную модель, вы можете и должны использовать любые знания о предметной области.

В приведенном выше фрагменте кода мы отключили недельную сезонность по умолчанию и добавили пользовательскую годовую сезонность. При необходимости можем выполнить гораздо больше действий:

- добавить пользовательский календарь праздников, например, с помощью команды `model.add_country_holidays(country_name='US')`;
- поэкспериментировать с точками изменения тренда, если мы уверены в том, что они имеются, но не знаем, где именно;
- выполнить гипернастройку порядка ряда Фурье для пользовательской сезонности.

Итак, в этом разделе вы изучили основы прогнозирования данных временных рядов с помощью библиотек `statsmodel` и `prophet`. Освоив эти навыки, вы сможете выявлять тенденции и составлять прогнозы, а также принимать обоснованные решения.

Прогнозирование временных рядов с помощью модели глубокого обучения LSTM

В этом разделе вы узнаете о передовых методах прогнозирования временных рядов с помощью модели глубокого обучения. В частности, рассмотрим способ реализации нейронной сети *долгой краткосрочной памяти* (Long Short-Term Memory, LSTM) с использованием библиотеки `keras`. Вне зависимости от того, работаете вы с традиционными временными рядами или с более сложными, высокоразмерными данными, эта модель глубокого обучения поможет вам повысить точность ваших прогнозов.

Мы будем использовать `keras` с бэкендом `tensorflow`, поэтому нужно будет установить обе библиотеки.

1. Как обычно, начинаем с загрузки необходимых библиотек и выполнения предварительной обработки данных временного ряда:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import LSTM, Dense
from sklearn.preprocessing import MinMaxScaler
```

```
# Загрузка данных временного ряда (замените название файла своим)
time_series_data = pd.read_excel('time_series_data.xlsx')

# Нормализация данных, чтобы они находились в диапазоне [0, 1]
scaler = MinMaxScaler()
data = scaler.fit_transform(time_series_data['Value'].to_numpy().
reshape(-1, 1))
```

2. Подготовив данные, мы разбиваем их на обучающий и тестовый наборы и изменяем их форму, чтобы подать на вход модели LSTM:

```
# Разбиение данных на обучающий и тестовый наборы
train_size = int(len(data) * 0.67)
train, test = data[0:train_size, :], data[train_size:len(data), :]

# Создание последовательностей и меток для обучения
def create_dataset(dataset, look_back=1):
    X, Y = [], []
    for i in range(len(dataset) - look_back):
        a = dataset[i:(i + look_back), 0]
        X.append(a)
        Y.append(dataset[i + look_back, 0])
    return np.array(X), np.array(Y)

look_back = 3
X_train, Y_train = create_dataset(train, look_back)
X_test, Y_test = create_dataset(test, look_back)

# Изменение формы данных для подачи их на вход модели LSTM
X_train = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
X_test = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))
```

3. Теперь, когда данные подготовлены, создадим и обучим модель LSTM:

```
model = Sequential()
model.add(LSTM(4, input_shape=(1, look_back)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(X_train, Y_train, epochs=100, batch_size=1, verbose=2)
```

   Этот код определяет простую модель LSTM, компилирует ее и обучает на данных обучающего набора. Вы можете настроить архитектуру модели и параметры обучения в соответствии с вашими потребностями. Обратите внимание: обучение этой модели занимает гораздо больше времени, чем обучение более простых моделей, о которых мы говорили выше.

4. Когда модель будет обучена, мы можем сделать прогнозы и масштабировать их, чтобы сравнить с исходным набором данных.

```
# Создание прогнозов:
trainPredict = model.predict(X_train)
testPredict = model.predict(X_test)

# Приведение прогнозов к исходному масштабу
trainPredict = scaler.inverse_transform(trainPredict)
testPredict = scaler.inverse_transform(testPredict)
```

5. Наконец, мы можем визуализировать фактический набор данных и прогнозы, сделанные на тестовом наборе:

```
# Создание графика прогнозов, сделанных на обучающих данных
trainPredictPlot = np.empty_like(data)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(trainPredict) + look_back, :] = \
    trainPredict

# Создание графика прогнозов, сделанных на тестовых данных
testPredictPlot = np.empty_like(data)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(trainPredict) + (look_back * 2):len(data), :] = \
    testPredict

# Выделение фактических данных синим цветом
plt.plot(time_series_data['Value'], color='blue', label='Actual Data')

# Создание заполненных цветом областей обучающих и тестовых данных
plt.fill_between(range(len(data)), 0,
    trainPredictPlot.reshape(-1),
    color='lightgray', label='Training Data')
plt.fill_between(range(len(data)), 0,
    testPredictPlot.reshape(-1),
    color='lightcoral', label='Test Data')

# Выделение прогнозов красным цветом
plt.plot(testPredictPlot, color='red', label='Predictions')

plt.title('Time Series Analysis with LSTM')
plt.legend()
plt.show()
```

   Полученные результаты являются весьма впечатляющими для такого небольшого набора данных и простой модели (рис. 10.16).

Учтите, что мы рассмотрели простейший пример. Модели LSTM могут быть куда более сложными, и в зависимости от имеющихся у вас данных временного ряда вам могут понадобиться такие методы, как многослойная сеть LSTM, настройка гиперпараметров и т. д.

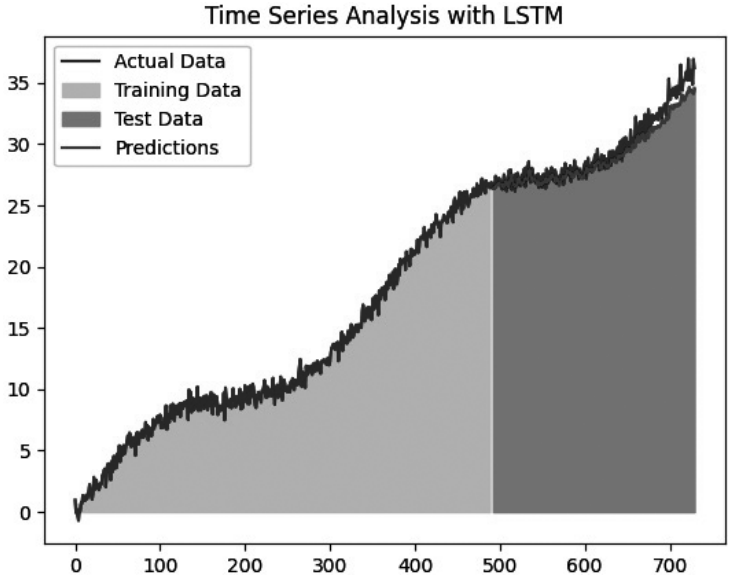


Рис. 10.16. Фактические данные (обозначены синим цветом) и прогнозы, сделанные на тестовом наборе (обозначены красным цветом)

На этом мы завершаем знакомство с темой анализа временных рядов с помощью Python.

Резюме

В этой главе вы освоили навыки анализа временных рядов, позволяющие выявлять скрытые закономерности и извлекать другую ценную информацию из темпоральных данных.

Мы начали с темы визуализации временных рядов, изучения способов создания основных диаграмм и понимания значения графиков АКФ/ЧАКФ.

Затем поговорили о статистике временных рядов, в том числе о тесте ADF, декомпозиции временных рядов и статистическом прогнозировании с помощью таких инструментов, как библиотеки `statsmodels` и `prophet`.

Чтобы повысить эффективность прогнозирования, мы обратились к методам глубокого обучения, в частности реализовали сеть LSTM с помощью библиотеки Python `keras`. Вы научились точно прогнозировать временные ряды и создавать информативные визуализации, помогающие принимать обоснованные решения.

Часть IV

Вызов кода R и Python из Excel

В этой части мы обсудим решения, разработанные для бесшовной интеграции кода R и Python с Excel. Вы научитесь использовать такие инструменты, как `BERT` и `xlwings`, для налаживания локального взаимодействия между Excel и предпочитаемым вами языком программирования. Кроме того, вы познакомитесь с такими решениями с открытым исходным кодом, как Plumber и FastAPI, а также с коммерческими решениями Posit Connect и ownR, которые позволят вам расширить возможности вашего рабочего процесса Excel с помощью API-интеграции.

11 Локальный вызов кода R/Python из Excel напрямую или через API

В этой главе мы рассмотрим способы вызова кода R и Python из Excel. Вы можете спросить, зачем это делать, если для решения задач можно использовать множество функций, встроенных в Excel, или написать VBA-макрос. Одна из причин для вызова кода R или Python из Excel заключается в том, что эти языки программирования можно использовать для анализа и визуализации данных. Excel — наиболее популярное приложение для работы с электронными таблицами, которое позволяет обрабатывать большие наборы данных, выполнять основные вычисления и применять распространенные функции. Однако оно имеет некоторые ограничения, когда речь заходит о таких более сложных или специализированных задачах, как статистическое моделирование, машинное обучение, веб-скрейпинг, обработка естественного языка и т. д. Вызывая код R или Python из Excel, вы можете получить доступ к библиотекам и пакетам этих языков и использовать более изощренные способы обработки, преобразования и визуализации данных. Кроме того, вы можете автоматизировать рабочие процессы, в том числе связанные с созданием отчетов и дашбордов. Вызов кода R или Python из Excel позволит повысить эффективность вашей работы, а также поможет расширить ваши аналитические возможности.

В этой главе мы рассмотрим следующие темы:

- причины для локального вызова кода R/Python из Excel;
- настройка среды;
- вызов кода R/Python напрямую из Excel;
- вызов кода Python из Excel с помощью xlwings;
- знакомство с API;

- решения с открытым исходным кодом, служащие для предоставления R и Python в качестве конечной точки API;
- вызов API из редактора VBA приложения Excel;
- плюсы и минусы решений, основанных на API;
- коммерческие API-решения для R и Python.

Технические требования

Файлы с кодом из примеров этой главы доступны по адресу https://github.com/PacktPublishing/Extending-Excel-with-Python-and-R/tree/main/Chapter%2011.

Для изучения материала главы вам потребуется установить следующее внешнее программное обеспечение:

- `BERT` (http://bert-toolkit.com/download-bert);
- Python и `xlwings` (см. раздел «Настройка среды» далее в главе);
- `plumber` для R;
- `FastAPI` для Python.

Начнем с обсуждения локальных вызовов.

Причины для локального вызова кода R/Python из Excel

Как было сказано во вступительной части, с помощью VBA в Excel можно решать самые разнообразные аналитические и программные задачи. Однако написание соответствующего кода может быть слишком утомительным и сложным занятием. Благодаря таким инструментам, как `BERT` и `xlwings`, вы можете воспользоваться уже готовым набором функций или написать собственные и использовать их в Excel.

С помощью `BERT` вы можете задействовать широкий спектр возможностей такого мощного языка статистического программирования, как R, непосредственно в приложении Excel, то есть не переключаясь на среду R. Это может быть очень удобно, если вы уже работаете в Excel и не хотите покидать эту программу.

Инструмент `BERT` позволяет писать функции на языке R, которые можно применять в Excel в качестве пользовательских функций. Такая возможность может быть актуальной, если нужно создавать функции, недоступные в Excel, или улучшать производительность функций, которые уже есть в этой программе. Например, вы

можете смоделировать броуновское движение, используя BERT для вызова функции, созданной в R, что гораздо проще сделать в среде этого языка, чем в Excel.

Что же касается пакета xlwings, то он предоставляет аналогичные преимущества в виде возможности создания решения на языке Python, которое можно вызывать непосредственно из Excel.

Настройка среды

Поскольку настройка среды для BERT и xlwings не является тривиальной задачей, то мы рассмотрим этот процесс более подробно.

Настройка BERT для R

В этом подразделе мы поговорим о том, как установить BERT в Windows, чтобы задействовать Excel из среды R. Сначала необходимо скачать программу установки BERT, доступную по адресу https://bert-toolkit.com/download-bert.

Затем нужно установить этот инструмент, как любую другую программу. После установки в разделе Надстройки ленты Excel появится кнопка BERT Console (рис. 11.1).

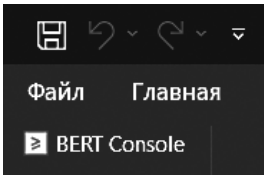


Рис. 11.1. Кнопка BERT Console в разделе Надстройки ленты Excel

Нажатие этой кнопки приведет к открытию консоли (рис. 11.2).

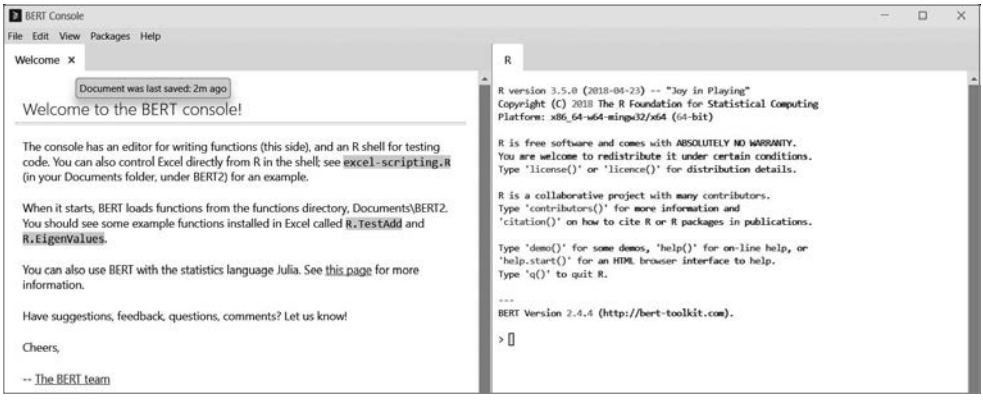


Рис. 11.2. Консоль BERT

Настройка xlwings для Python

В этом подразделе мы рассмотрим процесс настройки пакета `xlwings`.

Установка Python

Убедитесь в том, что Python установлен на вашем компьютере. Последнюю его версию можно скачать с официального сайта https://www.python.org/downloads/. Для облегчения доступа вы можете установить флажок Add Python to PATH в ходе инсталляции.

Установка надстроек Excel

Для установки пакета `xlwings` выполните следующие шаги.

1. Откройте командную строку и введите команду:

   ```
   pip install xlwings
   ```

   Вы можете получить предупреждение примерно следующего содержания: WARNING: The script xlwings.exe is installed in '<folder>' which is not on PATH. (Предупреждение: сценарий xlwings.exe установлен в '<папку>', которая отсутствует в PATH.) В этом случае вам придется указать полный путь, чтобы получить возможность вызвать `xlwings` на следующем шаге.

   Чтобы установить надстройку, воспользуйтесь клиентом командной строки:

   ```
   xlwings addin install (Re)start Excel
   ```

2. В Excel откройте редактор VBA, затем перейдите в меню Сервис ▸ Ссылки и установите флажок рядом с пунктом xlwings. Нажмите кнопку OK и вернитесь в Excel. Этот шаг не является обязательным; перед его выполнением проверьте доступность пакета `xlwings` на ленте после перезапуска Excel.

   Надстройка `xlwings` должна появиться справа от меню Справка на панели инструментов (рис. 11.3).

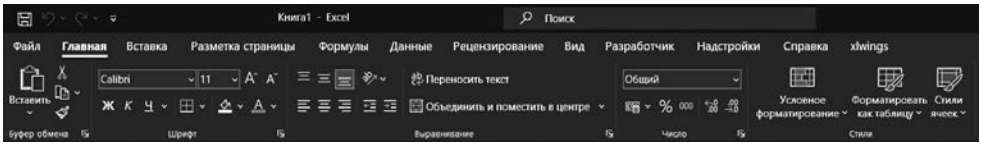


Рис. 11.3. Меню xlwings на ленте Excel

Настройка среды Excel и Python

Настройте `xlwings`, указав путь к интерпретатору Python в параметрах надстроек Excel. Обычно `xlwings` делает это автоматически.

Чтобы убедиться в правильности настройки, выполните любой из примеров кода, приведенных в следующем разделе.

Устранение неполадок

Если у вас возникнут проблемы, то в документации по `xlwings` (https://docs.xlwings.org/en/stable/) вы можете прочесть рекомендации по устранению неполадок.

Теперь, когда мы настроили все инструменты, поговорим о том, как их использовать.

Вызов кода R/Python напрямую из Excel

В этом разделе мы рассмотрим способы вызова кода R и Python из Excel с помощью настроенных ранее инструментов. Обсудим несколько способов решения этой задачи и приведем примеры, на которых вы сможете их опробовать.

Выполнение кода R с помощью макроса VBA и BERT

Вызвать код R из Excel можно с помощью макроса VBA. Для этого рабочую книгу необходимо сохранить в качестве рабочей книги с поддержкой макросов. Инструмент `BERT` предназначен для вызова кода R из Excel, поэтому выражение на языке R может быть написано в консоли VBA и вызвано с помощью следующего VBA-кода:

```
Application.Run "BERT.Exec", r
```

Рассмотрим простой пример:

```
Sub PlotNormalDensity()
    r = "plot(density(rnorm(1000)))"
    Application.Run "BERT.Exec", r
End Sub
```

Результатом выполнения этого кода является график плотности случайного нормального распределения (рис. 11.4).

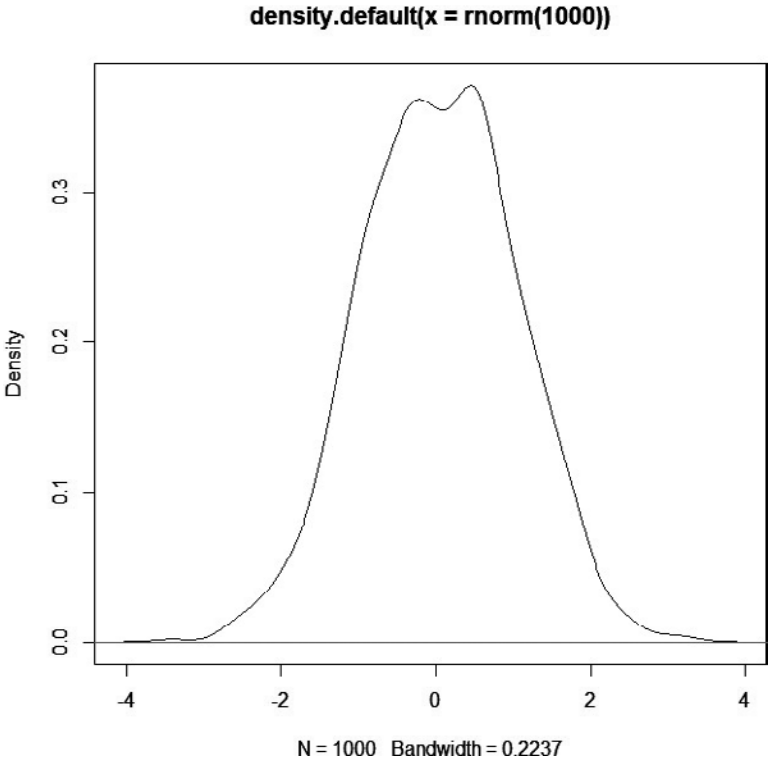


Рис. 11.4. Использование BERT для вызова кода R из VBA

Взаимодействие с Excel через BERT

Вы можете задействовать Excel с помощью `BERT`, используя интерфейс *Excel Scripting Interface*. Зачем может понадобиться такой вариант? Как вы помните, `BERT` позволяет получить доступ не только к Excel, но и к среде R. Это означает, что с помощью функций R, отсутствующих в Excel, вы можете генерировать данные. Рассмотрим пример.

Для начала определим диапазон ячеек:

```
rng <- EXCEL$Application$get_Range( "A1:B4" )
```

Этот код задает диапазон ячеек `A1:B4` и помещает его в глобальную среду как переменную `rng`. Код был введен в интерфейсе R инструмента `BERT`. Теперь, когда

диапазон определен, мы можем выяснить, сколько команд для работы с диапазонами есть в нашем распоряжении и что именно они собой представляют.

```
> length(ls(rng))
[1] 234
> head(ls(rng))
[1] "Activate"              "AddComment"             "AddCommentThreaded"
[4] "AdvancedFilter"        "AllocateChanges"        "ApplyNames"
```

Итак, мы видим, что нам доступны 234 команды для работы с диапазонами. При вызове `ls(rng)` в консоли R отображаются все команды; в данном случае мы использовали `head()`, чтобы отобразить лишь первые несколько. Далее мы используем команду Excel `RAND()`, чтобы поместить случайные числа в заданный диапазон ячеек (рис. 11.5):

	A	B
1	0.634742	0.799366
2	0.5541	0.413433
3	0.869921	0.794826
4	0.071008	0.92193

Рис. 11.5. Применение функции Excel RAND() из BERT

```
rng$put_Formula("=RAND()");
```

Если мы хотим остаться в среде `BERT` и увидеть значения, которые были отправлены в Excel, то можем сделать следующее:

```
> rng$get_Value()
                      [,1]           [,2]
[1,] 0.63474248 0.7993663
[2,] 0.55409965 0.4134328
[3,] 0.86992109 0.7948257
[4,] 0.07100827 0.9219299
```

Теперь, когда вы познакомились с основами работы с Excel через R, посмотрим, как можно добиться аналогичных результатов в случае с Python.

Вызов кода Python из Excel с помощью xlwings

Вызвать код Python из Excel можно с помощью `xlwings`, используя три варианта:

- кнопку `Выполнить` на вкладке `xlwings` ленты Excel;
- макросы, вызывающие код Python из Excel;
- *функции, определяемые пользователем* (User Defined Functions, UDF) (только в Windows).

Рассмотрим плюсы и минусы каждого из этих вариантов на конкретных примерах.

Кнопка Выполнить

Эта кнопка ожидает наличия функции `main` в модуле Python, имя которого совпадает с именем вашей рабочей книги. Это жесткое условие прописано в документации. Главное преимущество данного подхода заключается в отсутствии кода VBA и макросов. Вы можете использовать обычный XLSX-файл, что бывает очень полезно в ситуациях, когда использование XLSM-файлов запрещено по соображениям безопасности.

Ниже описаны действия, которые необходимы для запуска кода Python с помощью кнопки Выполнить.

1. Создайте модуль Python под названием `sumitup.py`, содержащий следующий код:

```
import xlwings as xw

def main():
            wb = xw.Book.caller()
            a = wb.sheets[0]['A1'].value
            b = wb.sheets[0]['A2'].value
            wb.sheets[0]['A3'].value = a + b
            pass
```

2. Откройте приложение Excel и заполните ячейку A1 значением `2`, а ячейку A2 — значением `3`.
3. Сохраните лист Excel под именем `sumitup.xlsx` в той же папке, в которой был сохранен файл `sumitup.py`.
4. Нажмите кнопку Выполнить на вкладке `xlwings` ленты Excel.

У этого варианта есть минусы: строгие правила именования и размещения файлов, а также отсутствие гибкости. При использовании данного метода будет вызываться только функция `main()`, кроме того, в коде Python должно быть прописано, какие поля будут использоваться в качестве входных данных и куда будут направляться выходные. Это означает невозможность передачи входных данных в функцию со стороны Excel.

Макросы

Если вам требуется бóльшая гибкость и у вас есть возможность использовать макрос (и сохранить файл в формате XLSM), то вы можете запустить команду `RunPython` из редактора VBA.

В этом примере `RunPython` импортирует модуль `world` и запустит функцию `hello-world()` из этого модуля. Чтобы выполнить этот пример, проделайте следующее.

1. Создайте файл `.xlsm` и сохраните его под именем `world.xlsm`.
2. Откройте редактор VBA и выполните следующие действия.
   - Перейдите в меню Сервис ▸ Ссылки и убедитесь в том, что рядом с пунктом xlwings установлен флажок (рис. 11.6).

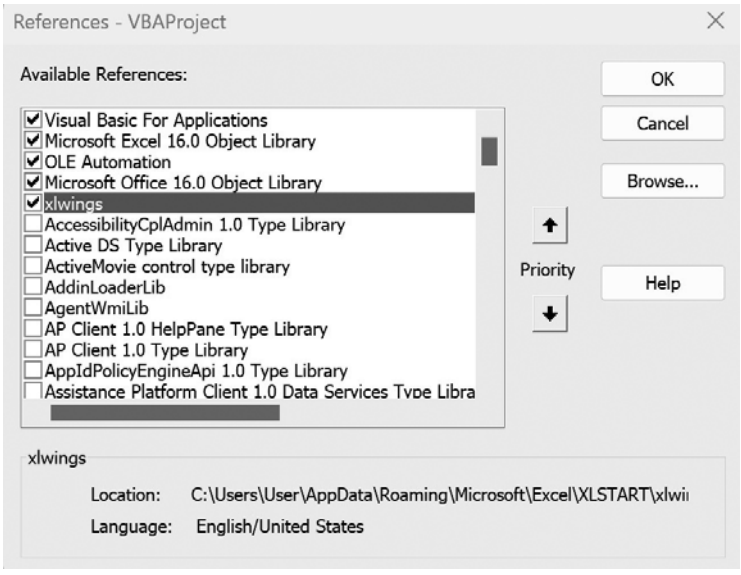


Рис. 11.6. Добавление ссылки на xlwings в проект VBA

   - Создайте в файле новый макрос с помощью оператора `Sub` со следующим кодом:

```
Sub HelloWorld()
    RunPython "import world; world.helloworld()"
End Sub
```

3. Создайте модуль `hello.py` с показанным ниже содержимым в той же папке, где вы сохранили файл `.xlsm`:

```
import xlwings as xw

def helloworld():
    wb = xw.Book.caller()
    wb.sheets[0]['A1'].value = 'Hello World!'
```

4. Теперь вы можете запустить макрос в файле `world.xlsm` и оценить результаты.

Как видите, использование макросов устраняет один из недостатков описанного ранее варианта, связанный со строгими правилами размещения файлов и именования используемых модулей и функций Python. Однако он тоже не позволяет передавать входные данные непосредственно из Excel; входные данные должны находиться в предварительно заданных ячейках файла Excel, и выходная ячейка (ячейки) тоже должна быть заранее определена.

Обратите внимание: вы можете изменить как имя модуля (чтобы оно не совпадало с именем файла Excel), так и его расположение (то есть сохранить его в папке, отличающейся от той, в которой хранится файл `.xlsm`). В документации по `xlwings` говорится: *«По умолчанию RunPython ожидает, что сценарий world.py находится в том же каталоге, что и одноименный файл Excel, но вы можете изменить и то и другое: если ваш файл Python находится в другой папке, то добавьте эту папку в PYTHONPATH в конфигурации. Если файл имеет другое имя, то измените команду RunPython соответствующим образом»*.

Функции, определяемые пользователем

Функции, определяемые пользователем (User Defined Functions, UDF), которые можно использовать так же, как `SUM()` и `IF()`, позволяют расширить возможности Excel.

Написать такую функцию на языке Python с помощью `xlwings` можно следующим образом:

```
import xlwings as xw
import numpy as np

@xw.func
@xw.arg('x', np.array, ndim=2)
@xw.arg('y', np.array, ndim=2)
def matrix_add(x, y):
    return x + y
```

Здесь мы указали, что в качестве входных данных используются экземпляры массивов `numpy`, и определили функцию для сложения матриц.

Чтобы интегрировать функции Python в Excel с помощью `xlwings`, необходимо определенным образом настроить исходный файл Python. По умолчанию `xlwings` ожидает, что исходный файл Python находится в том же каталоге, что и файл Excel, и имеет то же имя, что и этот файл, только с расширением `.py` вместо `.xlsm`.

В качестве альтернативы вы можете указать конкретный модуль в поле UDF Modules на вкладке xlwings ленты Excel.

Когда исходный файл Python будет подготовлен, вы можете импортировать функции в Excel, выполнив следующие действия.

1. Задайте для файла Excel имя, совпадающее с именем модуля Python, и сохраните его с расширением `.xlsm` в той же папке, в которой находится модуль Python.
2. В Excel перейдите на вкладку xlwings и нажмите кнопку Import Python UDFs, чтобы учесть изменения, внесенные в модуль Python.
3. Введите данные в два диапазона ячеек на листе Excel (убедитесь в том, что размерность диапазонов является допустимой для выполнения матричного умножения).
4. В ячейке введите формулу `=matrix_add([диапазон1], [диапазон2])`, где `[диапазон1]` и `[диапазон2]` — диапазоны ячеек, в которые вы ввели данные.

   После этого в ячейке должен отобразиться правильный результат (рис. 11.7).

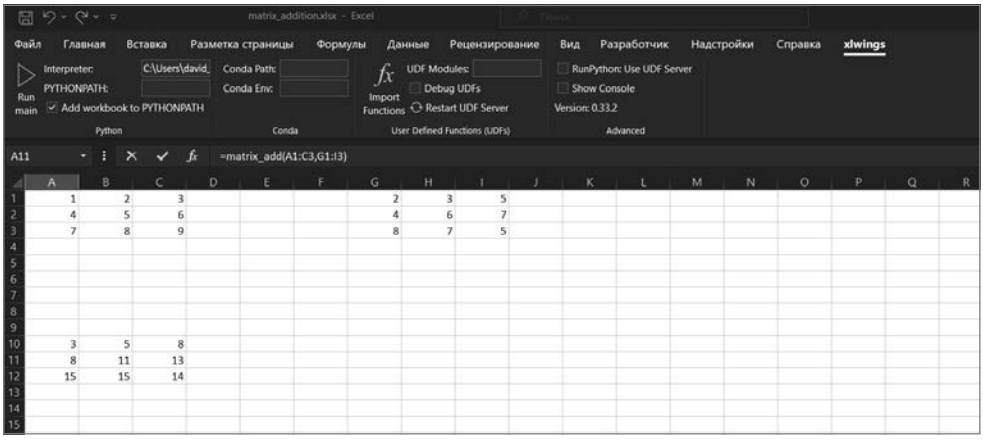


Рис. 11.7. Использование функции matrix_add из xlwings в Excel

Преимущество этого метода очевидно: пользователь Excel может определить, какие входные данные использовать и куда вывести результат, не взаимодействуя с кодом Python (или с его разработчиком).

Недостаток данного метода является общим для всех решений, подобных `xlwings`: для локальной установки Python (или R) необходимо корректно настроить все зависимости и сделать правильную версию доступной всем пользователям.

Обратите внимание: определяемые пользователем функции не работают с Python, который можно загрузить из Microsoft Store.

Следует отметить, что для `xlwings` существуют альтернативы, в частности PyXLL. Данная надстройка обеспечивает схожую функциональность и имеет аналогичные ограничения (в том числе невозможность работы с Python, установленным из Microsoft Store).

Итак, в предыдущих разделах вы узнали о том, что такое `BERT` и `xlwings` и как с помощью этих инструментов можно вызвать код R и Python непосредственно из Excel.

В следующей части главы мы рассмотрим другой способ реализации подобной интеграции, не требующий локальной установки и полного контроля над кодом и его версией (что может быть актуально в корпоративных условиях), но предполагающий использование серверной среды.

Знакомство с API

Интерфейс прикладного программирования (Application Programming Interface, API) позволяет различным приложениям определенным образом взаимодействовать и обмениваться данными. Он подобен официанту в ресторане, который принимает ваш заказ, передает его на кухню и приносит вам готовое блюдо.

В цифровом мире API задает способ взаимодействия компонентов программного обеспечения. Он определяет методы и форматы данных, которые приложения могут использовать для запроса информации и обмена ею. С помощью API можно реализовывать различные цели, например получать доступ к веб-сервисам, базам данных и даже аппаратным устройствам. В качестве примера использования API можно привести приложение на смартфоне, которое соединяется с метеорологической системой, чтобы получить сведения о текущих погодных условиях или ее прогноз.

Один из основных вариантов применения API сводится к предоставлению сторонним разработчикам возможности интегрировать свои приложения с существующими сервисами или платформами. В контексте программирования на языках R и Python с помощью API можно получать данные и обеспечивать взаимодействие между создаваемыми моделями и другими программами, в том числе Excel.

К числу достоинств API можно отнести их универсальность. Они абстрагируют сложность базовых систем, предоставляя разработчикам стандартизированный способ взаимодействия с ними. Подобная абстракция позволяет упростить процессы разработки и интеграции, способствуя сотрудничеству и инновациям.

API можно представить не только в качестве механизма обеспечения взаимодействия различных систем, но и в качестве контракта. В документации API указывается, как именно система может подключаться и взаимодействовать с этим интерфейсом, что ей разрешено делать и как часто.

Системы, поддерживающие API, как правило, взаимодействуют по принципу клиента и сервера. В настоящее время одним из самых популярных типов API является *REST* (representational state transfer) — прикладной программный интерфейс передачи репрезентативного состояния. Основное его преимущество заключается в отсутствии состояний. Это означает, что серверы не сохраняют данные клиента между запросами. Отправляемые на сервер запросы напоминают URL. Типичный вызов REST API выглядит следующим образом:

```
GET https://example.com/api/v1/users
```

Вы можете рассматривать предыдущий фрагмент кода в качестве GET-запроса, результатом которого будет получение списка пользователей. А вот как выглядит типичный POST-запрос:

```
POST https://example.com/api/v1/users
Content-Type: application/json
{
    "name" : "John Doe",
    "email": "john.doe@example.com"
}
```

Здесь данные отправляются HTTP-методом POST на сервер. Таким образом, об использовании REST API можно думать как об отправке информации на сервер и ее получении от него.

Теперь, когда вы знаете, что такое API и зачем они нужны, мы можем перейти к рассмотрению решений с открытым исходным кодом, служащих для предоставления R и Python в виде их конечных точек.

Решения с открытым исходным кодом, служащие для предоставления R в качестве конечной точки API

В этом разделе мы обсудим способ предоставления R в качестве конечной точки API с помощью пакета `plumber`. Его и сопутствующую документацию можно найти по адресу https://www.rplumber.io/index.html.

Сначала мы создадим очень простой API с одним аргументом, чтобы получить гистограмму стандартного нормального распределения. Код выглядит так:

```
#* Создание графика на основе данных из случайного нормального распределения
#* @param .mean Среднее значение стандартного нормального распределения
#* @get /plot
#* @serializer png
function(.mean) {
    mu <- as.numeric(.mean)
    hist(rnorm(n = 1000, mean = mu, sd = 1))
}
```

Строки, начинающиеся с `#*`, представляют собой комментарии. В API `plumber` эти комментарии используются в качестве документации. В них содержатся описание того, что делает конечная точка API, и информация о параметрах. В первом комментарии сообщается о назначении конечной точки API и говорится о том, что этот API будет генерировать график на основе данных из случайного нормального распределения. В следующей строке описывается параметр `.mean`, представляющий среднее значение стандартного нормального распределения. Параметры в API `plumber` являются чем-то вроде входных данных, которые можно передать API при выполнении запроса.

В комментарии `#* @get /plot` указывается, что доступ к этой конечной точке API можно получить с помощью HTTP-запроса `GET`, а путем к ней является `/plot`. Проще говоря, для использования этого API вы должны отправить запрос наподобие `http://your-api-url/plot`. Функция определяется следующим образом: `function(.mean)`. С этого начинается функция R, принимающая параметр `.mean`, который представляет собой среднее значение стандартного нормального распределения. Чтобы иметь возможность передать этот аргумент непосредственно в функцию `rnorm()`, мы должны объявить его в виде числового типа данных. Это делается следующим образом: `mu <- as.numeric(.mean)`. Данный фрагмент кода преобразует параметр `.mean` в числовое значение и сохраняет его в переменной `mu`. После преобразования аргумента в числовое значение мы можем передать его в функции `rnorm()` и `hist()` для создания графика.

Наконец, мы подходим к генерации данных и созданию графика на их основе. Это делается с помощью фрагмента кода `hist(rnorm(n = 1000, mean = mu, sd = 1))`,

который генерирует `1000` случайных чисел из нормального распределения с заданными средним значением `(mu)` и стандартным отклонением `(sd = 1)`, а затем создает гистограмму на их основе. По сути, все сводится к генерации случайных данных и созданию графика.

Таким образом, после развертывания этот API `plumber` создает конечную точку `(/plot)`, которая при обращении к ней генерирует гистограмму на основе случайных данных из нормального распределения. В качестве параметра, который вы указываете при запросе к этому API, используется среднее значение. Этот простой, но показательный пример демонстрирует, как можно использовать R и `plumber` для предоставления возможностей обработки данных через Интернет.

Теперь, когда мы сгенерировали код, отвечающий за создание API, оценим результат его выполнения. Для этого рассмотрим следующий сценарий:

```
# Загрузка библиотеки
library(plumber)

# Указание пути к каталогу и файлу
wd <- getwd()
sub_dir <- paste0("/Chapter 11/")
full_dir <- paste0(wd, sub_dir)
f <- "plumber_api.R"
f_path <- paste0(full_dir, f)

# Инициализация объекта root
root <- pr(f_path)
root

# Маршрутизатор plumber с 1 конечной точкой, 4 фильтрами и 0 субмаршрутизаторов

# Используйте функцию pr_run() на этом объекте, чтобы запустить API
├──[queryString]
├──[body]
├──[cookieParser]
├──[sharedSecret]
└──/plot (GET)
```

Теперь разберем этот фрагмент построчно.

Строка

```
# Загрузка библиотеки
library(plumber)
```

загружает пакет `plumber`, который необходим для создания соответствующих API.

В следующих строках задается путь к каталогу и файлу API `plumber`:

```
# Указание пути к каталогу и файлу
wd <- getwd()
sub_dir <- paste0("/Chapter 11/")
full_dir <- paste0(wd, sub_dir)
f <- "plumber_api.R"
f_path <- paste0(full_dir, f)
```

Функция `getwd()` возвращает рабочий каталог, то есть текущий каталог, который использует R. Функция `paste0()` применяется для конкатенации строк, поэтому переменная `sub_dir` содержит строку `/Chapter11/`, а переменная `full_dir` — путь к каталогу /Chapter12/. Переменная `f` содержит имя файла API `plumber`, то есть `plumber_api.R`. Переменная `f_path` содержит полный путь к файлу API `plumber`, который представляет собой комбинацию значений переменных `full_dir` и `f`.

Эта строка

```
# Инициализация объекта root
root <- pr(f_path)
```

инициализирует API `plumber`, вызывая функцию `pr()`, которой в качестве аргумента передается переменная `f_path`. Функция `pr()` считывает файл API `plumber` и создает объект.

Строка

```
root
```

просто возвращает переменную `root`, которая является объектом API `plumber`.

Таким образом, приведенный выше код на языке R создает API `plumber` из файла `plumber_api.R`, который затем можно использовать для предоставления функций R в качестве конечных точек.

Рассмотрим простую аналогию. Представьте, что у вас есть ресторан и вы хотите предложить услугу доставки. Вы можете создать простой сайт, через который клиенты смогут заказывать еду. Чтобы заказы можно было оформить, он должен взаимодействовать с кухней вашего ресторана. В данном случае API `plumber` соответствует сайту, а функции R — кухне ресторана. API `plumber` делает функции R доступными для внешнего мира, позволяя другим приложениям взаимодействовать с ними. Теперь запустим API и посмотрим, что произойдет; для этого мы используем следующий код:

```
pr_run(root)
```

В результате выполнения приведенного выше фрагмента кода мы увидим экран, позволяющий протестировать API (рис. 11.8).

Рис. 11.8. Экран API plumber

Давайте это и сделаем. Мы можем отправить GET-запрос к API, нажав кнопку GET. Откроется следующее окно (рис. 11.9).

Здесь мы можем ввести значение параметра `.mean`; введем `10`. Для этого сначала нажмите кнопку Try it out (Попробовать), а затем введите значение. После этого нажмите кнопку Execute (Выполнить) (рис. 11.10).

После нажатия кнопки Execute (Выполнить) мы получаем несколько результатов; одним из них является запрос `curl`:

```
curl -X GET "http://127.0.0.1:9220/plot?.mean=10" -H "accept: image/ png"
```

Далее мы получаем URL запроса:

```
http://127.0.0.1:9220/plot?.mean=10
```

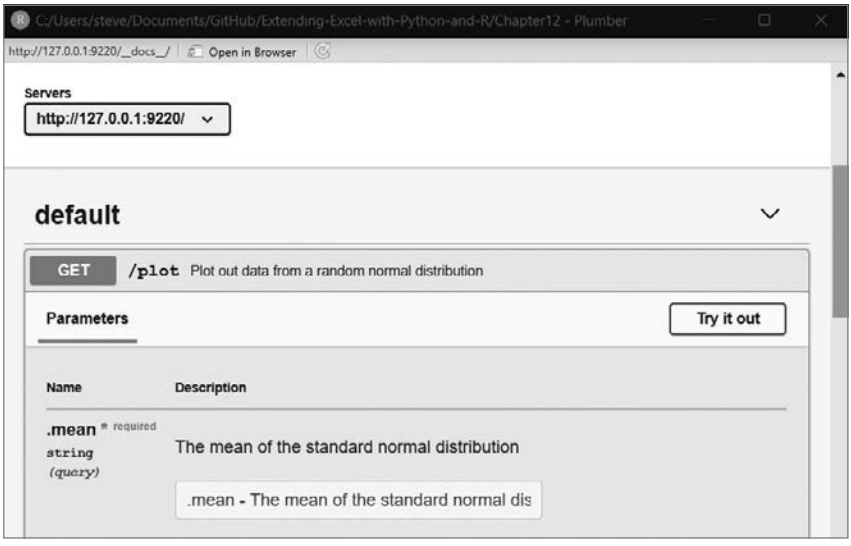


Рис. 11.9. Окно с полем для ввода аргументов

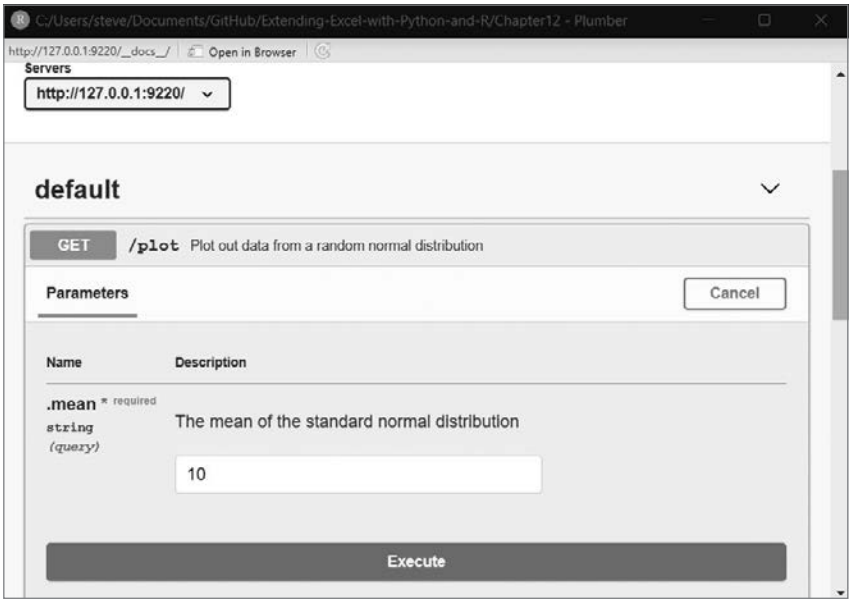


Рис. 11.10. Введите значение и нажмите кнопку Execute (Выполнить)

Наконец, мы получаем гистограмму (рис. 11.11).

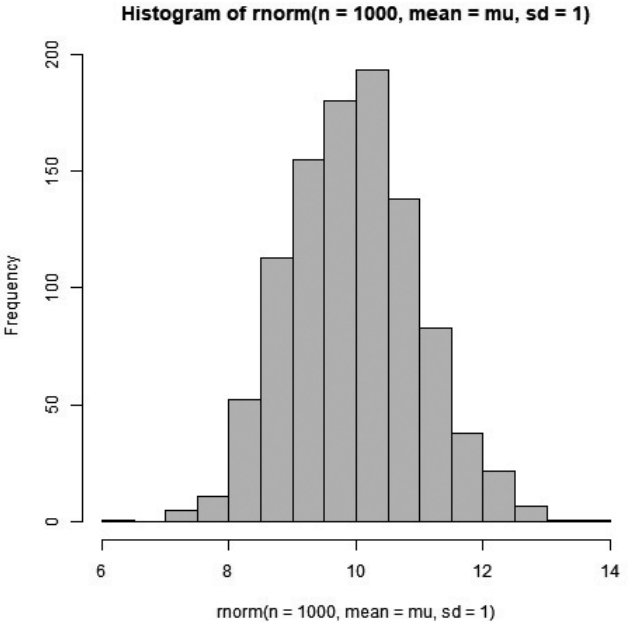


Рис. 11.11. Гистограмма, полученная в результате вызова API

Итак, в этом разделе вы научились создавать и использовать API для R, а в следующем вы узнаете, как сделать то же самое для Python.

Пример open-source-решения, служащего для предоставления Python в качестве конечной точки API

`FastAPI` представляет собой современный, быстрый (высокопроизводительный) веб-фреймворк для создания API с использованием Python 3.7+ на основе стандартных подсказок типов данных этого языка. Он прост в применении и позволяет быстро создавать надежные API.

Установить `FastAPI` можно с помощью менеджера пакетов `pip`:

```
pip install fastapi
```

Ниже приведен упрощенный пример создания конечной точки `FastAPI` для предоставления доступа к функции Python:

```
from fastapi import FastAPI, Query

app = FastAPI()
@app.get("/api/add")
def add_numbers(
    num1: int = Query(..., description="First number"),
    num2: int = Query(..., description="Second number"),
):
    result = num1 + num2
    return {"result": result}
```

В этом примере функция `add_numbers`, доступная в конечной точке `/api/add`, принимает два параметра запроса (`num1` и `num2`), представляющие числа, подлежащие сложению. Функция `Query` из `FastAPI` используется для определения этих параметров с необязательными описаниями. Результат вычисления возвращается в виде ответа в формате JSON.

С помощью приведенного выше кода вы можете отправить GET-запрос по адресу `/api/add?num1=3&num2=4`, чтобы получить результат. Параметры `num1` и `num2` указываются в строке запроса URL.

`FastAPI` автоматически генерирует документацию для OpenAPI и JSON Schema, что упрощает пользователям процесс взаимодействия с вашим API.

Чтобы запустить сервер разработки, выполните в командной строке такой код:

```
uvicorn your_app_name:app --reload
```

Замените `your_app_name` именем вашего файла Python, помня о том, что имя файла Python должно отличаться от `fastapi` или быть похожим на имя ранее установленного модуля.

ПРИМЕЧАНИЕ

Убедитесь в том, что ваш API защищен, особенно если к нему можно получить доступ через Интернет. FastAPI предоставляет функции для реализации процессов аутентификации и авторизации.

Итак, мы рассмотрели способы создания API для R и Python. Пришло время вызвать созданный нами API из Excel.

Вызов API из редактора VBA приложения Excel

В этом разделе мы рассмотрим код, который позволит нам использовать запрос `curl` для получения изображения от API, сгенерированного из файла `plumber_api.R`. Для этого вам нужно будет запустить код из предыдущего раздела: `root |> pr_run()`; этот фрагмент откроет диалоговое окно Swagger и предоставит URL, запрошенный из `plumber`. В нашем случае он выглядит так: `http://127.0.0.1:6855`.

Мы рассмотрим выполнение GET-запроса с помощью команды `curl` в редакторе VBA. Ниже приведен полный код:

```
Sub MakeCurlRequestAndInsertImage()

    ' Определение команды curl
    Dim curlCommand As String
    curlCommand = "curl -X GET ""http://127.0.0.1:6855/plot?. mean=0"" -H
    ""accept: image/png"" -o " & Environ("TEMP") & "\temp_image.png"

    ' Запуск команды curl с помощью оболочки Shell
    Shell "cmd /c " & curlCommand, vbHide

    ' Создание нового рабочего листа или обращение к существующему (Sheet1)
    Dim ws As Worksheet
    Set ws = ActiveWorkbook.Worksheets("Sheet1")

    ' Очистка листа Sheet1 от имеющегося содержимого
    ws.Cells.Clear

    ' Вставка изображения в рабочий лист
    ws.Pictures.Insert(Environ("TEMP") & "\temp_image.png").Select
End Sub
```

Теперь разберем этот код.

Мы начинаем с определения команды `curl`:

```
Dim curlCommand As String
curlCommand = "curl -X GET ""http://127.0.0.1:6855/plot?.mean=0"" -H \
  ""accept: image/png"" -o " & Environ("TEMP") & "\temp_image.png"
```

Здесь создается переменная (`curlCommand`) для хранения инструкции командной строки, которая приказывает компьютеру использовать `curl` (инструмент команд-

ной строки для выполнения HTTP-запросов) для получения изображения с указанного веб-адреса (`http://127.0.0.1:6855/plot?.mean=0`). Все это записывается в одну строку в редакторе VBA. Флаг `-H` указывает HTTP-заголовок, в данном случае говорящий серверу о том, что мы принимаем изображение в формате PNG. Флаг `-o` определяет выходной файл `temp_image.png` для сохранения изображения во временной папке.

Далее мы запускаем команду `curl` с помощью оболочки `Shell`:

```
Shell "cmd /c " & curlCommand, vbHide
```

Эта строка выполняет команду `curlCommand` с помощью командной строки ОС Windows (`cmd /c`). Фрагмент `vbHide` в конце означает, что окно командной строки не будет видно во время выполнения.

```
' Создание нового рабочего листа или обращение к существующему (Sheet1):
Dim ws As Worksheet
Set ws = ActiveWorkbook.Worksheets("Sheet1")
```

В этой части создается переменная (`ws`) для представления рабочего листа в Excel. Она либо создает новый рабочий лист `"Sheet1"`, либо ссылается на существующий лист активной рабочей книги.

Далее выполняется очистка листа `Sheet1`:

```
ws.Cells.Clear
```

Эта строка удаляет все содержимое, существующее в указанном рабочем листе (`"Sheet1"`).

После этого выполняется вставка изображения в рабочий лист:

```
ws.Pictures.Insert(Environ("TEMP") & "\temp_image.png").Select
```

Эта строка вставляет изображение, загруженное командой `curl`, в заданный рабочий лист (`"Sheet1"`). Фрагмент `Environ("TEMP")` указывает на временную папку в системе. `"\temp_image.png"` — это имя файла изображения, загруженного командой `curl`.

Итак, приведенный выше код VBA автоматизирует процесс использования команды `curl` для загрузки изображения с указанного веб-адреса, а затем вставляет это изображение в рабочий лист Excel `Sheet1`, который перед этим очищается от всего содержимого.

Теперь посмотрим на результат работы данного сценария VBA (рис. 11.12).

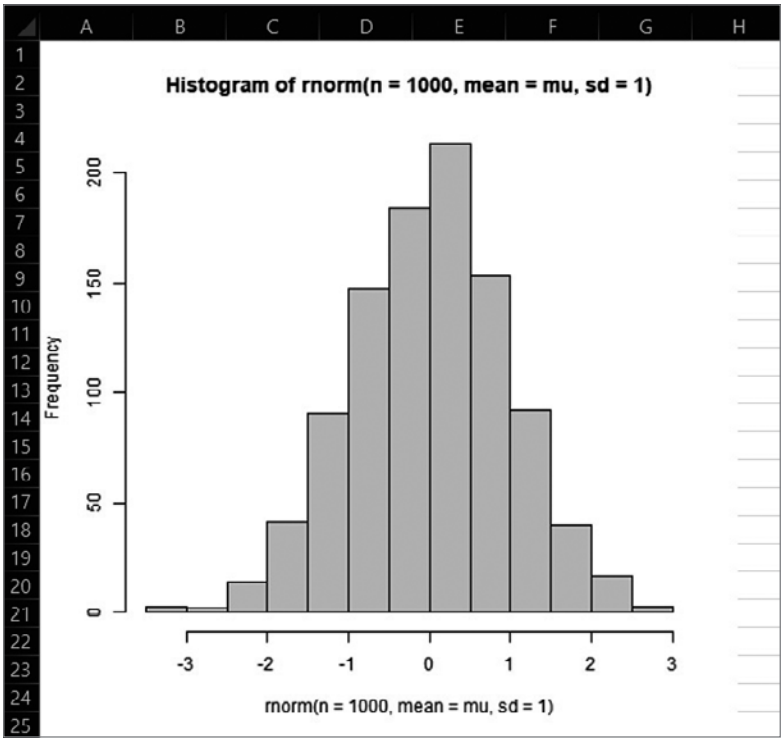


Рис. 11.12. Гистограмма, сгенерированная кодом R, запущенным из редактора VBA

Итак, теперь вы знаете, как создать API с помощью R-пакета `plumber` и вызвать его с помощью `curl`-запроса внутри VBA-макроса Excel.

Плюсы и минусы решений, основанных на API

К числу плюсов решения, основанного на API, можно отнести тот факт, что приложению Excel (и конечному пользователю) необязательно знать о том, что код, лежащий в основе API, написан на языке R. Тот же макрос можно использовать для вызова API, написанного на Python, не внося в него никаких изменений.

Кроме того, среди плюсов можно назвать и отсутствие локальной установки и необходимости поддерживать среду R/Python для каждого пользователя, а также

полный контроль над кодом, лежащим в основе API. Если вам понадобится обновить что-либо, начиная с версии R/Python через зависимости пакета и заканчивая самим кодом, то вы сможете сделать это на стороне сервера, не вовлекая в процесс конечного пользователя.

Недостаток, являющийся общим для всех решений на основе API, заключается в необходимости настройки и обслуживания сервера, что обычно является слишком сложным и недопустимым для аналитиков и специалистов по работе с данными. Это означает, что придется задействовать специализированные ИТ-ресурсы.

Кроме того, есть проблема, которая характерна для свободно распространяемых решений для API-хостинга и является общей для всех решений с открытым исходным кодом: экономя на лицензионных платежах, вы вынуждены очень многое делать самостоятельно. API-решения с открытым исходным кодом предполагают, что вы будете сами заниматься безопасностью (см. примечание в конце раздела, посвященного `FastAPI`), интеграцией с корпоративными системами (в частности, управлением пользователями) и многим другим.

Наконец, при работе как с `plumber`, так и с `FastAPI` приходится писать пользовательский код, позволяющий определять конечные точки и функциональность API. Для одного человека это не является большой проблемой, если не считать потраченных на работу часов. Но в больших командах часто могут возникать сложности, связанные с обеспечением стандартизации и соблюдением соглашений об именовании.

Коммерческие API-решения для R и Python

Если сравнивать с решениями с открытым исходным кодом, то у коммерческих решений есть ряд преимуществ: более высокий уровень оказываемой поддержки в таких вопросах, как обеспечение безопасности; более глубокая и простая интеграция с обширным корпоративным ландшафтом; оказание конкретной помощи в случае возникновения проблем.

Однако у коммерческих решений есть и недостаток: необходимо выделять бюджет на лицензионные платежи. Компании, предоставляющие решения, стремятся устанавливать цены так, чтобы сокращение ручного труда и увеличение добавленной стоимости оправдывали затраты на использование их продуктов, однако процесс согласования бюджета никогда не бывает простым.

В этом разделе мы рассмотрим некоторые из наиболее распространенных коммерческих API-решений для R и Python. Они обеспечивают высокий уровень

функциональности и интеграции Python и R с Excel и другими фронтендами, удовлетворяя различные потребности аналитиков и специалистов по работе с данными.

Azure Functions (и аналогичные решения других крупных облачных провайдеров)

Azure Functions — сервис бессерверных вычислений, предлагаемый Microsoft Azure. Он позволяет запускать код, реагирующий на события, без явного предоставления или управления инфраструктурой. С его помощью вы можете создавать API, реагировать на события и решать различные задачи, используя широкий спектр поддерживаемых языков, в том числе Python и JavaScript.

- *Примеры использования*. Идеально подходит для создания легковесных, ориентированных на события API или микросервисов. Интеграция с другими службами Azure обеспечивает масштабируемость и гибкость.
- *Плюсы*. «Бессерверная» инфраструктура означает минимальную потребность в участии ИТ-отдела.
- *Минусы*. Ограниченная гибкость (например, в плане выбора версии R/Python) и сложность смены облачного провайдера.

Posit Connect

Posit Connect — коммерческое решение, предназначенное для обеспечения взаимодействия R и Python с Excel. Оно облегчает интеграцию аналитических инструментов и методов науки о данных непосредственно в рабочие книги Excel, позволяя пользователям задействовать возможности R и Python в рамках привычного интерфейса.

- *Примеры использования*. Подходит для организаций, в которых Excel является основным инструментом анализа данных, дополняя функционал этого приложения расширенными аналитическими методами.
- *Плюсы*. Глубоко интегрированная экосистема для решений Posit, в том числе RStudio, Quarto и Shiny Pro.
- *Минусы*. Использует `plumber` и `FastAPI/Flask`, поэтому требует написания пользовательского кода для определения API, что приводит к недостаткам, которые мы уже упоминали в разделе, посвященном API-решениям с открытым исходным кодом. Поддерживает только серверы Linux, как локальные, так и облачные.

Платформа ownR Infinity

ownR Infinity — универсальная платформа, разработанная для применения функциональных возможностей R и Python в удобной для пользователя среде. Она позволяет создавать персонализированные функции и модели, превращая их в доступные API и обеспечивая бесшовную интеграцию с Excel.

- *Примеры использования*. Подходит для организаций, которые активно используют R и Python для выполнения анализа данных. Идеально подходит для пользователей, желающих дополнить возможности Excel статистическими и аналитическими функциями.
- *Плюсы*. Предоставляет удобный интерфейс, позволяющий задействовать возможности R и Python даже тем пользователям, которые не имеют глубоких знаний в области программирования. Обеспечивает бесшовную интеграцию функций R и Python с различными средами.
- *Минусы*. Поддерживает только серверы Linux, как локальные, так и облачные.

Для ознакомления с примером использования API ownR из Excel вы можете скачать рабочую книгу, доступную по адресу https://bitbucket.org/functionalanalytics/examples/src/master/ccy%20converter%20Excel%20demo%20button.xlsm. В этом примере используется API для следующей кодовой базы: https://bitbucket.org/functionalanalytics/converter/src/master/.

Обратите внимание: в данном случае нет необходимости писать собственный код, помимо чистого кода на Python (или R), а также вручную добавлять средства обеспечения безопасности, развертывания и т. д., поскольку все это автоматически добавляется платформой ownR.

Резюме

В этой главе вы узнали о том, как вызывать код R и Python из Excel, чтобы расширять возможности конечных пользователей. Мы рассмотрели причины применения этого подхода и два разных способа его реализации: локальный вызов кода R и Python и использование конечной точки API.

Мы обсудили применение инструментов `BERT` и `xlwings` для локального вызова кода R и Python, начиная с настройки и тестирования среды путем создания решений на языках R и Python и заканчивая вызовом этих решений из Excel с помощью различных методов, предоставляемых `BERT` и `xlwings`, таких как VBA-код и функции, определяемые пользователем.

Далее мы поговорили о конечных точках API и преимуществах их использования для налаживания взаимодействия Python и R с Excel. Мы обсудили плюсы и минусы этого варианта, а затем рассмотрели инструменты с открытым исходным кодом и коммерческие решения для хостинга API. Мы представили вам два наиболее популярных решения с открытым исходным кодом: `plumber` для R и `FastAPI` для Python, а также коммерческие решения для использования R и Python в качестве конечных точек API.

Теперь мы можем переходить к последней части книги.

Часть V

Анализ и визуализация данных в Excel с помощью R и Python на конкретном примере

В этой части мы поговорим об анализе и визуализации данных в Excel с применением возможностей Python и R. На конкретном примере мы рассмотрим, как с помощью этих универсальных языков программирования можно извлекать ценную информацию, выполнять визуализацию данных и принимать обоснованные решения непосредственно в Excel.

12 Анализ и визуализация данных в Excel с помощью R и Python на конкретном примере

В этой заключительной главе мы проведем анализ полученных из Excel данных путем их *визуализации* и создания простой модели, после чего возвратим результаты в Excel. Это может быть актуально, когда данных очень много или сами вычисления лучше проводить вне Excel.

Мы начнем с импорта данных и их исследования с помощью визуализаций. В этой главе мы будем использовать набор данных `diamonds` из R-пакета под названием `ggplot2`. Мы выясним, как разные характеристики бриллианта влияют на его цену. После визуализации этих данных создадим простую модель для предсказания цены бриллианта на основе его характеристик.

В этой главе мы рассмотрим следующие темы:

- создание визуализаций с помощью R;
- создание простой модели машинного обучения с помощью R;
- создание визуализаций с помощью Python;
- создание простой модели машинного обучения с помощью Python.

Технические требования

В этой главе мы будем использовать следующие пакеты/библиотеки:

- `ggplot2 3.4.4`;
- `dplyr 1.1.4`;

- `healthyR` `0.2.1`;
- `readxl` `1.4.3`;
- `tidyverse` `2.0.0`;
- `janitor` `2.2.0`;
- `writexl` `1.5.0`;
- `healthyR.ai` `0.0.13`.

Создание визуализаций с помощью R

В этом разделе мы создадим несколько визуализаций данных и дадим их краткую интерпретацию. Для этого мы создадим две гистограммы с помощью базового функционала R, а также несколько графиков, используя библиотеку `ggplot2`.

Получение данных

Сначала необходимо загрузить библиотеки и получить данные. Мы, авторы книги, работаем в каталоге, специально созданном для данной книги, поэтому в качестве источника функции можем указать главу, в которой была написана функция `read_excel_sheets()?`; в вашем случае путь может быть другим.

Код выглядит так:

```
# Загрузка библиотек
library(ggplot2)
library(dplyr)
library(healthyR)
library(readxl)

# Указание источника функций
source(paste0(getwd(),"/Chapter1/excel_sheet_reader.R"))

# Чтение данных
file_path <- paste0(getwd(), "/Chapter12/")
df <- read_excel_sheets(
  filename = paste0(file_path, "diamonds_split.xlsx"),"),
  single_tbl = TRUE
)
```

Здесь мы просто загрузили в свою среду несколько библиотек, вызвали функцию чтения листов и считали наши данные. Мы загрузили функцию `read_excel_sheets()` в нашу среду с помощью команды `source()`. Важно отметить, что данные для этого раздела были экспортированы из библиотеки `ggplot2`. Если вы хотите

воссоздать их, обеспечив при этом работоспособность предыдущего и последующих фрагментов кода, то используйте код, приведенный ниже:

```
# Загрузка библиотек
library(tidyverse)
library(writexl)
library(janitor)

# Запись файла на диск
file_path <- paste0(getwd(), "/Chapter12/")

# Разбиение данных по параметру cut (огранка) и очистка имен списка
df_list <- split(diamonds, diamonds$cut) |>
  clean_names()

# Запись данных в файл xlsx
df_list |>
  write_xlsx(paste0(file_path, "diamonds_split.xlsx"))
```

Теперь, когда мы разобрались с процессом создания и чтения данных, займемся их визуализацией.

Визуализация данных

В этом подразделе мы будем создавать графики с помощью базового функционала R и библиотеки `ggplot2`. Приступим.

Создание визуализаций с помощью базового функционала R

Начнем с того, что создадим несколько гистограмм на основе значений в столбце `price` набора данных `diamonds`. Цена (`price`) — итоговая переменная, которую мы будем использовать в качестве предиктора в нашей модели в следующем подразделе. Для начала нам нужно создать вектор точек разбиения, который будет передан функциям создания гистограмм. Методам выбора оптимальной стратегии разделения данных на интервалы посвящено большое количество литературы. Цель этого разделения заключается в придании гистограмме той формы, которая наилучшим образом представляет данные. Это отдельная обширная тема, которую мы не будем рассматривать в текущей главе. В пакете `healthyR` есть функция `opt_bin()`, которая принимает столбец значений (`value`) и выдает тиббл, содержащий точки разбиения. Посмотрим, как она работает.

```
breaks <- tibble(x = df$price) |>
  opt_bin(x) |>
  pull(value)
head(breaks)
[1]  326.000 1130.217 1934.435 2738.652 3542.870 4347.087
```

В данном случае наша цель заключается в том, чтобы правильно зафиксировать плотность информации в данных. Базовая функция `hist()` вполне справляется с этой задачей даже при использовании стандартного метода разбиения. Однако мы сравним этот стандартный метод с оптимальным. Чтобы расположить получившиеся в итоге графики рядом, мы используем фрагмент кода `par(mfrow = c(1, 2))`:

```
par(mfrow = c(1, 2))
hist(df$price, main = "Price Histogram - Default binning", xlab = "Price", \
      ylab = "Frequency")
hist(df$price, breaks = breaks, main = "Price Histogram - Optimal binning", \
      xlab = "Price", ylab = "Frequency")
par(mfrow = c(1, 1))
```

Результат показан на рис. 12.1.

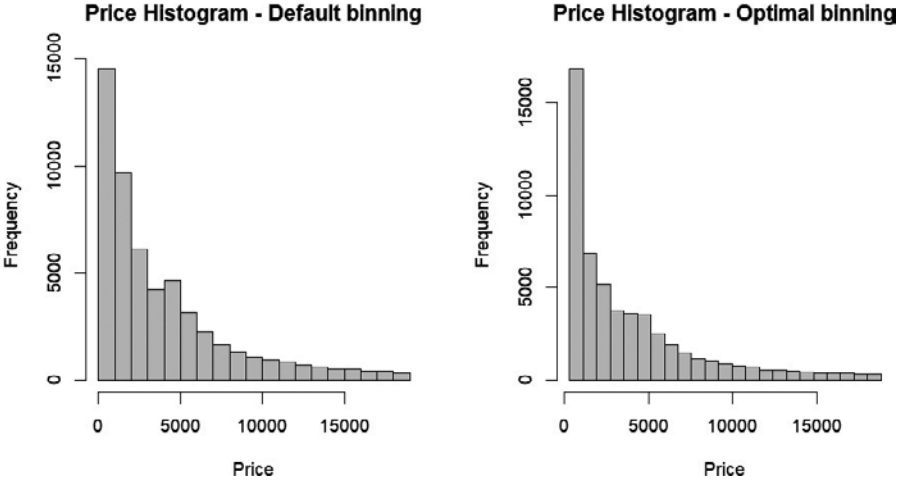


Рис. 12.1. Сравнение гистограмм, созданных с использованием стандартного *(слева)* и оптимального *(справа)* методов разбиения данных на интервалы

Как видите, формы гистограмм немного различаются. Но опять же, вы можете предпочесть другую стратегию разбиения данных. Этот пример лишь иллюстрирует существование различных методов.

Создание визуализаций с помощью библиотеки ggplot2

Остальные визуализации мы создадим с помощью библиотеки `ggplot2`, поскольку ее синтаксис нам кажется несколько более простым, а создаваемые с ее помощью графики — чуть более изощренными. Кроме того, этот пакет является частью коллекции `tidyverse`, а значит, способен взаимодействовать с другими входящими

в нее пакетами, такими как `dplyr`. Вам также может понадобиться установить пакет `hexbin`. Итак, приступим.

```
df |>
  ggplot(aes(x = carat, y = price, fill = cut)) +
  geom_hex(bins = length(breaks), alpha = 1/5) +
  facet_wrap(~ clarity, scales = "free") +
  theme_minimal() +
  labs(
      x = "Carat",
      y = "Price",
      title = "Diamonds Data",
      fill = "Cut"
  ) +
  hr_scale_color_colorblind()
```

Разберем этот код по частям.

Данные и визуальные параметры:

- строка `df |> ggplot(...)` запускает процесс визуализации данных, содержащихся в `df`;
- строка `aes(x = carat, y = price, fill = cut)` задает визуальные параметры графика:
  - `x` — по оси *X* откладывается вес в каратах;
  - `y` — по оси *Y* откладывается цена;
  - `fill` — цвет заливки обозначает уровень качества огранки.

Отображение точек данных:

- строка `geom_hex(bins = length(breaks), alpha = 1/5)` создает шестиугольники, представляющие точки данных;
- параметр `bins` задает количество интервалов (бинов) для создания гексагональной сетки. В данном случае используется значение, заданное для параметра `breaks` (который отсутствует в представленном коде);
- параметр `alpha` задает степень непрозрачности шестиугольников. Для обеспечения лучшей видимости мы используем 1/5.

Разбиение на фасеты (подграфики):

- строка `facet_wrap(~ clarity, scales = "free")` разделяет данные на подгруппы исходя из степени чистоты бриллиантов и создает для них отдельные графики с независимыми цветовыми шкалами.

Темы и подписи:

- строка `theme_minimal()` применяет минималистичную тему для получения максимально чистых и незагроможденных графиков;
- строка `labs(..., title = "Diamonds Data")` добавляет подписи осей и заголовок диаграммы.

Оптимизация цветовой шкалы для дальтоников:

- строка `hr_scale_color_colorblind()` гарантирует, что цветовая палитра будет нормально восприниматься людьми, страдающими дальтонизмом.

Результат представлен на рис. 12.2.

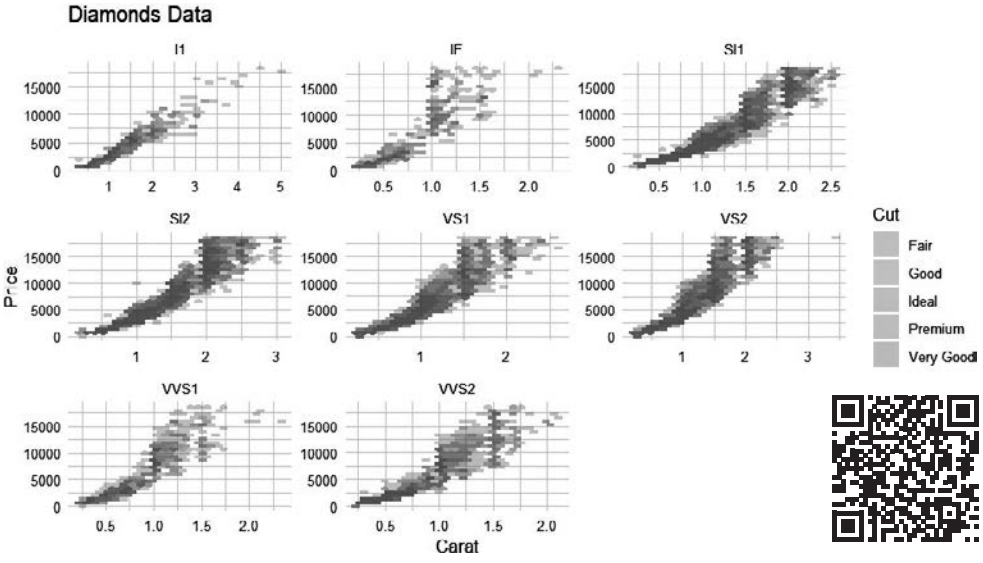


Рис. 12.2. Данные о бриллиантах, визуализированные с помощью ggplot2

Итак, приведенный выше код визуализирует взаимосвязь между весом бриллиантов в каратах, ценой и качеством огранки с разбивкой на группы по степени чистоты и использованием цветовой схемы, оптимизированной для дальтоников.

Следующей визуализацией будет диаграмма размаха, отражающая дисперсию данных.

```
df |>
  ggplot(aes(x = carat, y = price, fill = cut)) +
  geom_boxplot(alpha = 1/5, outlier.color = "lightgrey") +
  facet_wrap(~ clarity, scales = "free") +
  theme_minimal() +
  labs(
     x = "Carat",
     y = "Price",
     title = "Diamonds Data",
     fill = "Cut"
  ) +
  hr_scale_color_colorblind()
```

Результат выполнения этого кода показан на рис. 12.3.

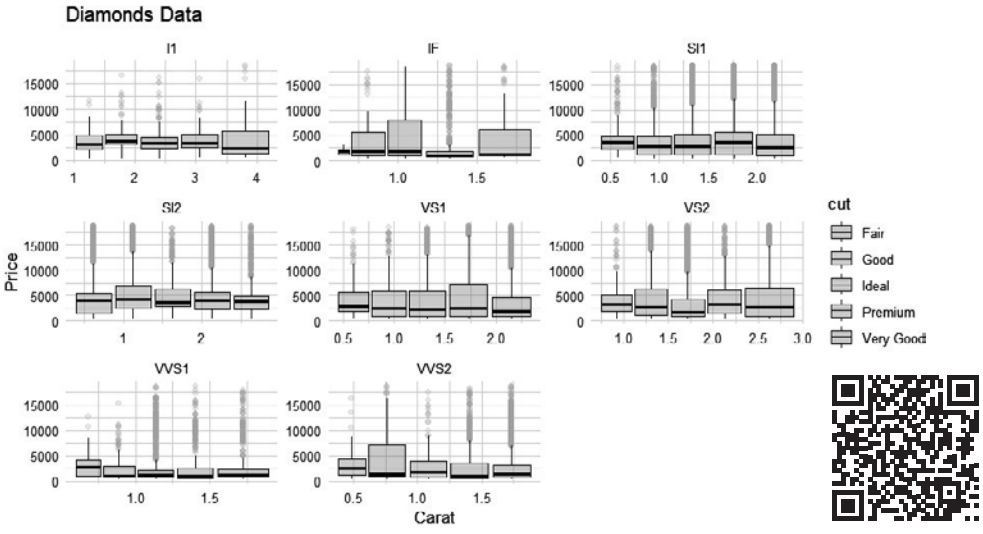


Рис. 12.3. Диаграмма размаха ggplot2, отражающая разброс цен

Теперь мы можем создать график средней цены и посмотреть, насколько точно он отражает информацию. Для этого мы используем следующий код:

```
df |>
  summarize(m = mean(price), .by = c(clarity, cut)) |>
  ggplot(aes(x = clarity, y = m, group = cut, color = cut)) +
  geom_point() +
  geom_line() +
  geom_smooth() +
  facet_wrap(~cut, ncol = 2) +
  labs(x= "Clarity",
         y = "Mean Price",
         title = "Mean Price by Clarity and Cut",
         color = "Cut") +
  theme_minimal() +
  hr_scale_color_colorblind()
```

Результат выполнения этого кода показан на рис. 12.4.

Вот еще один способ взглянуть на среднюю цену, на этот раз за карат:

```
df |>
  summarize(m = mean(price/carat), .by = c(cut, color, clarity)) |>
  ggplot(aes(x = color, y = m, group = clarity, color = clarity)) +
```

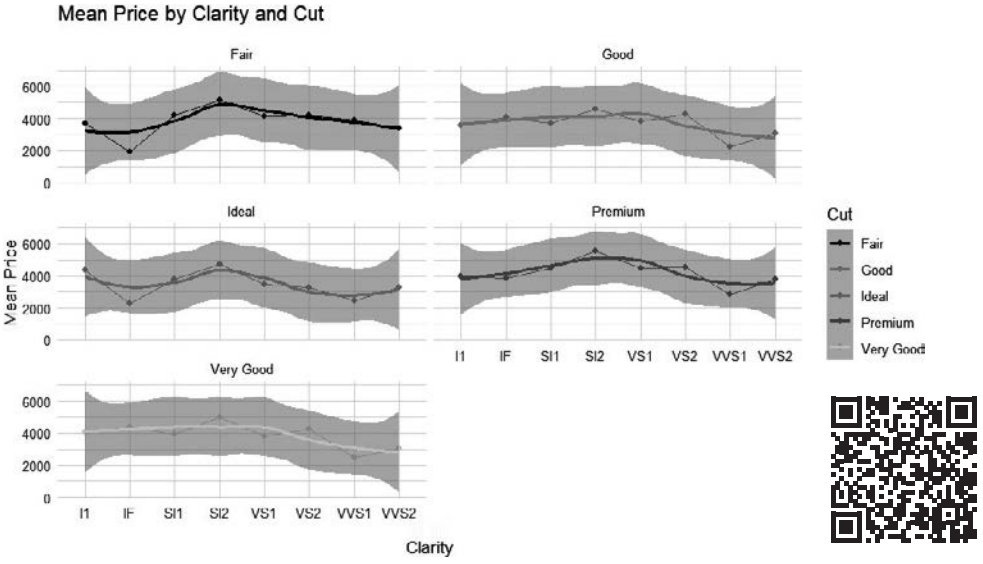


Рис. 12.4. График средних цен, созданный с помощью ggplot2

```
  geom_point() +
  geom_line() +
  facet_wrap(~ cut, ncol = 2, scales = "free") +
  labs(x= "Clarity",
         y = "Mean Price",
         title = "Mean Price per Carat by Clarity, Color and Cut",
         color = "Cut") +
  theme_minimal() +
  hr_scale_color_colorblind()
```

Посмотрим, о чем говорит этот показатель (рис. 12.5).

Это очень хорошие бриллианты. Судя по всему, качество огранки и цвет не имеют особого значения, если степень чистоты выше удовлетворительного (Fair).

Наконец, создадим гистограммы распределения цен, сгруппированных по уровню качества огранки, используя заданное ранее значение параметра `breaks` для разбиения данных на интервалы.

```
df |>
  ggplot(aes(x = price)) +
  geom_histogram(breaks = breaks, fill = "lightblue", color = "black") +
  theme_minimal() +
  facet_wrap(~ cut, ncol = 2, scales = 'free') +
  labs(x = "Price", y = "Frequency", title = "Price Histogram by Cut")
```

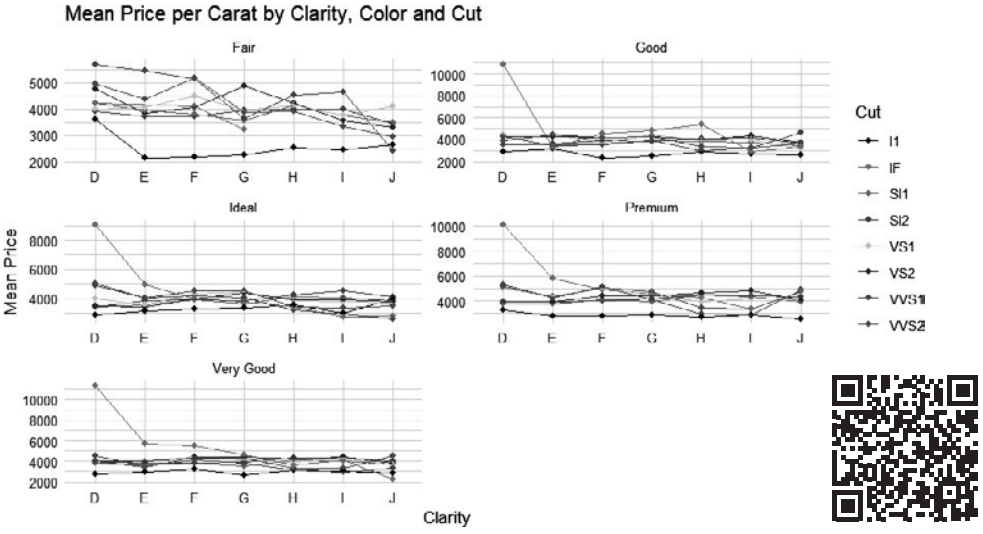


Рис. 12.5. График средних цен за карат, созданный с помощью ggplot2

Посмотрим, что получилось в итоге (рис. 12.6).

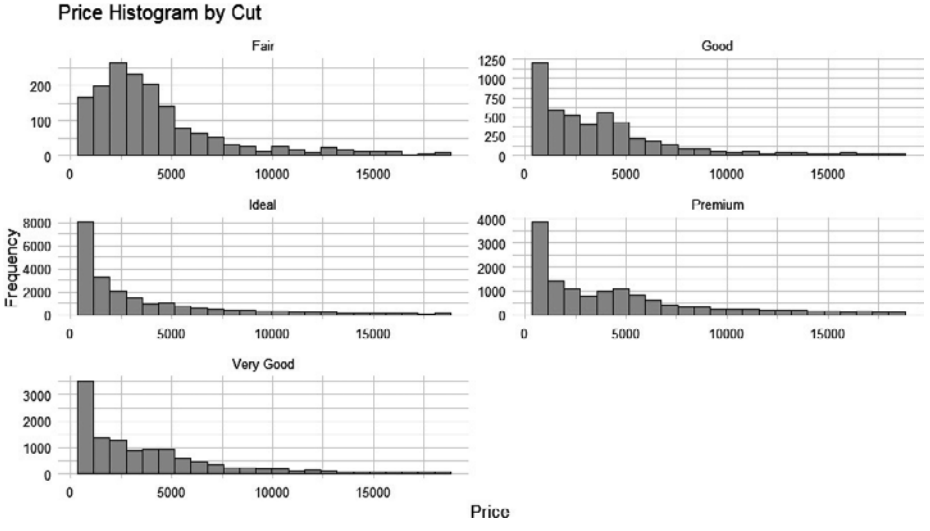


Рис. 12.6. Гистограммы распределения цен, сгруппированных по уровню качества огранки

Теперь, когда мы создали все визуализации, можно переходить к этапу моделирования.

Создание простой модели машинного обучения с помощью R

Создать простую модель машинного обучения с помощью R можно множеством различных способов, перечислить которые здесь было бы крайне затруднительно. Однако вы можете ознакомиться с представлением задач машинного обучения на ресурсе CRAN, перейдя по ссылке https://cran.r-project.org/view=MachineLearning.

В этом разделе мы будем использовать алгоритм XGBoost, реализованный в пакете `healthyR.ai`. Эта версия алгоритма отличается не написанием, а лишь способом сохранения выходных данных. Кроме того, пакет `healthyR.ai` содержит препроцессор для алгоритма XGBoost, позволяющий гарантировать соответствие входных данных тому, что ожидает получить алгоритм перед моделированием. Мы будем использовать две основные функции: `hai_xgboost_data_prepper()` и `hai_auto_xgboost()`.

Мы не станем обсуждать загрузку данных, так как этот процесс уже был подробно рассмотрен ранее, поэтому перейдем сразу к делу.

Предварительная обработка данных

Прежде чем приступить к моделированию, мы должны обработать данные, чтобы привести их в соответствие с требованиями алгоритма. Это довольно легко сделать с помощью функции `hai_xgboost_data_prepper()` из библиотеки `healthyR.ai`. Далее посмотрим, как выглядят данные до и после их обработки. Начнем с просмотра кода, а затем оценим результат его выполнения.

```
# Загрузка библиотек
library(healthyR.ai)
library(dplyr)

glimpse(head(df, 2))
Rows: 2
Columns: 10
$ carat   <dbl> 0.22,   0.86
$ cut     <chr> "Fair", "Fair"
$ color   <chr> "E",    "E"
$ clarity <chr> "VS2",  "SI2"
$ depth   <dbl> 65.1,   55.1
$ table   <dbl> 61,     69
$ price   <dbl> 337,    2757
$ x       <dbl> 3.87,   6.45
$ y       <dbl> 3.78,   6.33
$ z       <dbl> 2.49,   3.52
```

Так выглядят данные до обработки. В выводе мы видим десять столбцов с указанием типа данных каждого из них. Теперь создадим объект `recipe`, передав наши

данные в функцию `hai_xgboost_data_prepper()` и проверив результат. Эта функция принимает два аргумента: `.data` и `.recipe_formula`.

```
# Передача данных в препроцессор
rec_obj <- hai_xgboost_data_prepper(
  .data = df,
  .recipe_formula = price ~ .
)
rec_obj
── Recipe ──────────
── Inputs
Number of variables by role
outcome:   1
predictor: 9
── Operations
• Factor variables from: tidyselect::vars_select_helpers$where(is. character)
• Novel factor level assignment for: recipes::all_nominal_predictors()
• Dummy variables from: recipes::all_nominal_predictors()
• Zero variance filter on: recipes::all_predictors()
```

Теперь взглянем на обработанные данные. Мы видим, что были добавлены дополнительные столбцы и все типы данных преобразованы в `<dbl>` в соответствии с требованиями препроцессора.

```
# Просмотр результата обработки данных
get_juiced_data(rec_obj) |>
  head(2) |>
  glimpse()
Rows: 2
Columns: 24
$ carat         <dbl> 0.22, 0.86
$ depth         <dbl> 65.1, 55.1
$ table         <dbl> 61,   69
$ x             <dbl> 3.87, 6.45
$ y             <dbl> 3.78, 6.33
$ z             <dbl> 2.49, 3.52
$ price         <dbl> 337,  2757
$ cut_Good      <dbl> 0,    0
$ cut_Ideal     <dbl> 0,    0
$ cut_Premium   <dbl> 0,    0
$ cut_Very.Good <dbl> 0,    0
$ color_E       <dbl> 1,    1
$ color_F       <dbl> 0,    0
$ color_G       <dbl> 0,    0
$ color_H       <dbl> 0,    0
$ color_I       <dbl> 0,    0
$ color_J       <dbl> 0,    0
$ clarity_IF    <dbl> 0,    0
$ clarity_SI1   <dbl> 0,    0
$ clarity_SI2   <dbl> 0,    1
$ clarity_VS1   <dbl> 0,    0
```

```
$ clarity_VS2   <dbl> 1,    0
$ clarity_VVS1  <dbl> 0,    0
$ clarity_VVS2  <dbl> 0,    0
```

Теперь, когда мы получили представление о результате обработки данных, воспользуемся функцией `hai_auto_xgboost()` для выполнения моделирования. Ниже приведен полный вызов функции, а документацию по ней можно найти на сайте https://www.spsanderson.com/healthyR.ai/reference/hai_auto_xgboost.html.

```
hai_auto_xgboost(
  .data,
  .rec_obj,
  .splits_obj = NULL,
  .rsamp_obj = NULL,
  .tune = TRUE,
  .grid_size = 10,
  .num_cores = 1,
  .best_metric = "f_meas",
  .model_type = "classification"
)
```

Создадим модель и посмотрим, что получится. Мы, авторы книги, используем параметры `.num_cores = 10`, `.best_metric = "rsq"` и `.model_type = "regression"`, но не советуем вам делать то же самое, если у вас немного свободного времени.

Теперь выполним моделирование с помощью функции `hai_auto_xgboost()`:

```
auto_xgb <- hai_auto_xgboost(
  .data = df,
  .rec_obj = rec_obj,
  .best_metric = "rsq",
  .num_cores = 10,
  .model_type = "regression"
)
```

В результате получается довольно большой объект, который на нашем компьютере занимает 196,1 Мбайт, причем самая большая его часть приходится на `$tuned_info` (`169836312` байтов), что в основном обусловлено графиком `plotly` и тиббломом перекрестной проверки методом Монте-Карло и связано с размером входных данных. Теперь посмотрим на экспортируемые объекты:

```
xgb_wflw_fit <- auto_xgb$model_info$fitted_wflw
class(xgb_wflw_fit)
[1] "workflow"
mod_spec <- xgb_wflw_fit[["fit"]][["actions"]][["model"]][["spec"]]
mod_spec
Boosted Tree Model Specification (regression)

Main Arguments:
  trees = 817
  min_n = 17
```

```
  tree_depth = 9
  learn_rate = 0.0205081386887847
  loss_reduction = 2.0421383990836e-05
  sample_size = 0.762693894910626

Computational engine: xgboost
```

Первым делом мы извлекли подогнанный объект рабочего процесса, который можно использовать для получения прогнозов на основе данных с помощью универсальной функции `predict()`. О том, что это объект `workflow`, нам известно из результата запуска функции `class(xgb_wflw_fit)`.

Важно отметить, что мы, авторы книги, не использовали зерно, а значит, вы можете получить другие результаты. Наша цель заключалась не в полном описании входных и выходных данных, а в том, чтобы показать процесс подгонки модели XGBoost к данным из файла Excel, учитывая относительную сложность выполнения этой задачи моделирования.

Теперь мы можем проделать аналогичные операции над тем же набором данных, на этот раз используя Python.

Создание визуализаций с помощью Python

В этом разделе мы используем библиотеку `plotnine`, являющуюся аналогом R-пакета `ggplot2`, а затем интерпретируем результаты.

Получение данных

Как и в предыдущих главах, загрузим данные с помощью `pandas`. Помните, что в вашем случае путь к XLSX-файлу может отличаться от того, который указан здесь, поэтому скорректируйте значение `file_path` соответствующим образом.

```
import pandas as pd

# Определение пути к файлу (у вас он может быть другим)
file_path = "./Chapter 12/diamonds.xlsx"

# Загрузка набора данных в датафрейм pandas
df = pd.read_excel(file_path)

# Отображение нескольких первых строк датафрейма
print(df.head())
```

Обратите внимание: мы используем необработанный набор данных `diamonds`, не подвергая его разбиению и дальнейшей рекомбинации, как это было сделано в разделе, посвященном созданию визуализаций с помощью R.

Визуализация данных

Когда данные будут загружены, мы сможем использовать библиотеку `plotnine` для создания визуализаций. В этом разделе мы визуализируем различные аспекты набора данных `diamonds`.

Итак, используем загруженный ранее набор данных для создания первого графика.

```
from plotnine import ggplot, aes, geom_bin2d, facet_wrap, theme_minimal, \
  labs, scale_fill_manual

# Определение палитры, оптимизированной для дальтоников
color_palette = ["#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", \
                 "#D55E00", "#CC79A7"]

# Создание графика с помощью plotnine
(
    ggplot(df, aes(x='carat', y='price', fill='cut')) +
    geom_bin2d(bins=20) + # При необходимости скорректируйте параметр bins
    facet_wrap('~ clarity', scales='free') +
    theme_minimal() +
    labs(
        x='Carat',
        y='Price',
        title='Diamonds Data',
        fill='Cut'
    ) +
    scale_fill_manual(values=color_palette)
)
```

Этот код на языке Python воспроизводит код R, написанный в начале главы, с помощью библиотеки `plotnine` для визуализации данных. Функция `ggplot()` инициализирует график, `aes()` задает визуальные параметры, `geom_bin2d()` добавляет геометрию, `facet_wrap()` создает подграфики (фасеты), `theme_minimal()` применяет тему, `labs()` добавляет подписи, а `scale_fill_manual(values=color_palette)` задает цветовую палитру, оптимизированную для дальтоников, используя предопределенные значения `color_palette`.

Результат выполнения этого кода выглядит следующим образом (рис. 12.7).

Эти графики отображают зависимость между весом бриллиантов в каратах и их ценой путем цветового кодирования уровня качества огранки с помощью палитры, оптимизированной для дальтоников.

Теперь создадим диаграмму размаха (для этого мы не будем снова импортировать все функции `plotnine`).

```
from plotnine import geom_boxplot

# Создание графика с помощью plotnine
(
    ggplot(df, aes(x='carat', y='price', fill='cut')) +
```

```
    geom_boxplot(alpha=1/5, outlier_color="lightgrey") +
    facet_wrap('~ clarity', scales='free') +
    theme_minimal() +
    labs(
        x='Carat',
        y='Price',
        title='Diamonds Data',
        fill='Cut'
    ) +
    scale_fill_manual(values=color_palette)
)
```

В данном случае для создания диаграмм размаха используется функция `geom_boxplot()`. Для изменения цвета выбросов в качестве значения параметра `outlier_color` используется `lightgrey`.

Результат показан на рис. 12.8.

Основной целью визуализации по-прежнему является получение информации о данных, с помощью которой можно составить о них более глубокое представление. Что если мы создадим график средней цены? Увидим ли мы то, что нам нужно?

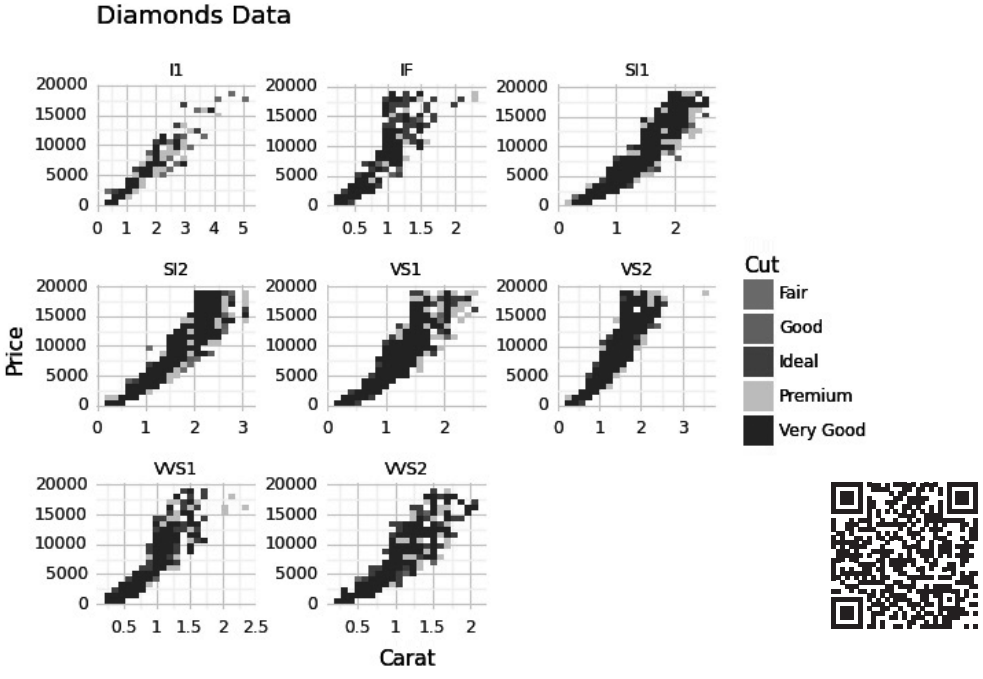


Рис. 12.7. Диаграммы рассеяния, созданные с помощью библиотеки plotnine на основе набора данных diamonds

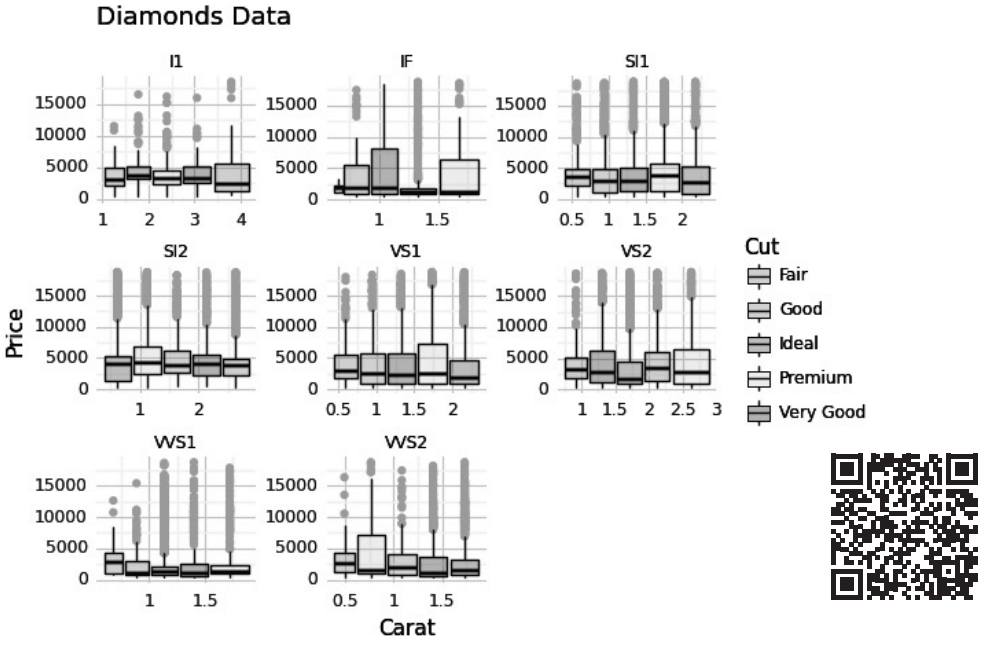


Рис. 12.8. Диаграммы размаха, созданные на основе набора данных diamonds

Мы можем использовать функцию `groupby` из библиотеки `pandas`, чтобы агрегировать цены, вычислить среднее значение для разных групп данных и создать графики, визуализирующие зависимость средней цены бриллианта от его чистоты и качества огранки.

```
from plotnine import geom_point, geom_line, geom_smooth, scale_color_manual

# Создание графика средней цены
(
    ggplot(df.groupby(['clarity', 'cut']).mean().reset_index(), \
      aes(x='clarity', y='price', group='cut', color='cut')) +
    geom_point() +
    geom_line() +
    geom_smooth() +
    facet_wrap('~ cut', ncol=2) +
    labs(
        x='Clarity',
        y='Mean Price',
        title='Mean Price by Clarity and Cut',
        color='Cut'
    ) +
    theme_minimal() +
    scale_color_manual(values=color_palette)
)
```

Получившиеся графики выглядят следующим образом (рис. 12.9).

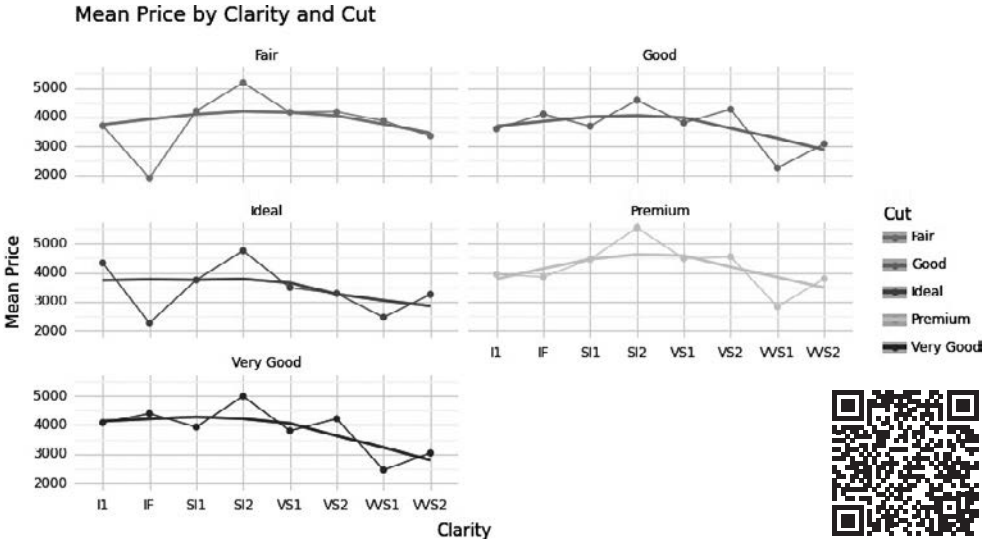


Рис. 12.9. Графики зависимости средней цены бриллианта от его чистоты и качества огранки

В каждой категории бриллиантов, разделенных по качеству огранки (Cut), кривая ведет себя похожим образом: средняя цена сначала растет в зависимости от степени чистоты (Clarity), а затем падает. Эти рост и падение наименее заметны для бриллиантов категории *Ideal* и наиболее заметны для бриллиантов категорий *Premium* и *Very Good*.

Можем ли мы получить дополнительную информацию, если создадим график средней цены, сгруппировав данные чуть иначе? Визуализируем динамику средней цены за карат.

```
# Вычисление средней цены за карат в зависимости от чистоты,
# цвета и качества огранки
df_mean = df.groupby(['cut', 'color', 'clarity']).apply(lambda x: \
    (x['price'] x['carat']).mean()).reset_index(name='m')

# Создание графика с помощью plotnine
(
    ggplot(df_mean, aes(x='color', y='m', group='clarity', color='clarity')) +
    geom_point() +
    geom_line() +
    facet_wrap('~ cut', ncol=2, scales='free') +
```

```
    labs(
        x='Clarity',
        y='Mean Price',
        title='Mean Price per Carat by Clarity,
        Color and Cut',
        color='Cut'
    ) +
    scale_color_manual(values=color_palette)
)
```

Полученное изображение действительно демонстрирует кое-что интересное (рис. 12.10).

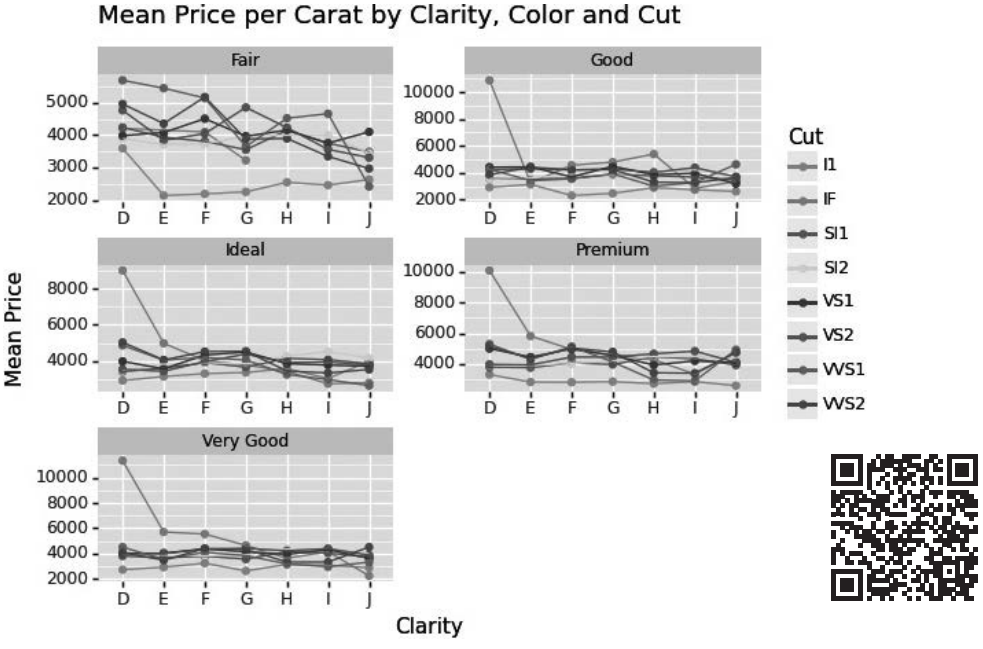


Рис. 12.10. Графики зависимости средней цены за карат от чистоты, цвета и качества огранки

Во всех категориях камней, кроме *Fair*, мы видим экстремально высокую цену на внутренне безупречные (IF) бриллианты цвета D, тогда как для остальных цены остаются примерно одинаковыми. Однако в категории *Fair* цены демонстрируют явную тенденцию к снижению, а существенная разница в цене наблюдается лишь между бриллиантами цвета D с видимыми глазу включениями (I1) и камнями остальных цветов той же чистоты.

Наконец, прежде чем перейти к моделированию, посмотрим на гистограммы распределения цен, сгруппированных по уровню качества огранки.

```
from plotnine import geom_histogram

# Создание гистограмм распределения цен, сгруппированных
# по уровню качества огранки
(
    ggplot(df, aes(x='price')) +
    geom_histogram(fill='lightblue', color='black') +
    theme_minimal() +
    facet_wrap('~ cut', ncol=2, scales='free') +
    labs(x='Price', y='Frequency', title='Price Histogram by Cut')
)
```

Результат показан на рис. 12.11.

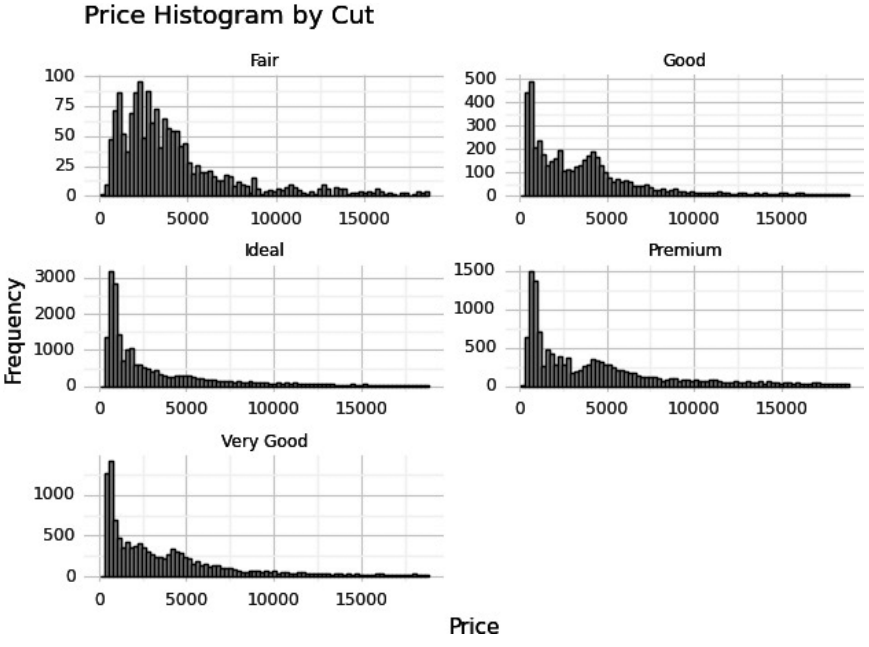


Рис. 12.11. Гистограммы распределения цен, сгруппированных по уровню качества огранки

Здесь мы используем стандартный метод разбиения данных на интервалы, поскольку, к сожалению, пакет `healthyR`, который мы использовали в R-версии, (пока) недоступен для Python.

Мы видим, что распределения цен имеют очень длинные хвосты (то есть чрезвычайно высокие цены являются относительно типичными, хоть и встречаются неча-

сто). Вдобавок мы можем заметить второй максимум в категориях *Good* и *Premium* (и менее выраженный максимум в категории *Very Good*).

Теперь, когда мы гораздо лучше понимаем данные благодаря их визуализации, можем приступить к моделированию!

Создание простой модели машинного обучения с помощью Python

В этом разделе мы создадим простую модель машинного обучения с помощью Python. Этот язык уже давно превратился в основной инструмент решения задач машинного обучения (его очевидной альтернативой является R), и количество пакетов, реализующих соответствующие алгоритмы, постоянно растет. Тем не менее самым распространенным из них остается `sklearn`, поэтому в данном разделе мы будем использовать именно его. Как и в части главы, посвященной моделированию с помощью R, мы будем использовать модель `xgboost`, поскольку она отличается превосходным балансом между производительностью и объяснимостью.

Мы используем данные, загруженные в предыдущем разделе.

Предварительная обработка данных

Первое, что необходимо сделать на этапе моделирования, — подготовить данные. К счастью, библиотека `sklearn` предусматривает встроенную функцию для их предварительной обработки.

Рассмотрим этапы данного процесса.

- **Обработка отсутствующих значений.** Перед обучением модели необходимо решить проблему, связанную с отсутствующими значениями в наборе данных. Для этого библиотека `sklearn` предусматривает методы импутации (восстановления) таких значений или удаления содержащих их строк/столбцов.
- **Масштабирование признаков.** Многие алгоритмы машинного обучения работают лучше, когда признаки имеют один масштаб. Библиотека `sklearn` предлагает утилиты для масштабирования признаков, в том числе для их стандартизации (масштабирование признаков до нулевого среднего значения и единичной дисперсии) и нормализации (масштабирование признаков до определенного диапазона).
- **Кодирование категориальных переменных.** Алгоритмы машинного обучения обычно принимают на вход числовые данные, поэтому категориальные переменные необходимо закодировать в числовые представления. Библиотека `sklearn` предусматривает методы унитарного кодирования категориальных переменных, а также методы их кодирования с помощью порядковых меток.

- **Разбиение данных.** Перед обучением модели необходимо разбить данные на обучающий и тестовый наборы, что позволит выполнить оценку ее эффективности. Библиотека `sklearn` предлагает функции, с помощью которых можно разделить набор данных на обучающее и тестовое множества в заданных пропорциях.
- **Конструирование признаков.** Библиотека `sklearn` поддерживает различные методы конструирования признаков, такие как генерация полиномиальных признаков, создание членов взаимодействия и снижение размерности с помощью *метода главных компонент* (principal component analysis, PCA).

ПРИМЕЧАНИЕ

Важно, чтобы процесс конструирования признаков не приводил к загрязнению обучающих данных информацией из тестового набора. Этого не должно происходить и при очистке данных (например, при восстановлении отсутствующих значений).

Мы подробно обсуждали процесс очистки данных в главе 8, поэтому воспользуемся тем фактом, что набор данных `diamonds` уже был очищен, и сразу перейдем к масштабированию признаков и кодированию категориальных переменных.

```
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.model_selection import train_test_split
import numpy as np

# Кодирование категориальных переменных
encoder = OneHotEncoder()
df_encoded = encoder.fit_transform(df[['cut', 'color', 'clarity']])

# Масштабирование числовых признаков
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df[['carat', 'depth', 'table', 'x', 'y', 'z']])

# Конкатенация закодированных категориальных признаков
# с масштабированными числовыми признаками
df_processed = np.concatenate((df_encoded.toarray(), df_scaled), axis=1)

# Разбиение данных на обучающий и тестовый наборы
X_train, X_test, y_train, y_test = train_test_split(
    df_processed, df["price"], test_size=0.2, random_state=42)
```

В этом фрагменте кода показан процесс унитарного кодирования категориальных переменных (качество огранки (`cut`), цвет (`color`) и чистота (`clarity`)) и масштабирования числовых признаков с помощью функции `StandardScaler` из библиотеки `sklearn`. Затем закодированные категориальные признаки объединяются

с масштабированными числовыми признаками, и набор данных разбивается на обучающее и тестовое множества с помощью функции `train_test_split()`.

Теперь сравним данные до и после предварительной обработки.

Исходный набор данных выглядит следующим образом (рис. 12.12).

```
>>> print(df)
       carat        cut color clarity  depth  table  price     x     y     z
0       0.23      Ideal     E     SI2   61.5   55.0    326  3.95  3.98  2.43
1       0.21    Premium     E     SI1   59.8   61.0    326  3.89  3.84  2.31
2       0.23       Good     E     VS1   56.9   65.0    327  4.05  4.07  2.31
3       0.29    Premium     I     VS2   62.4   58.0    334  4.20  4.23  2.63
4       0.31       Good     J     SI2   63.3   58.0    335  4.34  4.35  2.75
...      ...        ...   ...     ...    ...    ...    ...   ...   ...   ...
53935   0.72      Ideal     D     SI1   60.8   57.0   2757  5.75  5.76  3.50
53936   0.72       Good     D     SI1   63.1   55.0   2757  5.69  5.75  3.61
53937   0.70  Very Good     D     SI1   62.8   60.0   2757  5.66  5.68  3.56
53938   0.86    Premium     H     SI2   61.0   58.0   2757  6.15  6.12  3.74
53939   0.75      Ideal     D     SI2   62.2   55.0   2757  5.83  5.87  3.64

[53940 rows x 10 columns]
>>>
```

Рис. 12.12. Необработанные данные, считанные из Excel

Как видите, набор данных содержит комбинацию числовых и категориальных переменных (последние будут закодированы с помощью унитарного кодирования).

После обработки набор данных выглядит следующим образом (рис. 12.13).

```
>>> pd.DataFrame(X_train)
       0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16   17   18   19        20        21        22        23        24        25
0     0.0  1.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  2.557052 -2.547385  2.928129  2.227628  2.149919  1.744764
1     0.0  0.0  0.0  0.0  1.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.447378 -1.221133  1.137995  0.747798  0.661464  0.548273
2     0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.637249  0.523936  0.242928  0.765627  0.705242  0.795341
3     0.0  1.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  1.481118 -0.174092  3.375663  1.318335  1.256846  1.248797
4     0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  1.523312  0.244725 -0.204605  1.371823  1.388180  1.404672
...   ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...       ...       ...       ...       ...       ...
43147 0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.531765  0.454133  0.690462  0.667566  0.678975  0.724489
43148 0.0  0.0  1.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0 -0.691846 -0.523105 -1.099672 -0.625056 -0.634367 -0.678389
43149 0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0 -0.987200 -1.011725  0.242928 -1.106447 -1.115926 -1.188526
43150 0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.215314  0.733344  0.690462  0.355554  0.258706  0.398568
43151 0.0  0.0  0.0  1.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.721636 -0.941922  0.242928  0.970664  0.924133  0.809512

[43152 rows x 26 columns]
```

Рис. 12.13. Предварительно обработанные обучающие данные

Набор предварительно обработанных обучающих данных содержит меньше строк (остальные составляют тестовый набор), но больше столбцов. Столбец `price` (цена) отсутствует (так как это переменная, которую мы хотим предсказать), а категориальные переменные заменены значениями `0` и `1` в результате унитарного кодирования. Для каждого уникального значения каждой категориальной переменной был добавлен новый столбец, который содержит `1`, если в исходном наборе данных есть это значение, и `0`, если в нем имеется что-то другое.

Переменная `y_train` содержит значение столбца `price` для каждой строки набора обучающих данных.

Выполнив предварительную обработку данных, можем приступить к моделированию:

```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error

# Создание экземпляра регрессора XGBoost
xgb_reg = GradientBoostingRegressor(random_state=42)

# Обучение модели
xgb_reg.fit(X_train, y_train)

# Прогнозирование на тестовом наборе
y_pred = xgb_reg.predict(X_test)

# Вычисление RMSE
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error:", rmse)
```

В этом коде мы выполняем следующие действия:

- импортируем `GradientBoostingRegressor` из `sklearn.ensemble`;
- создаем экземпляр регрессора на основе алгоритма градиентного бустинга (`xgb_reg`) из реализации `scikit-learn`;
- обучаем модель с помощью метода `fit` на наборе обучающих данных (`X_train` и `y_train`);
- делаем предсказания на тестовом наборе с помощью метода `predict` и вычисляем *квадратный корень из среднеквадратичной ошибки* (`RMSE`), показывающей разницу между прогнозными (`y_pred`) и фактическими целевыми значениями (`y_test`).

RMSE — это показатель, который широко используется в регрессионном анализе и измеряет среднюю величину ошибок или разностей между прогнозируемыми и наблюдаемыми значениями. Он представляет собой числовое значение, позволяющее оценить степень соответствия регрессионной модели данным. RMSE измеряется в тех же единицах, что и целевая переменная (в данном случае `price`).

Чем ниже значение RMSE, тем ближе прогнозы модели к фактическим значениям и тем выше ее производительность. Другими словами, более низкое значение RMSE говорит о меньшем отклонении прогнозов модели от истинных значений, что свидетельствует о том, что ее точность и прогнозные возможности более высоки.

Полезность RMSE заключается в том, что данный показатель учитывает величину ошибок и штрафует за большие ошибки сильнее, чем за маленькие. Поэтому

минимизация значения RMSE позволяет получить модель, дающую более точные прогнозы.

В целом RMSE — ценный инструмент сравнения различных регрессионных моделей и оценки их прогнозных возможностей в реальных приложениях.

Значение RMSE нашей модели составляет около `720`, что значительно ниже средней цены (`3933`) и стандартного отклонения переменной `price` (`3989`). Это хорошая новость, говорящая о том, что степень соответствия модели данным достаточно высока.

Разумеется, вы можете использовать другие модели машинного обучения (случайные леса, `lightgbm` или `catgbm`, или даже *модели глубокого обучения*) и другие показатели для оценки соответствия (R^2, MAE и т. д.). Их описание выходит за рамки нашей книги, а в этом разделе мы всего лишь хотели познакомить вас с общим рабочим процессом.

Резюме

В последней главе вы изучили методы анализа и визуализации полученных из Excel данных с помощью R и Python. Мы начали с загрузки набора данных `diamonds` и его визуализации с использованием библиотек `ggplot2` и `plotnine`. Создав различные графики, в частности диаграммы размаха, графики средней цены и гистограммы, вы получили представление о взаимосвязях между различными переменными в наборе данных.

При создании модели машинного обучения мы использовали библиотеки `healthyR` и `scikit-learn` для предварительной обработки данных, в том числе для кодирования категориальных переменных и разбиения набора данных на обучающее и тестовое множества. Затем мы реализовали регрессионную модель с помощью алгоритма XGBoost и оценили ее производительность, используя показатель RMSE.

В завершение мы хотим поблагодарить вас за то, что вместе с нами вы изучали мир анализа и визуализации данных Excel с помощью R и Python. Надеемся, что материал книги показался вам увлекательным, а примеры — интересными. Рекомендуем продолжать изучение указанных тем и применять знания, полученные в ходе чтения нашей книги. Это позволит вам открывать новые возможности в сфере анализа и визуализации данных. Желаем удачи!

Эту книгу мы выпустили вместе с Orion soft

Orion soft создает ПО для ИТ-инфраструктуры Enterprise-бизнеса. Главные принципы разработки, которым отвечает каждый продукт компании, — стабильность, простота использования и безопасность.

В продуктовую экосистему Orion soft входит zVirt — платформа виртуализации серверов, дисков и сетей с одной из крупнейших баз инсталляций в России.

Orion soft стремится постоянно развивать как свои компетенции, так и российское ИТ-сообщество, собирая и распространяя лучшие современные практики разработки.