

Аналитика в Power BI с помощью R и Python



Загрузка,
преобразование и
визуализация данных



Райан Уэйд

ДМК
ИЗДАТЕЛЬСТВО

Райан Уэйд

Аналитика в Power BI с помощью R и Python



Advanced Analytics in Power BI with R and Python



Ingesting, Transforming, Visualizing

Ryan Wade



Apress®



Аналитика в Power BI с помощью R и Python

**Загрузка, преобразование
и визуализация данных**

Райан Уэйд



Москва, 2021

УДК 004.424
ББК 32.372
У97



Уэйд Р.

У97 Аналитика в Power BI с помощью R и Python / пер. с англ. А. Ю. Гинько. – М.: ДМК Пресс, 2021. – 338 с.: ил.

ISBN 978-5-97060-923-1

В данной книге подробно рассказывается, как использовать на практике языки программирования R и Python для визуализации данных, загрузки в модель, преобразования и выполнения других задач с помощью аналитического инструмента Power BI. Вы узнаете, как создавать пользовательские элементы визуализации, реализовывать методы машинного обучения и искусственного интеллекта, применять продвинутые методы обработки текстовой информации с использованием техник, недоступных в Power Query и DAX, обеспечивать взаимодействие со службами Microsoft Cognitive Services без необходимости приобретать дорогостоящую подписку на Power BI Premium. В заключение рассказывается, как можно воспользоваться языками программирования R и Python в корпоративных решениях, внедренных в Power BI.

Для выполнения практических упражнений понадобится облачная платформа Microsoft Azure. Также для работы с примерами из данной книги рекомендуется настроить виртуальную машину для анализа данных (Data Science Virtual Machine – DSVM).

Издание адресовано читателям, которые работают с большими объемами данных и хотят эффективно применять инструменты бизнес-аналитики

УДК 004.424
ББК 32.372



First published in English under the title Advanced Analytics in Power BI with R and Python; Ingesting, Transforming, Visualizing by Ryan Wade, edition: 1. This edition has been translated and published under licence from APress Media, LLC, part of Springer Nature. APress Media, LLC, part of Springer Nature takes no responsibility and shall not be made liable for the accuracy of the translation. Russian language edition copyright © 2021 by DMK Press. All rights reserved.

Все права защищены. Любая часть этой книги не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами без письменного разрешения владельцев авторских прав.

ISBN 978-1-4842-5828-6 (англ.)
ISBN 978-5-97060-923-1 (рус.)

© Ryan Wade, 2020
© Перевод, оформление, издание,
ДМК Пресс, 2021

Я хотел бы поблагодарить моих коллег из BlueGranite. Это большая честь – быть частью команды умных и талантливых людей. Железо точит железо. Также я благодарен окружению, в котором вырос – на западе Мичигана в штате Индиана. Опыт, полученный там, для меня поистине бесценен. Хочу сказать спасибо Патрику Леблану (Patrick Leblanc), Мико Юку (Mico Yuk), Терри Моррису (Terry Morris), доктору Брандейсу Маршаллу (Dr. Brandeis Marshall) и доктору Сайдику Вотсон (Dr. Sydeaka Watson). Вы обладаете качествами, которые я очень ценю, и я постоянно учился на ваших примерах.

Кроме того, хотелось бы выразить благодарность всем моим напарникам по командам, начиная с низших лиг и заканчивая колледжем. Спорт был одним из лучших моих учителей в жизни, и мне выпала большая честь пройти через все это вместе с вами. Я с огромным уважением и почтением отношусь ко многим из вас. Спасибо моим бывшим тренерам, особенно в старшей школе и колледже. Вызовы, которые вы передо мной ставили, закалили мой характер и позволили стать лучше во многих аспектах жизни. И за это я вам очень признателен.

Также я безмерно благодарен моей семье. Вырастить ребенка очень непросто, и я бесконечно ценю помощь, которую мне оказывали мои родственники. Спасибо вам, Тим Адамс младший и маленькая Камилла. Вы обладаете редким сочетанием высокого интеллекта и простоты в общении. Совсем скоро мы поменяемся ролями: вы будете учить, а я – учиться.

Хочу отметить признанием моих братьев в Мичигане и братьев, с которыми познакомился, играя в футбол в Луисвилле. Мы вместе прошли через многое. Что бы ни случилось, мы останемся братьями навсегда. Хотел бы сказать спасибо и моим двоюродным братьям и сестрам. Мы выросли вместе, у нас много общих воспоминаний. Многие из вас поддерживали меня в не самые лучшие времена. Я этого никогда не забуду. Спасибо моим родным братьям и сестрам: Полетте (мир ее праху), Стефани, Тине и Люку, а также моему племяннику Джунбагу, который был мне как брат. Когда мы нуждались друг в друге, мы всегда оказывались рядом. Пусть так будет и дальше. Наконец, я благодарен моим родителям: Лютеру и Эрнестине Уэйд. Я очень ценю вашу любовь, поддержку и жертвенность во имя своих детей. Как сказал бы папа: «Я люблю вас, и это навсегда!»

Последним по порядку, но не по значению, я хотел бы сказать спасибо Всевышнему. Я благодарен за все способности и таланты, которыми Ты меня наделил. Обещаю применить их по назначению, чтобы принести максимальную пользу обществу.

Содержание

От издательства.....	20
Об авторе.....	21
О техническом редакторе.....	22
Благодарности	23
Введение.....	24

Часть I. СОЗДАНИЕ ПОЛЬЗОВАТЕЛЬСКОЙ ВИЗУАЛИЗАЦИИ ПРИ ПОМОЩИ R	41
---	----

Глава 1. Грамматика графиков	42
---	----

Пошаговое создание визуализации в Power BI при помощи R.....	43
Шаг 1. Настройте Power BI	43
Шаг 2. Перенесите визуальный элемент R в рабочую область Power BI	43
Шаг 3. Определитесь с набором данных.....	43
Шаг 4. Спроектируйте визуальный элемент в среде разработки R.....	44
Шаг 5. Используйте следующий шаблон для разработки элемента на R	45
Шаг 6. Добавьте скрипту функциональности	45
Рекомендованные шаги по созданию визуального элемента на R при помощи ggplot2.....	46
Шаг 1. Импортируйте нужные для скрипта пакеты	46
Шаг 2. Выполните необходимое преобразование исходных данных	47
Шаг 3. Создайте визуализацию при помощи функции ggplot()	48
Шаг 4. Добавьте нужные геометрии.....	48
Шаг 5. Определите заголовки, подзаголовки и подписи	50
Шаг 6. Приведите в порядок оси.....	50
Шаг 7. Примените тему при необходимости	53
Шаг 8. Используйте функцию theme() для настройки оформления	54
Дополнительный шаг: задайте цвета точек на диаграмме рассеяния	55
Важность оперирования «чистыми» данными.....	58
Популярные геометрии.....	58
Управление эстетиками через шкалы	64
Встроенные в пакет ggplot2 темы.....	65
Использование визуальных элементов R в службе Power BI.....	66
Вспомогательные пакеты ggplot2.....	66
Заключение	67

Глава 2. Создание пользовательских визуализаций на R в Power BI при помощи ggplot2	68
Диаграмма с аннотацией	69
Шаг 1. Загрузите исходные данные.....	70
Шаг 2. Создайте срез на основании года на панели фильтров.....	71
Шаг 3. Настройте визуальный элемент R в Power BI.....	71
Шаг 4. Экспортируйте данные в R Studio для дальнейшей разработки.....	72
Шаг 5. Загрузите необходимые пакеты.....	73
Шаг 6. Создайте переменные для проверки данных.....	73
Шаг 7. Выполните проверку данных.....	74
Шаг 8. Добавьте столбцы к набору данных, необходимые для нашей визуализации.....	75
Шаг 9. Создайте переменные для динамических составляющих диаграммы.....	76
Шаг 10. Постройте диаграмму при помощи функции ggplot().....	78
Шаг 11. Добавьте слой со столбчатой диаграммой на визуальный элемент.....	78
Шаг 12. Добавьте текстовый слой на визуальный элемент.....	79
Шаг 13. Измените ось y.....	80
Шаг 14. Преобразуйте вертикальную столбчатую диаграмму в горизонтальную.....	81
Шаг 15. Добавьте на диаграмму динамическую аннотацию.....	81
Шаг 16. Добавьте динамические заголовки и подпись на визуальный элемент.....	83
Шаг 17. Удалите метки с осей x и y.....	84
Шаг 18. Удалите легенду с диаграммы.....	84
Шаг 19. Измените внешний вид диаграммы при помощи темы theme_few().....	85
Шаг 20. Расположите заголовки по центру.....	86
Шаг 21. Перенесите код в Power BI.....	87
Пузырьковая диаграмма	89
Шаг 1. Загрузите исходные данные.....	90
Шаг 2. Загрузите данные в Power BI.....	90
Шаг 3. Создайте срез на основании года.....	90
Шаг 4. Выполните базовую настройку визуального элемента R.....	91
Шаг 5. Экспортируйте данные в R Studio для разработки элемента.....	91
Шаг 6. Загрузите требуемые пакеты.....	91
Шаг 7. Создайте переменные для проверки данных.....	91
Шаг 8. Создайте код для проверки.....	92
Шаг 9. Определите цвета для конференций и дивизионов.....	92
Шаг 10. Динамически определите заголовок диаграммы.....	93
Шаг 11. Создайте набор данных для диаграммы.....	93
Шаг 12. Создайте диаграмму при помощи функции ggplot.....	94
Шаг 13. Добавьте слой для пузырьковой диаграммы при помощи геометрии geom_point.....	94
Шаг 14. Добавьте метки на диаграмму.....	96

Шаг 15. Измените цвет границ и заливок пузырьков на диаграмме	97
Шаг 16. Создайте заголовок диаграммы	98
Шаг 17. Задайте тему.....	98
Шаг 18. Перенесите код в Power BI	98
Визуализация прогнозирования	99
Шаг 1. Загрузите исходные данные	101
Шаг 2. Создайте срез по квотербекам на панели фильтров	102
Шаг 3. Настройте визуальный элемент R в Power BI.....	102
Шаг 4. Экспортируйте данные в R Studio для дальнейшей разработки.....	102
Шаг 5. Загрузите необходимые пакеты	102
Шаг 6. Создайте переменные для проверки данных	103
Шаг 7. Выполните проверку данных	103
Шаг 8. Создайте динамический заголовок для визуализации	104
Шаг 9. Создайте набор данных, необходимый для составления прогноза	104
Шаг 10. Постройте прогноз.....	105
Шаг 11. Постройте диаграмму.....	105
Шаг 12. Перенесите код в Power BI	106
Линейная диаграмма с затенением	107
Шаг 1. Загрузите исходные данные	109
Шаг 2. Загрузите данные в Power BI	110
Шаг 3. Создайте срезы в отчете	111
Шаг 4. Настройте визуальный элемент R в Power BI.....	111
Шаг 5. Экспортируйте данные в R Studio для дальнейшей разработки.....	112
Шаг 6. Загрузите необходимые пакеты	112
Шаг 7. Создайте переменные для проверки данных	112
Шаг 8. Выполните проверку данных	113
Шаг 9. Создайте новый датафрейм на основании датафрейма dataset.....	113
Шаг 10. Создайте переменные для динамических составляющих диаграммы	114
Шаг 11. Создайте наборы данных, необходимые для наложения тени	114
Шаг 12. Создайте наборы данных, необходимые для отрисовки графика.....	118
Шаг 13. Создайте символьный вектор для хранения цветовой схемы затенения	118
Шаг 14. Постройте диаграмму при помощи функции ggplot()	119
Шаг 15. Добавьте слой для создания затенения	119
Шаг 16. Добавьте линейную диаграмму на основании выбора пользователя	120
Шаг 17. Раскрасьте фоновую заливку в соответствии с предопределенной цветовой схемой партий	121
Шаг 18. Отформатируйте ось <i>y</i> в соответствии с выбором пользователя	121
Шаг 19. Добавьте метки на оси <i>x</i> и <i>y</i>	121
Шаг 20. Снабдите диаграмму динамическим заголовком и подзаголовком	122
Шаг 21. Измените внешний вид диаграммы в стиле журнала The Economist	122

Шаг 22. Перенесите код в Power BI	122
Карта	123
Шаг 1. Загрузите исходные данные	125
Шаг 2. Загрузите данные в Power BI	126
Шаг 3. Создайте срез в отчете на основании выбранного в фильтре штата	127
Шаг 4. Настройте визуальный элемент R в Power BI	127
Шаг 5. Экпортируйте данные в R Studio для дальнейшей разработки	127
Шаг 6. Загрузите необходимые пакеты	127
Шаг 7. Создайте переменные для проверки данных	127
Шаг 8. Выполните проверку данных	128
Шаг 9. Создайте переменные для заголовков диаграммы	128
Шаг 10. Добавьте к набору данных столбец с квинтилем	129
Шаг 11. Создайте символьный вектор для хранения цветовой схемы затемнения	129
Шаг 12. Постройте диаграмму при помощи функции ggplot()	130
Шаг 13. Добавьте слой с картой	130
Шаг 14. Отформатируйте оси <i>x</i> и <i>y</i>	131
Шаг 15. Раскрасьте округа на основании квинтилей	131
Шаг 16. Улучшите отображение карты выбранного штата	133
Шаг 17. Снабдите диаграмму динамическим заголовком и подзаголовком	133
Шаг 18. Примените тему theme_map()	133
Шаг 19. Перенесите код в Power BI	133
Диаграмма квадрантов	134
Шаг 1. Загрузите исходные данные	137
Шаг 2. Загрузите данные в Power BI	138
Шаг 3. Создайте срезы в отчете по типу игры и четверти матча	138
Шаг 4. Настройте визуальный элемент R в Power BI	138
Шаг 5. Экпортируйте данные в R Studio для дальнейшей разработки	138
Шаг 6. Загрузите необходимые пакеты	139
Шаг 7. Создайте переменные для проверки данных	139
Шаг 8. Выполните проверку данных	140
Шаг 9. Создайте заголовки диаграммы	140
Шаг 10. Добавьте дополнительные столбцы в набор данных	140
Шаг 11. Постройте диаграмму при помощи функции ggplot()	141
Шаг 12. Используйте геометрию geom_point() для создания диаграммы рассеяния	141
Шаг 13. Добавьте метки игроков для всех квадрантов	141
Шаг 14. Добавьте горизонтальные и вертикальные линии, проходящие через центр	142
Шаг 15. Добавьте на диаграмму заголовки квадрантов	143
Шаг 16. Добавьте метки на оси <i>x</i> и <i>y</i>	145
Шаг 17. Снабдите диаграмму динамическими заголовками и подписями	145
Шаг 18. Примените тему theme_tufte	145
Шаг 19. Выполните финальную очистку	145

Шаг 20. Перенесите код в Power BI	148
Добавление линии регрессии.....	149
Шаг 1. Загрузите исходные данные.....	150
Шаг 2. Загрузите данные в Power BI.....	151
Шаг 3. Настройте визуальный элемент R в Power BI.....	151
Шаг 4. Экспортируйте данные в R Studio для дальнейшей разработки.....	151
Шаг 5. Загрузите необходимые пакеты	151
Шаг 6. Создайте переменные для проверки данных.....	151
Шаг 7. Выполните проверку данных	152
Шаг 8. Постройте диаграмму при помощи функции ggplot()	152
Шаг 9. Используйте геометрию geom_point() для создания диаграммы рассеяния	153
Шаг 10. Добавьте на визуализацию слой с линией регрессии	153
Шаг 11. Снабдите диаграмму заголовком и подзаголовком	154
Шаг 12. Примените тему	155
Шаг 13. Выполните финальную очистку.....	155
Шаг 14. Перенесите код в Power BI	155

Часть II. ЗАГРУЗКА ИНФОРМАЦИИ В МОДЕЛЬ ДАННЫХ POWER BI ПРИ ПОМОЩИ R И PYTHON

Глава 3. Чтение файлов CSV.....

Динамическое объединение файлов	158
Пример сценария.....	159
Выбор файлов за скользящий период из 24 месяцев при помощи R	159
Шаг 1. Импортируйте необходимые пакеты для скрипта	159
Шаг 2. Установите рабочую директорию на папку, содержащую наборы данных о продажах	160
Шаг 3. Считайте имена файлов в символьный вектор.....	161
Шаг 4. Создайте вектор дат	162
Шаг 5. Создайте датафрейм, состоящий из двух векторов.....	163
Шаг 6. Получите верхнюю и нижнюю границы желаемого диапазона дат.....	164
Шаг 7. Ограничьте датафрейм только нужными нам месяцами	164
Шаг 8. Создайте датафрейм на основании объединенных файлов	165
Шаг 9. Соберите написанный код и перенесите в редактор скриптов в Power BI.....	166
Выбор файлов за скользящий период из 24 месяцев при помощи Python	168
Шаг 1. Создайте скрипт на Python и загрузите необходимые библиотеки	168
Шаг 2. Установите рабочую директорию на папку Python_Code	168
Шаг 3. Загрузите перечень имен файлов в список	169
Шаг 4. Создайте датафрейм pandas с информацией о файлах для объединения.....	169

Шаг 5. Создайте новый столбец с датой в датафрейме.....	170
Шаг 6. Определите границы нужного нам диапазона дат.....	170
Шаг 7. Ограничьте датафрейм нужным диапазоном	170
Шаг 8. Объедините файлы в единый датафрейм.....	171
Шаг 9. Перенесите скрипт в Power BI.....	172
Фильтрация строк на основе регулярных выражений.....	174
Использование регулярных выражений в R.....	174
Шаг 1. Загрузите необходимые для работы пакеты	175
Шаг 2. Загрузите в R файл с потенциальными избирателями.....	175
Шаг 3. Определите регулярное выражение	175
Шаг 4. Исключите неправильные адреса из набора данных.....	176
Шаг 5. Объедините написанный код в один скрипт и перенесите в редактор скриптов в Power BI.....	176
Использование регулярных выражений в Python	176
Шаг 1. Загрузите необходимые для работы библиотеки	177
Шаг 2. Загрузите в Python файл с избирателями и присвойте его содержимое датафрейму	177
Шаг 3. Определите регулярное выражение	177
Шаг 4. Исключите неправильные адреса из набора данных.....	177
Шаг 5. Объедините написанный код в один скрипт и перенесите в редактор скриптов Python в Power BI.....	177

Глава 4. Чтение данных из Microsoft Excel..... 179

Чтение файлов Excel при помощи R	180
Шаг 1. Импортируйте пакеты tidyverse и readxl.....	180
Шаг 2. Создайте оболочку функции combine_sheets.....	181
Шаг 3. Получите имена листов для объединения из указанной рабочей книги	181
Шаг 4. Преобразуйте символьный вектор, полученный на предыдущем шаге, в именованный символьный вектор.....	181
Шаг 5. Используйте функцию map_dfr() для объединения информации с листов в один датафрейм	182
Шаг 6. Верните датафрейм из функции	183
Шаг 7. Направьте рабочую директорию на папку с файлами Excel.....	183
Шаг 8. Сохраните в переменной excel_file_paths список файлов для обработки.....	184
Шаг 9. Используйте функцию map_dfr() для применения функции combine_sheets() ко всем выбранным файлам	184
Шаг 10. Скопируйте скрипт и вставьте в редактор скриптов R в Power BI через инструмент Получить данные (GetData)	184
Чтение файлов Excel при помощи Python.....	185
Шаг 1. Импортируйте библиотеки os и pandas	186
Шаг 2. Создайте оболочку функции combine_sheets()	186
Шаг 3. Создайте объект Excel на основании пути, переданного в функцию в аргументе excel_file_path	186
Шаг 4. Создайте список имен листов в рабочей книге.....	186

Шаг 5. Используйте метод <code>read_excel()</code> из библиотеки <code>pandas</code> для считывания данных в один датафрейм.....	187
Шаг 6. Верните датафрейм <code>df</code> из функции <code>combine_sheets</code>	187
Шаг 7. Установите рабочую директорию в папку, в которой находятся файлы Excel.....	187
Шаг 8. Получите список файлов в текущей рабочей директории и присвойте его переменной <code>excel_file_paths</code>	188
Шаг 9. Создайте пустой датафрейм и назовите его <code>combined_workbooks</code> ...	188
Шаг 10. Создайте заготовку для цикла <code>for</code>	188
Шаг 11. Объедините данные со всех листов в один датафрейм при помощи функции <code>combine_sheets()</code>	189
Шаг 12. Добавьте датафрейм <code>combined_workbook</code> к главному датафрейму <code>combined_workbooks</code>	189
Шаг 13. Скопируйте скрипт и вставьте в редактор скриптов Python в Power BI через инструмент Получить данные (GetData)	190
Глава 5. Чтение данных из SQL Server	191
Добавление базы данных <code>AdventureWorksDW_StarSchema</code> к вашему экземпляру SQL Server.....	191
Чтение данных из SQL Server в Power BI при помощи R.....	192
Шаг 1. Создайте DSN для подключения к базе данных SQL Server.....	193
Шаг 2. Создайте таблицу лога в SQL Server	196
Шаг 3. Начните написание скрипта на R для загрузки таблицы <code>DimDate</code>	197
Шаг 4. Создайте переменную для хранения имени загружаемой таблицы	197
Шаг 5. Создайте переменную для хранения SQL-выражения.....	197
Шаг 6. Создайте подключение к SQL Server	197
Шаг 7. Извлеките данные из SQL Server и сохраните их в датафрейм.....	198
Шаг 8. Получите текущее время.....	198
Шаг 9. Получите количество прочитанных записей.....	198
Шаг 10. Добавьте в датафрейм запись с информацией для сохранения в лог.....	198
Шаг 11. Сохраните собранную информацию в базе данных	199
Шаг 12. Закройте соединение	200
Шаг 13. Скопируйте написанный скрипт в Power BI	200
Шаг 14. Создайте скрипт для загрузки таблицы <code>DimProduct</code> на базе <code>ReadLog_DimDate.R</code>	201
Шаг 15. Создайте скрипт для загрузки таблицы <code>DimPromotion</code>	202
Шаг 16. Создайте скрипт для загрузки таблицы <code>DimSalesTerritory</code> на основе <code>ReadLog_DimDate.R</code>	202
Шаг 17. Создайте скрипт для загрузки таблицы <code>FactInternetSales</code> на основе <code>ReadLog_DimDate.R</code>	203
Чтение данных из SQL Server в Power BI при помощи Python	204
Шаг 1. Создайте DSN для SQL Server	204
Шаг 2. Создайте таблицу для ведения логов в SQL Server.....	204
Шаг 3. Создайте скрипт для загрузки таблицы <code>DimDate</code>	205

Шаг 4. Определите переменную для хранения имени таблицы, предназначенной для загрузки в Power BI.....	205
Шаг 5. Создайте подключение к базе данных с помощью библиотеки sqlalchemy	205
Шаг 6. Прочитайте содержимое таблицы DimDate и сохраните его в переменной df_read	206
Шаг 7. Получите текущую дату и время и сохраните в переменной timestamp.....	206
Шаг 8. Посчитайте количество записей в таблице DimDate.....	206
Шаг 9. Добавьте запись в датафрейм с информацией для сохранения логов	207
Шаг 10. Добавьте информацию, добытую на предыдущем шаге, в таблицу логов	207
Шаг 11. Скопируйте скрипт в Power BI	208
Шаг 12. Создайте скрипт для загрузки таблицы DimProduct на основе ReadLog_DimDate.py	208
Шаг 13. Создайте скрипт для загрузки таблицы DimPromotion на основе ReadLog_DimDate.py	209
Шаг 14. Создайте скрипт для загрузки таблицы DimSalesTerritory на основе ReadLog_DimDate.py	210
Шаг 15. Создайте скрипт для загрузки таблицы FactInternetSales на основе ReadLog_DimDate.py	211

Глава 6. Чтение в модель данных Power BI посредством API.....212

Чтение и загрузка данных в Power BI из API с помощью R.....	212
Шаг 1. Получите персональный ключ API к Census	212
Шаг 2. Загрузите необходимые пакеты R	213
Шаг 3. Определите переменные для возврата из вашего набора данных.....	213
Шаг 4. Создайте символьный вектор, содержащий нужные вам переменные	214
Шаг 5. Сконфигурируйте функцию get_acs	215
Шаг 6. Присвойте переменным (столбцам) осмысленные имена	215
Шаг 7. Скопируйте скрипт в Power BI	216
Чтение и загрузка данных в Power BI из API с помощью Python.....	217
Шаг 1. Получите персональный ключ API к Census	217
Шаг 2. Загрузите необходимые библиотеки Python	218
Шаг 3. Определите переменные для возврата из вашего набора данных.....	218
Шаг 4. Создайте список, содержащий нужные вам переменные	219
Шаг 5. Создайте список кортежей с географическими фильтрами для набора данных	220
Шаг 6. Извлеките данные при помощи функции censusdata.download()	220
Шаг 7. Переиндексируйте датафрейм, созданный на шестом шаге.....	221
Шаг 8. Дайте столбцам осмысленные имена	221
Шаг 9. Переименуйте столбцы в датафрейме.....	221
Шаг 10. Скопируйте скрипт в Power BI	222
Заключение	222

Часть III. ПРЕОБРАЗОВАНИЕ ДАННЫХ ПРИ ПОМОЩИ R И PYTHON 223

Глава 7. Продвинутое строковое операции и распознавание шаблонов 224

Защита конфиденциальных сведений	225
Защита конфиденциальных сведений в Power BI с помощью R	225
Шаг 1. Импортируйте пакеты tidyverse и stringr	225
Шаг 2. Напишите функцию для очистки данных	226
Шаг 3. Считайте комментарии в датафрейм	228
Шаг 4. Скройте телефонные номера и номера социального страхования в поле комментариев	228
Шаг 5. Скопируйте скрипт в Power BI	229
Защита конфиденциальных сведений в Power BI с помощью Python	229
Шаг 1. Импортируйте библиотеки pandas, os и re	230
Шаг 2. Напишите функцию mask_text()	230
Шаг 3. Установите рабочую директорию	232
Шаг 4. Считайте комментарии в датафрейм	232
Шаг 5. Выполните замену телефонных номеров и номеров социального страхования	232
Шаг 6. Скопируйте скрипт в Power BI	232
Подсчет количества слов и предложений в обзорах	233
Подсчет количества слов и предложений в обзорах с помощью R	233
Шаг 1. Импортируйте библиотеки tidyverse и stringr	233
Шаг 2. Измените рабочую директорию	234
Шаг 3. Считайте информацию из файла	234
Шаг 4. Ограничьте набор данных требуемыми столбцами	234
Шаг 5. Добавьте столбцы с количеством слов и предложений	234
Шаг 6. Скопируйте скрипт в Power BI	235
Подсчет количества слов в обзорах с помощью Python	235
Шаг 1. Импортируйте библиотеки pandas и os	235
Шаг 2. Установите рабочую директорию	236
Шаг 3. Считайте информацию из файла	236
Шаг 4. Создайте в датафрейме столбец word_count	236
Шаг 5. Скопируйте скрипт в Power BI	236
Удаление имен неподходящего формата	237
Удаление имен неподходящего формата с помощью R	237
Шаг 1. Импортируйте пакеты tidyverse и stringr	237
Шаг 2. Установите рабочую директорию	237
Шаг 3. Создайте регулярное выражение с правильным шаблоном имени	238
Шаг 4. Считайте данные в датафрейм	239
Шаг 5. Выполните обновление столбца Name	239
Шаг 6. Скопируйте скрипт в Power BI	239
Удаление имен неподходящего формата с помощью Python	239
Шаг 1. Импортируйте библиотеки pandas, re и os	240

Шаг 2. Установите рабочую директорию.....	240
Шаг 3. Считайте данные из файла DimEmployee.csv в датафрейм	240
Шаг 4. Создайте регулярное выражение, соответствующее правильному формату имени	240
Шаг 5. Скомпилируйте регулярное выражение	241
Шаг 6. Напишите функцию для проверки имен на совместимость с шаблоном.....	241
Шаг 7. Примените функцию к столбцу, чтобы избавиться от лишних имен	242
Шаг 8. Скопируйте скрипт в Power BI	242
Определение шаблонов в строках на основании условной логики	243
Поиск шаблонов в строках на основании условной логики с помощью R.....	244
Шаг 1. Импортируйте пакеты tidyverse и stringr.....	244
Шаг 2. Установите рабочую директорию.....	244
Шаг 3. Напишите функцию для поиска изделий	245
Шаг 4. Считайте данные из файла ProductionOrders.csv в датафрейм	246
Шаг 5. Добавьте в датафрейм df столбец Monitored Products	246
Шаг 6. Скопируйте скрипт в Power BI	246
Поиск шаблонов в строках на основании условной логики с помощью Python	247
Шаг 1. Импортируйте библиотеки pandas, re и os	247
Шаг 2. Установите рабочую директорию.....	247
Шаг 3. Скомпилируйте регулярное выражение	247
Шаг 4. Напишите функцию для распознавания нужных нам деталей....	248
Шаг 5. Считайте данные в датафрейм Pandas	248
Шаг 6. Создайте новый столбец с именем Monitored Products.....	249
Шаг 7. Скопируйте скрипт в Power BI.....	249
Заключение	249
Глава 8. Вычисляемые столбцы с помощью R и Python	250
Создание ключа Google Geocoding API	251
Шаг 1. Зайдите в консоль Google.....	251
Шаг 2. Настройте учетную запись	251
Шаг 3. Добавьте новый проект	251
Шаг 4. Активируйте API геокодирования	252
Геокодирование адресов с помощью R.....	254
Геокодирование адресов с помощью Python	256
Вычисление расстояния между точками с помощью пользовательской функции в R	258
Вычисление расстояния между точками с помощью пользовательской функции в Python.....	260
Вычисление расстояния между точками с помощью готовой функции в R...	263
Вычисление расстояния между точками с помощью готовой функции в Python.....	264
Заключение	266

Часть IV. МАШИННОЕ ОБУЧЕНИЕ И ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ В POWER BI ПРИ ПОМОЩИ R И PYTHON267

Глава 9. Применение методов машинного обучения и искусственного интеллекта к моделям данных Power BI..... 268

Применение алгоритмов машинного обучения к набору данных перед загрузкой в модель Power BI..... 269

 Прогнозирование цен на недвижимость с помощью R..... 270

 Шаг 1. Пусть аналитик данных сохранит для вас модель 270

 Шаг 2. Загрузите пакет tidyverse..... 270

 Шаг 3. Загрузите объект модели и набор данных для оценки 271

 Шаг 4. Ограничьте датафрейм столбцами, необходимыми для вашей модели..... 271

 Шаг 5. Примените модель машинного обучения к своему набору данных для составления прогноза цен на недвижимость 271

 Шаг 6. Добавьте прогноз к исходному набору данных 272

 Шаг 7. Скопируйте скрипт в Power BI..... 272

 Прогнозирование цен на недвижимость с помощью Python 272

 Шаг 1. Пусть аналитик данных сохранит для вас модель 273

 Шаг 2. Загрузите необходимые библиотеки 273

 Шаг 3. Загрузите объект модели и актуальный набор данных 273

 Шаг 4. Извлеките нужную информацию из датафрейма 274

 Шаг 5. Примените модель к подготовленному набору данных для расчета прогноза 274

 Шаг 6. Добавьте прогнозные данные к исходному набору данных 275

 Шаг 7. Скопируйте скрипт в Power BI..... 275

Использование готовых моделей ИИ для расширения функционала моделей данных в Power BI..... 276

 Настройка Cognitive Services в Azure 277

 Виртуальная машина для анализа данных (Data Science Virtual Machine – DSVM) 277

 Анализ тональности текста в Microsoft Cognitive Services при помощи Python..... 278

 Шаг 1. Загрузите набор данных с отзывами Yelp с сайта Kaggle 279

 Шаг 2. Загрузите нужные библиотеки, модули и функции для скрипта..... 279

 Шаг 3. Инициализируйте переменные для работы скрипта..... 280

 Шаг 4. Считайте фрагмент файла с отзывами в датафрейм 280

 Шаг 5. Преобразуйте датафрейм к формату, приемлемому для службы Microsoft Cognitive Services 281

 Шаг 6. Оцените отзывы посетителей при помощи метода sentiment() 281

 Шаг 7. Создайте датафрейм, содержащий оценки отзывов 282

 Шаг 8. Скопируйте скрипт в Power BI 282

Применение сторонних моделей машинного обучения к моделям данных Power BI 283

Конфигурирование средства анализа настроения текста в IBM Watson	284
Шаг 1. Заведите аккаунт в IBM Cloud	284
Шаг 2. Выполните вход в IBM Cloud	284
Шаг 3. Перейдите на страницу Tone Analyzer	284
Шаг 4. Настройте службу Tone Analyzer	284
Шаг 5. Получите ключ API	285
Написание скрипта на Python для анализа настроения текста в IBM Watson	286
Шаг 1. Импортируйте необходимые библиотеки и модули	286
Шаг 2. Создайте экземпляр класса IAMAuthenticator	286
Шаг 3. Создайте экземпляр класса ToneAnalyzerV3	286
Шаг 4. Установите ссылку на службу для созданного объекта	287
Шаг 5. Создайте датафрейм с исходными данными для анализа	287
Шаг 6. Создайте основу для датафрейма с оценочными данными	287
Шаг 7. Определите циклическую конструкцию для отправки документов в службу IBM Watson	288
Шаг 8. Отформатируйте и оцените настроение текста в документе	288
Шаг 9. Извлеките результат анализа текста и инициализируйте переменные исходными значениями	289
Шаг 10. Пройдите по тонам и присвойте их значения соответствующим переменным	290
Шаг 11. Создайте датафрейм на основе списка listReturnedUtterance	291
Шаг 12. Объедините датафреймы dfReturnedUtterance и dfDocuments	291
Шаг 13. Скопируйте скрипт в Power BI	292

Глава 10. Создание моделей анализа данных и скриптов для обработки информации

Прогнозирование цен на недвижимость в Power BI с помощью R со службой SSMLS	294
Написание скрипта на языке R для добавления модели в SQL Server	295
Шаг 1. Загрузите необходимые пакеты	296
Шаг 2. Загрузите модель R в вашу сессию	296
Шаг 3. Подключитесь к базе данных	296
Шаг 4. Определите переменные модели	296
Шаг 5. Напишите выражение на T-SQL для добавления модели в базу данных	297
Шаг 6. Добавьте код, необходимый для запуска выражения T-SQL из R	297
Шаг 7. Сохраните скрипт	298
Использование SSMLS совместно с R для оценки данных	298
Шаг 1. Запустите SQL Server Management Studio	298
Шаг 2. Создайте подключение к серверу, который хотите использовать	298
Шаг 3. Добавьте базу данных BostonHousingInfo на ваш сервер	298
Шаг 4. Добавьте модель в базу данных	299
Шаг 5. Создайте в базе данных хранимую процедуру для прогноза	299

Шаг 6. Извлеките данные с прогнозами из SQL Server в Power BI.....	301
Прогнозирование цен на недвижимость в Power BI с помощью Python со службой SSMLS.....	303
Написание скрипта на языке Python для добавления модели в SQL Server.....	303
Шаг 1. Подберите версии библиотек.....	303
Шаг 2. Создайте окружение conda.....	304
Шаг 3. Напишите код для загрузки модели в SQL Server.....	305
Использование SSMLS совместно с Python для оценки данных.....	307
Шаг 1. Запустите SQL Server Management Studio.....	307
Шаг 2. Создайте подключение к серверу, который хотите использовать.....	307
Шаг 3. Добавьте базу данных BostonHousingInfo на ваш сервер.....	307
Шаг 4. Добавьте модель в базу данных.....	307
Шаг 5. Создайте в базе данных хранимую процедуру для прогноза.....	308
Шаг 6. Извлеките данные с прогнозами из SQL Server в Power BI.....	310
Анализ тональности текста в Power BI с помощью R со службой SSMLS.....	311
Добавление готовых моделей R в SSMLS с помощью PowerShell.....	311
Шаг 1. Проверьте, установлены ли предварительно обученные модели.....	311
Шаг 2. Откройте PowerShell от имени администратора.....	312
Шаг 3. Загрузите скрипт PowerShell.....	312
Шаг 4. Запустите загруженный скрипт в PowerShell.....	312
Решение проблем.....	312
Использование готовой модели R в SSMLS для анализа тональности текста в Power BI.....	312
Шаг 1. Определите хранимую процедуру.....	313
Шаг 2. Определите переменные.....	313
Шаг 3. Инициализируйте переменную @Query.....	313
Шаг 4. Инициализируйте переменную @RScript.....	314
Шаг 5. Сконфигурируйте процедуру sp_execute_external_script.....	315
Шаг 6. Определите выходные данные.....	316
Шаг 7. Создайте хранимую процедуру в базе данных.....	316
Шаг 8. Вызовите процедуру из Power BI.....	317
Анализ тональности текста в Power BI с помощью Python со службой SSMLS.....	318
Добавление готовых моделей Python в SSMLS.....	319
Шаг 1. Проверьте, установлены ли предварительно обученные модели.....	319
Шаг 2. Откройте PowerShell от имени администратора.....	319
Шаг 3. Загрузите скрипт PowerShell.....	319
Шаг 4. Запустите загруженный скрипт в PowerShell.....	319
Решение проблем.....	320
Использование готовой модели Python в SSMLS для анализа тональности текста в Power BI.....	320
Шаг 1. Определите хранимую процедуру.....	320
Шаг 2. Определите переменные.....	321

Шаг 3. Инициализируйте переменную @Query	321
Шаг 4. Инициализируйте переменную @PythonScript.....	321
Шаг 5. Сконфигурируйте процедуру sp_execute_external_script	322
Шаг 6. Определите выходные данные	322
Шаг 7. Создайте хранимую процедуру в базе данных.....	322
Шаг 8. Вызовите процедуру из Power BI.....	323
Вычисление расстояния между точками в Power BI с помощью R со службой SSMLS.....	324
Шаг 1. Убедитесь, что в SSMLS загружен пакет dplyr.....	325
Шаг 2. Запустите SSMS и подключитесь к SQL Server	325
Шаг 3. Добавьте базу данных CalculateDistance на ваш сервер.....	325
Шаг 4. Создайте хранимую процедуру для расчета расстояний.....	326
Шаг 5. Вызовите процедуру из Power BI	328
Вычисление расстояния между точками в Power BI с помощью Python со службой SSMLS.....	329
Шаг 1. Запустите SSMS и подключитесь к SQL Server	330
Шаг 2. Добавьте базу данных CalculateDistance на ваш сервер.....	330
Шаг 3. Создайте хранимую процедуру для расчета расстояний.....	331
Шаг 4. Вызовите процедуру из Power BI	333
Предметный указатель.....	335



От издательства



Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Скачивание исходного кода примеров

Скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте www.dmkpress.com на странице с описанием соответствующей книги.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг, мы будем очень благодарны, если вы сообщите о ней главному редактору по адресу dmkpress@gmail.com. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Maker Media очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу электронной почты dmkpress@gmail.com.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Об авторе



Райан Уэйд (Ryan Wade) является профессиональным аналитиком данных с более чем 20-летним стажем. Своему образованию и опыту работы Райан обязан тем, что приобрел целостное понимание аналитических процессов как с технической, так и с организационной точки зрения. Является обладателем сертификата MCSE в области бизнес-аналитики и Microsoft R и профессионально программирует на R, Python, DAX, T-SQL, M и VBA применительно к локальным и облачным решениям в аналитике на платформе данных Microsoft (Microsoft Data Platform).

Райан принимает активное участие в открытых мероприятиях по R и Python, а также выступает на конференциях SQLSaturdays, TDWI, BDPA и PASS Summit с лекциями по анализу данных. Разработал полноценный онлайн-курс для ExcelTv, в рамках которого демонстрирует применение языка R в Power BI для проведения углубленного анализа данных и их визуализации.



О техническом редакторе

Аадития Марутхи (Aaditya Maruthi) является ведущим специалистом в области разработки баз данных в крупной компании. Обладает более чем 10-летним стажем работы с различными СУБД, включая Microsoft SQL Server и Oracle. По большей части работал с технологиями от Microsoft, такими как SSAS, SSRS, SSIS и Power BI. Аадития также является обладателем сертификата Certified AWS Solutions Architect Associate.



Благодарности

Хочу поблагодарить технических редакторов Майка Хаффера (Mike Huffer) и Аадитию Марутхи (Aaditya Maruthi) за качественную обратную связь, которая помогла сделать эту книгу лучше. Также хотел бы сказать спасибо Джонатану Геннику (Jonathan Gennick) и Джилл Бальцано (Jill Balzano). Признателен вам за ваше терпение и помощь. Без ваших напутствий мне было бы очень непросто завершить начатое. Кроме того, вы постоянно держали руку на пульсе проекта. Изначально я хотел включить в книгу слишком много всего, и в этом случае ее написание заняло бы невероятно много времени. Вы помогли мне понять, что действительно важно, и это позволило завершить работу в приемлемые сроки.



Введение

Microsoft Power BI является одним из наиболее популярных инструментов в области бизнес-аналитики. За последние годы этот программный комплекс опередил своих прямых конкурентов QlikView и Tableau и прочно занял лидирующее положение на рынке. Одним из главных преимуществ Power BI является то, что он представляет собой нечто гораздо большее, чем просто инструмент визуализации данных. Вот лишь несколько явных преимуществ Power BI:

- встроенный язык запросов *DAX*, позволяющий крайне эффективно извлекать информацию из модели данных с применением сложной бизнес-логики;
- интегрированный инструмент подготовки и преобразования данных *Power Query*, при помощи которого можно легко извлекать и трансформировать исходную информацию в вид, пригодный для анализа;
- движок *Vertipaq*, позволяющий хранить данные в оптимальном для формирования отчетов виде и быстро и эффективно обрабатывающий сложные вычисления;
- заранее подготовленные пакеты интерактивных элементов визуализации, помогающие представлять данные в понятной и четкой форме.

Глядя на этот список преимуществ, вы вполне можете задаться вопросом, зачем же столь мощному инструменту понадобилась помощь языков программирования R и Python. Ответ прост – чтобы заполнить области, в которых встроенные средства недостаточно хороши. Вот лишь несколько примеров применения этих языков программирования в рамках Power BI:

- создание пользовательских элементов визуализации без особых усилий;
- реализация интеллектуальной обработки данных, методов машинного обучения и искусственного интеллекта без необходимости приобретать дорогостоящую подписку на *Power BI Premium*;
- применение продвинутых методов обработки текстовой информации с использованием техник, недоступных в *Power Query* и *DAX*;
- взаимодействие со службами *Microsoft Cognitive Services* без необходимости приобретать подписку на *Power BI Premium*;
- взаимодействие со сторонними интерфейсами API с целью эффективного обогащения моделей данных Power BI;
- и многое другое...

В данной книге мы подробно расскажем о том, как использовать на практике языки программирования R и Python для обеспечения всей перечисленной выше функциональности в Power BI. Язык R идеально подходит для Power BI по причине того, что он был создан специально для анализа данных. Уже долгие годы аналитики активно используют R для преобразования и визуализации информации. Так что то небольшое, на что не способна программная среда Power BI, может быть с лихвой компенсировано при помощи языка R.

Что касается Python, то этот язык программирования приобрел чрезвычайную популярность в области анализа данных в последнее десятилетие. Одним

из главных преимуществ Python является то, что он подходит для решения не только аналитических задач, но и общих задач программирования. К примеру, взаимодействие с интерфейсами API довольно легко осуществить при помощи Python, тогда как посредством Power Query это будет сделать не так-то просто.

Все эти особенности делают языки R и Python идеально подходящими для такого мощного аналитического инструмента, как Power BI. И в книге, которую вы держите в руках, мы проиллюстрируем на примерах все перечисленные выше возможности и техники. При этом все решения и сценарии будут сопровождаться подробными описаниями, чтобы вы досконально поняли используемую реализацию.

Но перед тем как приступить к конкретным примерам, необходимо для начала настроить среду выполнения. Давайте пройдемся по соответствующим пунктам и подготовимся к работе.

НАСТРОЙКА ВАШЕГО ОКРУЖЕНИЯ AZURE

В данной книге мы будем часто упоминать и пользоваться различными составляющими облачной платформы *Microsoft Azure*. В частности, для внедрения механизмов искусственного интеллекта в модели данных Power BI мы будем использовать набор служб *Microsoft Cognitive Services*. Также для работы с примерами из данной книги рекомендуется настроить *виртуальную машину для анализа данных* (Data Science Virtual Machine – DSVM). Это не обязательное условие, но желательное. Кроме того, для комфортной работы с тем, что мы будем обсуждать, вам необходимо настроить окружение со следующими инструментами:

- SQL Server 2017 или выше;
- SQL Server Machine Learning Services 2017 или выше с поддержкой R и Python;
- дистрибутив Python из Anaconda;
- R;
- R Studio;
- VS Code;
- Power BI Desktop.

Сконфигурировать такое окружение вручную – задача не из легких, но если вы установите виртуальную машину для анализа данных, большинство настроек будет выполнено за вас автоматически. В следующих разделах мы поговорим о том, как настроить *Azure*, чтобы можно было пользоваться службами *Microsoft Cognitive Services*. Кроме того, вы узнаете, как развернуть *DSVM*.

Подписка на Azure



Оформить подписку на *Azure* можно по адресу <https://azure.microsoft.com/en-us/free>. В результате вы получите 12 месяцев на пользование выборочными службами плюс кредит на сумму \$200 на первый месяц.

Подписка на Microsoft Cognitive Services

Мы будем использовать службу *Microsoft Cognitive Services* для проведения анализа тональности текста (sentiment analysis) в Power BI с помощью Python. Для начала вам необходимо будет настроить службу *Microsoft Cognitive Services* в Azure, после чего можно будет обращаться к ней из Power BI. Для настройки службы нужно выполнить следующие шаги.

1. Войдите на портал Azure.
2. Введите в строке поиска *Cognitive Services* и нажмите **Enter**.
3. Вы должны оказаться на странице *Cognitive Services*. Нажмите кнопку создания для запуска процесса регистрации.
4. Введите следующую информацию:
 - имя;
 - подписка;
 - расположение;
 - ценовая категория;
 - группа ресурсов.
5. Установите флажок, оповещающий о том, что вы прочитали и согласны с условиями.
6. Нажмите кнопку создания.

Учтите, что использование службы *Microsoft Cognitive Services* является платным. Чтобы получить информацию о ценах, перейдите к ресурсу *Microsoft Cognitive Services*, введите в окно поиска текст *Ценовая категория* и откройте результаты поиска. Выберите подходящую вам категорию. Вы будете перенаправлены на страницу с информацией о ценах согласно выбранному вами региону. На примеры, показанные в данной книге, вам с лихвой хватит выданного вам на первый месяц кредита.

Создание виртуальной машины для анализа данных (DSVM)

Предпочтительным вариантом будет создание виртуальной машины и добавление ресурсов, не установленных в ней по умолчанию. Я рекомендую идти этим путем, поскольку полностью ручная настройка окружения для выполнения примеров из этой книги может занять уйму времени. Использование виртуальной машины позволит сконфигурировать необходимое окружение за несколько минут – притом что в ручном режиме у вас бы это могло отнять много дней проб и ошибок. Если вы решите пойти длинным путем, позже я опишу примерный план действий. Сейчас же приведу инструкции по настройке виртуальной машины.

Шаги создания виртуальной машины в Azure

1. Перейдите на портал <https://portal.azure.com>. Если система попросит, введите данные учетной записи, созданной ранее.

2. Нажмите на кнопку **Создать ресурс** (Create a resource) в левом верхнем углу.
3. В строке поиска введите **Data Science Virtual Machine – Windows 2019**.
4. Нажмите на кнопку **Создать** (Create). Появится форма для ввода информации о конфигурации виртуальной машины, в которой будет открыта вкладка **Основные** (Basics). В следующих шагах вы узнаете, как заполнить эту форму.
5. **Подписка** (Subscription): выберите подписку, которую собираетесь использовать. По умолчанию будет выбрана подписка, настроенная ранее.
6. **Группа ресурсов** (Resource group): если у вас уже есть группа ресурсов, которую вы желаете использовать, выберите ее. В противном случае создайте новую для своей виртуальной машины.
7. **Имя виртуальной машины** (Virtual machine name): название, которое вы хотите присвоить виртуальной машине.
8. **Регион** (Region): ближайший к вам географический регион Azure.
9. **Image**: убедитесь, что в данном списке выбран пункт **Data Science Virtual Machine – Windows 2016**.
10. **Размер** (Size): я использую B4ms, поскольку это самый дешевый вариант для 16 Гб оперативной памяти. Этот параметр очень важен для R, Python и Power BI.
11. **Имя пользователя** (Username): придумайте имя пользователя.
12. **Пароль** (Password): введите пароль.
13. **Подтвердите пароль** (Confirm password): введите пароль еще раз.
14. Перейдите на вкладку **Диски** (Disks).
15. **Тип диска ОС** (OS disk type): укажите **Стандартный SSD** (Standard SSD) – это будет оптимальный для нашего случая.
16. Перейдите на вкладку **Сетевые подключения** (Networking).
17. Убедитесь, что все нужные поля заполнены. Обязательные к заполнению поля помечены звездочкой. В эти поля должны быть введены значения по умолчанию. Если таких значений нет, нажмите на ссылку **Создать** (Create new) под соответствующим полем и введите недостающий элемент.
18. Перейдите на вкладку **Управление** (Management).
19. Оставьте все по умолчанию и перейдите на вкладку **Дополнительно** (Advanced).
20. Примите значения по умолчанию и перейдите на вкладку **Теги** (Tags).
21. Откройте вкладку **Просмотр и создание** (Review + create).
22. Вы увидите сводную информацию о создаваемой виртуальной машине. Кроме того, для вас будет рассчитана стоимость ее использования. Если вы согласны с введенными данными, нажмите кнопку **Создать** (Create).

Не забывайте останавливать виртуальную машину после каждого использования, чтобы не платить лишние деньги. На всякий случай вы можете настроить автоматическое отключение машины в определенное время. Для этого необходимо сделать следующее.

1. Перейдите к своей виртуальной машине на портале Azure.
2. Введите текст **Автозавершение работы** (auto-shutdown) в строку поиска и перейдите в соответствующий раздел.
3. Переведите переключатель **Включено** (Enabled) в положение **Вкл** (On).
4. Выберите время, в которое машина будет выключаться, в поле **Запланированное завершение работы** (Scheduled shutdown).
5. В выпадающем списке **Часовой пояс** (Time zone) выберите нужную зону.
6. Если хотите, чтобы система отправляла вам уведомление о выключении виртуальной машины, установите переключатель в разделе **Отправлять уведомление перед автоматическим завершением работы** (Send notification before auto-shutdown) в положение **Да** (Yes). Уведомление будет отправлено на адрес, введенный в поле **Адрес электронной почты** (Email address).

Настройка R на виртуальной машине

Мы будем использовать другой дистрибутив языка R по сравнению с установленным. Нашим выбором будет дистрибутив *Microsoft R Open (MRO)*. Он полностью совместим с дистрибутивом, распространяемым через центральную систему хранения пакетов CRAN, но при этом существенно улучшен в отношении определенных типов вычислений, а также снабжен дополнительными полезными инструментами. Выполните следующие шаги, чтобы загрузить дистрибутив *MRO* в виртуальную машину.

1. Узнайте версию R, используемую в Power BI. Соответствующую информацию можно найти на сайте Microsoft по адресу <https://docs.microsoft.com/en-us/power-bi/visuals/service-r-visuals>.
2. Откройте браузер в виртуальной машине и перейдите по следующей ссылке: <https://mran.microsoft.com/open>. В виртуальной машине по умолчанию установлены два браузера: *Microsoft Edge* и *Firefox*.
3. Нажмите на кнопку **Download** справа, и вы будете перенаправлены на страницу загрузок. Здесь щелкните по ссылке **Past Releases** справа, которая откроет страницу со всеми предыдущими версиями *Microsoft R Open*. Выберите версию, которую использует Power BI.
4. Нажмите на кнопку **Download** напротив версии для *Windows*.
5. Выполните установку загруженного дистрибутива.
6. Откройте R Studio на виртуальной машине.
7. Откройте меню **Tools => Global Options** и убедитесь, что выбрана версия MRO, которую вы только что установили. Если это не так, нажмите на кнопку **Change...** и выберите нужный дистрибутив из списка, после чего нажмите на кнопку **OK**.

Настройка Python на виртуальной машине

Одним из преимуществ использования виртуальной машины является то, что вы получаете предустановленный дистрибутив Python, идеально под-

ходящий для анализа данных. Этот дистрибутив называется *Anaconda*. Он поставляется с более чем 1500 библиотек, популярных в среде анализа данных. Также вместе с ним идет *диспетчер пакетов* (package manager) и *система управления окружением* (environment management system) под названием *conda*. Инсталлировать пакеты предпочтительно именно посредством *conda* по причине установки правильных зависимостей между ними. Система управления окружением *conda* значительно облегчает задачу создания изолированной копии Python с предустановленными библиотеками нужных версий.

Давайте для примеров из этой книги создадим отдельное окружение Python с именем *pbi*. Для этого необходимо выполнить следующие действия.

1. Подключитесь к виртуальной машине.
2. Откройте командную строку, нажав на значок поиска рядом с иконкой Windows и введя команду *cmd*.
3. Введите следующую команду для создания окружения *conda* с именем *pbi* на базе Python 3.7:

```
conda create -n pbi python=3.7
```

Решение использовать Python версии 3.7 основывается на информации, полученной из инструкции от Microsoft по адресу <https://docs.microsoft.com/en-us/business-applications-release-notes/october18/intelligence-platform/power-bi-service/pervasive-artificial-intelligence-bi/python-service>. Согласно документации, службы Power BI совместимы с Python 3.x, так что текущая версия 3.x должна подойти.

Теперь у нас есть окружение Python, которое мы можем использовать для примеров из данной книги.



Настройка SQL Server Machine Learning Services на виртуальной машине

В нескольких примерах из этой книги вам потребуется наличие служб машинного обучения SQL Server (*SQL Server Machine Learning Services – SSMLS*). *SSMLS* предоставляет вам инструменты, позволяющие проводить углубленную аналитику в базах данных с использованием R и Python, а также средства, облегчающие работу с большими данными. Еще в составе *SSMLS* есть несколько предварительно обученных моделей от Microsoft, которыми вы можете пользоваться в процессе учебы и работы. В виртуальной машине по умолчанию запущены службы *SSMLS*. Если вы не используете виртуальную машину, вам необходимо будет вручную запустить эти службы, воспользовавшись подробной инструкцией по адресу <https://docs.microsoft.com/en-us/sql/machine-learning/install/sql-machine-learning-services-windows-install?view=sql-server-ver15>. Предварительно обученные модели, которые мы будем использовать в этой книге, могут быть добавлены в ваш экземпляр SQL Server поверх базовой установки. Перейдите по следующей ссылке и следуйте инструкциям: <https://docs.microsoft.com/en-us/sql/machine-learning/install/sql-pretrained-models-install?view=sql-server-ver15>.

Установка пакетов R

Некоторые скрипты на языке R из этой книги могут ссылаться на пакеты, которые у вас изначально могут быть не установлены. Исправить это очень просто. Следующая инструкция в консоли R позволит установить популярный пакет с названием *data.table*:

```
install.packages("data.table")
```

Бывает, что вам требуется установить сразу несколько пакетов за раз. Например, вы хотите одновременно установить пакеты *data.table* и *dplyr*. Для этого достаточно объединить названия этих пакетов в вектор и присвоить результат переменной *pkgs*. Затем можно передать эту переменную на вход функции *install.packages()*, как показано ниже:

```
pkgs <- c("data.table", "dplyr")
install.packages(pkgs)
```

Примечание. Символьный вектор представляет собой тип данных в языке R для хранения текстовой информации в виде одномерного массива. Вы узнаете больше об этом и других типах данных в R в процессе чтения книги.

При создании визуальных элементов в Power BI при помощи R вам необходимо знать, какую версию пакета использует служба Power BI. Вы можете получить список всех доступных пакетов R в Power BI вместе с их версиями по ссылке <https://docs.microsoft.com/en-us/power-bi/service-r-packages-support>.

При помощи функции *install.packages()* можно установить последнюю версию пакета из репозитория, который вы используете, если у вас дистрибутив R из CRAN. При использовании дистрибутива *Microsoft R Open* будет установлена последняя версия пакета, основываясь на *дате снимка* (snapshot date). В обоих случаях может получиться так, что будет установлен пакет не той версии, с которой работает служба. Чтобы устранить это неудобство, необходимо сначала узнать требуемую версию пакета по ссылке выше, после чего загрузить ее при помощи пакета *devtools*. Ниже приведен пример использования пакета *devtools* для установки пакета *ggplot2* версии 0.9.1 из CRAN:

```
library(devtools)
install_version(
  "ggplot2",
  version = "0.9.1",
  repos = "http://cran.us.r-project.org")
```

Установка библиотек Python

Установить библиотеки Python можно разными способами. В данной книге мы будем использовать два метода: при помощи *conda* и при помощи *pip*.

Стоит отметить, что установка библиотек в Python выполняется не так просто, как в R. В книге мы будем в основном использовать командную строку *conda* для установки библиотек Python. Для этого необходимо выполнить следующие действия.

1. Откройте строку поиска, иконка которой расположена рядом со значком *Windows*.
2. Введите слово *Anaconda*, после чего в окне выбора появится вариант *Anaconda Prompt*. Щелкните по нему.
3. Активируйте окружение, которое используете для Power BI, введя следующую команду в командную строку:

```
conda activate "<environment name>"
```

Рекомендуется использовать окружение для разработки на Python, ассоциированное с этой книгой.

4. Установите пакет при помощи *conda*, используя следующий шаблон:

```
conda install <"package name">
```

Например, если вы устанавливаете пакет *pandas*, команда должна иметь следующий вид:

```
conda install pandas
```

Если вам необходимо установить пакет *pandas* версии 1.0.4, используйте следующую команду:

```
conda install pandas=1.0.4
```

Не все пакеты доступны для установки при помощи *conda*. Обратитесь к следующей ссылке для получения списка пакетов, которые могут быть установлены с использованием *conda* в Python 3.6: https://docs.anaconda.com/anaconda/packages/py3.6_win-64. Один из пакетов, который мы будем использовать в этой книге, недоступен в *conda*, но может быть установлен при помощи *PyPI*. Имя этого пакета *CensusData*. Вам необходимо использовать систему управления пакетами *pip* для установки библиотеки *CensusData*, как показано ниже:

```
pip install CensusData
```

Настройка Power BI на виртуальной машине

В *Power BI Desktop* необходимо выполнить ряд изменений в настройках, чтобы можно было работать с R и Python. Подробно эти изменения описаны в репозитории кода книги на *GitHub*. Здесь вы также найдете инструкции по созданию и использованию окружения *conda* в Python. Ссылка на репозиторий кода: <https://github.com/Apress/adv-analytics-in-power-bi-w-r-and-python>.



АЛЬТЕРНАТИВНАЯ НАСТРОЙКА

Использование виртуальной машины – предпочтительная, но вовсе не обязательная опция для работы с примерами из этой книги. Если вы хотите пойти другим путем, вам придется устанавливать все программное обеспечение вручную. Приводим ссылки на все необходимые установки:

- Power BI: <http://www.microsoft.com/en-us/download/details.aspx?id=58494>;
- R Studio: <https://rstudio.com/products/rstudio/download>;
- Microsoft R Open: <https://mran.microsoft.com/download>;
- Anaconda: <http://www.anaconda.com/products/individual>;
- VS Code: <https://code.visualstudio.com/download>;
- SQL Server 2019 Developer: <http://www.microsoft.com/en-us/sql-server/sql-server-downloads>;
- SQL Server Machine Learning Services: <https://docs.microsoft.com/en-us/sql/machine-learning/install/sql-machine-learning-services-windows-install?view=sql-server-ver15>.

Если вы остановите выбор на этом варианте, я очень рекомендую использовать виртуальную машину на базе *Windows Server 2016* или выше. Предлагаю ссылку YouTube на пошаговую инструкцию по установке *Windows Server 2019* на *VirtualBox*: <http://www.youtube.com/watch?v=ZjQSuyuN0nA&t=8s>.

ЗАГРУЗКА ПАКЕТОВ R в SSMLS

В главе 10 вы научитесь работать с моделями машинного обучения посредством служб *SQL Server Machine Learning Services 2019* с использованием языка R. И для этого вам понадобится, чтобы необходимые пакеты были загружены в *SSMLS 2019*. Ниже представлен скрипт на языке T-SQL, который вы можете использовать для вывода информации о том, какие пакеты в данный момент загружены в ваш экземпляр *SSMLS 2019*:

```
EXECUTE sp_execute_external_script
    @language=N'R',
    @script = N'
packagematrix <- installed.packages();
Name <- packagematrix[,1];
Version <- packagematrix[,3];
OutputDataSet <- data.frame(Name, Version);'

WITH RESULT SETS ((PackageName nvarchar(250), PackageVersion nvarchar(max)) )
```

Если пакета, который вам нужен, нет в списке, вам необходимо будет загрузить его вручную. Для этого нужно выполнить следующую пошаговую инструкцию.

Шаг 1. Загрузите пакет *sqlmlutils* в папку *Documents*

Пакет *sqlmlutils* можно загрузить по адресу <https://github.com/Microsoft/sqlmlutils/tree/master/R/dist>.

Скачайте файл zip с репозитория GitHub и сохраните в папке *Documents*.

Шаг 2. Запустите следующий код из командной строки

Откройте командную строку под администратором и запустите следующий код на выполнение:

```
R -e "install.packages('RODBCext', repos='https://cran.microsoft.com')"
```

```
R CMD INSTALL %UserProfile%\Documents\sqlmlutils_0.7.1.zip
```

Этот код сработает, если вы предварительно положили архив *sqlmlutils* в папку *Documents* под вашим профилем. Если файл располагается в другом месте, вам необходимо откорректировать путь.

Шаг 3. Загрузите необходимые пакеты

После выполнения второго шага инструкции вы будете готовы к загрузке пакетов в *SSMLS 2019* из скрипта R в R Studio. Ниже приведем фрагмент кода для загрузки пакета *dplyr* в *SSMLS 2019*:

```
library(sqlmlutils)
connection <- connectionInfo(
  server = "server",
  database = "database",
  uid = "username",
  pwd = "password")

sql_install.packages(connectionString = connection,
  pkgs = "dplyr", verbose = TRUE, scope = "PUBLIC")
```

Вы можете загружать несколько пакетов одновременно. Скажем, вам необходимо загрузить пакеты *dplyr* и *data.table*. Это можно сделать путем создания символьного вектора, содержащего оба пакета, и передачи его параметру *pkgs*, как показано ниже:

```
library(sqlmlutils)
connection <- connectionInfo(
  server = "<server>",
  database = "<database>",
  uid = "<username>",
  pwd = "<password>")

pkgList <- c("dplyr", "data.table")
sql_install.packages(connectionString = connection,
  pkgs = pkgList, verbose = TRUE, scope = "PUBLIC")
```

ЗАГРУЗКА НЕОБХОДИМЫХ БИБЛИОТЕК PYTHON В SSMLS

Как и в случае с загрузкой пакетов R в SQL Server Machine Learning Services 2019, вы должны знать, как установить необходимые пакеты Python при помощи *sqlmlutils* для чтения заключительных глав этой книги. Для этого вам нужно выполнить следующие шаги.

Шаг 1. Скачайте пакет *sqlmlutils* на свой компьютер в папку *Documents*

Загрузить пакет *sqlmlutils* можно по ссылке <https://github.com/Microsoft/sqlmlutils/tree/master/Python/dist>.

Скачайте архив zip и сохраните его в папке *Documents*.

Шаг 2. Откройте командную строку и введите следующие инструкции

```
pip install "pymssql<3.0"
pip install --upgrade --upgrade-strategy only-if-needed c:\temp\sqlmlutils-0.7.2.zip
```



Шаг 3. Загрузите необходимые пакеты

После выполнения шага 2 вы сможете загружать нужные вам пакеты в *SSMLS 2019* посредством запуска скрипта на Python в *VS Code*. Ниже представлен фрагмент кода, позволяющий загрузить библиотеку *pandas* в *SSMLS 2019*:

```
import sqlmlutils
connection = sqlmlutils.ConnectionInfo(
    server="<имя сервера>", database="<база данных>",
    uid="<имя пользователя>", pwd="<пароль>")
sqlmlutils.SQLPackageManager(connection).install("pandas")
```



Локальный шлюз данных

Локальный шлюз данных (on-premises data gateway) представляет собой инструмент для обеспечения безопасной передачи данных между локальными источниками данных и облаком Azure. Локальный шлюз может быть запущен в двух режимах: персональном и стандартном. Если вы захотите развернуть решения, рассматриваемые в главах с третьей по девятую, в службе Power BI и запускать их с определенной периодичностью, вам понадобится установить локальный шлюз данных в персональном режиме. На момент написания книги скрипты на R и Python, используемые в Power Query, могут быть запущены через локальный шлюз только в персональном режиме. Недостатком

использования шлюза в таком режиме является то, что ваше решение нельзя будет назвать корпоративным, поскольку доступ к нему будет, как ясно из названия режима, только у вас.

Но в главе 10 вы узнаете, как можно использовать R и Python в корпоративном решении на базе Power BI через *SQL Server Machine Learning Services (SSMLS)*. При использовании служб *SSMLS 2019* ваш код на R и Python будет заключен в специальную хранимую процедуру на языке T-SQL. А хранимые процедуры допустимо использовать в стандартном режиме локального шлюза. В главе 10 вы также познакомитесь с основами рефакторинга кода на R и Python, описанного в предыдущих главах, под специальные хранимые процедуры, используемые в службах *SSMLS 2019*. Заметьте, что визуальные элементы R не полагаются на локальный шлюз, поскольку они обрабатываются при помощи экземпляра R непосредственно в службе Power BI.

Источники информации

В данной книге описываются разнообразные технологии, и было бы невозможно досконально рассказать о них всех. Понимая это, я решил предоставить вам наиболее полный список литературы для самостоятельного изучения тем, которые вам пока незнакомы. Также я включил ссылку на репозиторий с исходными кодами, используемыми в книге, и дополнил список ссылками на полезные ресурсы.

Репозиторий книги

Исходный код для всех упражнений из книги собран в едином репозитории по адресу <https://github.com/Apress/adv-analytics-in-power-bi-w-r-and-python>. Полные скрипты на R и Python сгруппированы в хранилище по главам и темам. В репозитории также представлены актуальные источники данных, используемые в примерах, или информация о получении доступа к ним.

Ресурсы по R



Книги:

- *R for Data Science*: прекрасная книга, в которой о языке R рассказывает один из самых плодотворных создателей пакетов Хэдли Уикхэм (Hadley Wickham). В ней в том числе описываются полезные пакеты из авторской библиотеки *tidyverse*. Книга доступна бесплатно по адресу <https://r4ds.had.co.nz>;
- *An Introduction to Statistical Learning: With Applications in R*: эта книга описывает базовые принципы статистики, без которых не обойтись при изучении машинного обучения с R. Книге уже несколько лет, но она не утратила популярности в сообществе языка R. Часто материалы



из этой книги используются в качестве учебных пособий для курсов во многих колледжах и университетах.

Веб-сайты:

- *RStudio*: на данном портале собраны разнообразные обучающие ресурсы по языку R. Для получения доступа к ним перейдите в меню на вкладку **Resources**, после чего вам будет предложен выбор бесплатных вебинаров, инструкций и книг. Также на этом сайте вы можете скачать последнюю версию рекомендованной среды для работы с языком R – *R Studio*. Адрес сайта: <https://rstudio.com>;
- *The R Graph Gallery*: на этом сайте представлены визуальные элементы, созданные при помощи R, с полными исходными кодами. Вы можете использовать эти наработки для собственных идей. Адрес сайта: www.r-graph-gallery.com;
- *R Bloggers*: прекрасный сайт для общения в сообществе по языку R. Адрес сайта: www.r-bloggers.com.

Обучение:

- *dplyr tutorial Part 1*: первая из двух частей обучающего видео по пакету *dplyr* от его автора Хэдли Уикхэма. Материал был записан в 2014 году, но до сих пор не утратил своей актуальности. Ссылка на первую часть видео: www.youtube.com/watch?v=8SGif63VW6E;
- *dplyr tutorial Part 2*: вторая часть обучающего видео от Хэдли Уикхэма: www.youtube.com/watch?v=Ue08LVuk790.

Ресурсы по Python

Книги:

- *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*: превосходная книга, освещающая применение принципов машинного обучения и искусственного интеллекта к вашим данным с помощью инструментов Python. Книга доступна для покупки на большинстве ресурсов;
- *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*: эта книга написана автором библиотеки *pandas* Уэсом Маккинни (Wes McKinney). *Pandas* по сей день является наиболее популярной библиотекой для работы с данными в Python. Книга доступна для покупки на большинстве ресурсов.

Видеоблоги, подкасты и курсы:

- *Data School*: это канал в YouTube от Кевина Маркхэма (Kevin Markham), на котором публикуется масса обучающих видео на тему анализа данных в Python при помощи библиотек *pandas*, *matplotlib*, *scikit-learn* и др. Кевин прекрасно справляется с задачей донесения до слушателей сложных тем довольно простым и понятным языком. Адрес канала: www.youtube.com/channel/UCnVzApLJE2ljPZSeQylSEyg;
- *Google's Python Class*: это достаточно старый, но все еще актуальный ресурс введения в Python от Google. В данном обучающем материале прекрасно представлено введение в структуры данных языка и другие

базовые концепции, лежащие в основе любого проекта. Также на сайте присутствует весьма информативный раздел, посвященный регулярным выражениям. Адрес двухдневного курса по Python: <https://developers.google.com/edu/python>;

- *Talk Python to Me*: очень информативный подкаст, в котором обсуждаются разные области применительно к Python, в том числе и анализ данных. Адрес подкаста: <https://talkpython.fm>.

Веб-сайты:

- *PEP 8*: одним из главных преимуществ Python в сравнении с другими языками программирования является его доступность и легкость чтения исходного кода. В отличие от большинства языков, Python жестко регламентирует стиль написания кода для облегчения его восприятия в дальнейшем. И именно стилю программирования на Python посвящен этот великолепный сайт, находящийся по адресу <https://pep8.org>.

Ресурсы по Power BI

Видеоблоги:

- *Guy in a Cube*: однозначно лучший видеоблог по Power BI. На этом канале YouTube освещаются все важнейшие аспекты Power BI, включая администрирование, моделирование данных и визуализацию. Ссылка на канал: www.youtube.com/channel/UCFp1vaKzpfvoGai0vE5VJ0w.

Веб-сайты:

- *SQLBI*: создатели этого сайта (Марко Руссо и Альберто Феррари) являются очень авторитетными авторами книг по языку запросов DAX применительно к Power BI. На их сайте содержится очень много статей, видеоуроков и обучающих материалов по DAX, а также полезные инструменты вроде *DAX Studio*;
- *Tabular Editor*: Tabular Editor – это инструмент с открытым кодом, который должен быть в арсенале любого серьезного разработчика Power BI. В скором времени он должен быть интегрирован в Power BI. Чтобы лучше изучить этот полезный инструмент, перейдите по ссылке, ведущей на его страницу: <https://tabulareditor.com>.

Книги:

- *The Definitive Guide to DAX* («Подробное руководство по DAX»): эта книга – настоящая библия для тех, кто хочет освоить язык запросов DAX, используемый в Power BI. Книга доступна для покупки в большинстве магазинов (<https://dmkpress.com/catalog/computer/data/978-5-97060-859-3/>);
- *M Is for Data Monkeys*: в этой книге дано введение в функциональный язык программирования M, используемый в инструменте Power Query. Авторы проделали хорошую работу, рассказав обо всех основных принципах и возможностях преобразования данных доступным языком. Эта книга может быть использована как основа для перехода к более сложным ресурсам по языку M. Книга доступна для покупки в большинстве магазинов.

Подкасты:

- *BIFOCAL*: прекрасный подкаст, позволяющий быть в курсе всего, что происходит в мире Power BI. Найти его можно на популярных платформах подкастинга.

Общие ресурсы

Книги:

- *Data Science for Business*: книга от Фостера Провоста (Foster Provost) и Тома Фосетта (Tom Fawcett), в которой представлены основные принципы науки о данных (data science) строгим языком программиста. Это очень популярная книга в сообществе бизнес-аналитики и может заинтересовать тех, кто ищет способы реализации методов науки о данных в области бизнес-приложений. Книга доступна для покупки во многих магазинах.

Веб-сайты:

- *Data Science Central*: это один из самых популярных сайтов, посвященных науке о данных. Адрес сайта: www.datasciencecentral.com;
- *Kaggle*: сайт изначально задумывался как место для соревнований в области программирования, но сейчас перерос в нечто большее. На этом сайте можно найти большое количество наборов данных, которые удобно использовать при работе с примерами по машинному обучению и искусственному интеллекту. Адрес сайта: www.kaggle.com;
- *ExcelTv*: Microsoft Excel был и в обозримом будущем останется одним из основных инструментов в аналитике данных. Авторы сайта ExcelTv подготовили большое количество обучающих материалов, которые помогут вам стать настоящим профессионалом в Excel. Адрес сайта: <https://excel.tv>.

Видеоблоги и обучающие сайты:

- *Excel on Fire*: Oz – создатель и ведущий этого видеоблога, – пожалуй, самый яркий и необычный преподаватель Excel и Power Query. За последние несколько лет он записал большое количество полезных видео по искусственному преобразованию данных при помощи инструмента Power Query. При этом сами ролики записаны на высшем уровне, а способность ведущего доносить сложные вещи простым и понятным языком просто поражает. Ссылка на канал: www.youtube.com/user/WalrusCandy/featured;
- *Regular Expression Tutorial* от Кори Шафера (Corey Schafer): отличный канал с вводными и продвинутыми уроками по использованию регулярных выражений. Поверьте, после просмотра нескольких видео на этом канале вы не только поймете, что означают все эти мудреные символы в регулярных выражениях, но и поразитесь, насколько полезными они могут быть при решении самых разных задач. Ссылка на видеоблог: www.youtube.com/watch?v=sa-TUpSx1JA.

Подкасты:

- *Analytics on Fire podcast*: это универсальный источник еженедельных мастер-классов по бизнес-аналитике от ведущих специалистов в дан-

ной области. Настраивайтесь на волну каждую неделю, и вы будете получать наслаждение от этого микса из образования и развлечения. Найти подкаст можно на популярных платформах подкастинга;

- *Data Skeptic*: в этом подкасте вопросы из области науки о данных повышенной сложности преподносятся в простой и легкой для усвоения форме с очень увлекательными примерами. Найти подкаст можно на популярных платформах подкастинга;
- *Freakonomics*: изучить технические аспекты науки о данных – это лишь полдела. Очень важно также развить в себе аналитические способности. И этот подкаст, безусловно, поможет вам в этом. Найти подкаст можно на популярных платформах подкастинга;
- *SQL Data Partners*: это очень познавательный и нескучный подкаст, освещающий вопросы, касающиеся *платформы данных Microsoft* (Microsoft Data Platform). Ведущие подкаста – признанные специалисты в этой области, но при этом они много шутят и смеются. Возможно, они будущие комики. :) Их подкаст можно найти на популярных платформах подкастинга;
- *Storytelling with Data*: этот прекрасный подкаст от Коул Нафлик (Cole Knaflie) посвящен различным техникам визуализации данных. Коул также является автором одноименной книги. Найти ее подкаст можно на популярных платформах подкастинга.

ОПИСАНИЕ ГЛАВ

- Глава 1 «Грамматика графиков». Вероятно, одним из главных преимуществ языка R над Python является его богатая оснащенность средствами визуализации данных. Ведущим пакетом R в области визуализации является *ggplot2*, базирующийся на концепции, известной как *грамматика графиков* (grammar of graphics). В данной главе мы изучим основы графического пакета *ggplot2* и рассмотрим его применение на практике совместно с Power BI.
- Глава 2 «Создание пользовательских визуализаций на R в Power BI при помощи *ggplot2*». Одно из явных преимуществ пакета *ggplot2* состоит в выразительности, с которой вы можете создавать свои собственные визуализации. В данной главе мы на нескольких примерах продемонстрируем идею выбора типа визуализации на языке R из всех доступных в Power BI при помощи пакета *ggplot2*.
- Глава 3 «Чтение файлов CSV». В этой главе мы рассмотрим концепции применительно к языкам R и Python, позволяющие динамически комбинировать файлы CSV, что с использованием инструмента Power Query было бы достаточно затруднительно.
- Глава 4 «Чтение данных из Microsoft Excel». Здесь мы научимся при помощи R и Python динамически сочетать рабочие листы из нескольких рабочих книг Excel, что в Power Query реализовать бывает непросто.
- Глава 5 «Чтение данных из SQL Server». Из данной главы вы узнаете, как посредством R и Python загружать данные из SQL Server в модель

данных Power BI. Одним из преимуществ такого метода загрузки информации – и мы покажем это в наших примерах – является возможность осуществления логирования.

- Глава 6 «Чтение в модель данных Power BI посредством API». В данной главе вы познакомитесь со способами извлечения данных в Power BI при помощи API с использованием языков R и Python. Посредством Power Query реализовать подобные методы бывает довольно сложно, а иногда просто невозможно.
- Глава 7 «Продвинутые строковые операции и распознавание шаблонов». Из этой главы вы узнаете, как при помощи регулярных выражений в R и Python осуществлять сложные манипуляции со строками. Регулярные выражения – очень мощный инструмент для работы с текстом, и если в R и Python он присутствует по умолчанию, то в Power Query пока нативно не реализован.
- Глава 8 «Вычисляемые столбцы с помощью R и Python». Здесь мы рассмотрим технику создания сложных математических выражений при помощи R и Python. Вы познакомитесь с основами написания математических формул и узнаете, как использовать готовые функции, скрывающие от вас истинную сложность вычислений. В качестве примера мы используем формулу гаверсинуса.
- Глава 9 «Применение методов машинного обучения и искусственного интеллекта к моделям данных Power BI». В девятой главе книги мы обсудим множество тем, включая использование в бизнес-аналитике методов машинного обучения и искусственного интеллекта. Начнем мы с примеров того, как использовать пользовательские модели машинного обучения, реализованные на R и Python, к модели данных Power BI. Затем посмотрим, как можно улучшить модели данных Power BI с применением службы *Microsoft Cognitive Services* без необходимости оформления дорогостоящей подписки на Power BI Premium. При этом мы не будем ограничиваться лишь инструментами от Microsoft и покажем, как можно воспользоваться службой *IBM Watson Natural Language Understanding* для выполнения специфического анализа текста, недоступного в рамках *Microsoft Cognitive Services*.
- Глава 10 «Создание моделей анализа данных и скриптов для обработки информации». В заключительной главе мы посмотрим, как можно воспользоваться языками программирования R и Python в корпоративных решениях, внедренных в Power BI. Показанные методы ориентированы на бесплатные решения, доступные пользователям, у которых уже установлена локальная версия SQL Server версии 2017 и выше.

Теперь, когда все подготовительные мероприятия завершены, вы можете приступить к чтению книги. Инструмент Power BI славится своими богатыми возможностями в области визуализации данных. И начнем мы с того, как можно значительно расширить их при помощи языка R и пакета *ggplot2*.

Часть I



.....

СОЗДАНИЕ ПОЛЬЗОВАТЕЛЬСКОЙ ВИЗУАЛИЗАЦИИ ПРИ ПОМОЩИ R



Глава 1

.....

Грамматика графиков

Визуализация данных во все времена была одной из важнейших составляющих статистики как науки. Именно посредством визуализаций специалисты в этой области могут делиться с окружающими своими изысканиями и при этом не нагружать их сухими цифрами, а говорить на понятном им языке. С учетом того, что язык программирования R был создан аналитиками и для аналитиков, участники сообщества приложили немало усилий, чтобы разработчики могли создавать богатые визуализации, способные помочь им доходчиво рассказывать свои истории о данных.

Наиболее популярным пакетом визуализации данных в R, безусловно, является *ggplot2*. Он был разработан Хэдли Уикхэмом и базируется на концепции, получившей название *многослойная грамматика графиков* (layered grammar of graphics). В рамках этой концепции любой график определяется следующими характеристиками:

- набором данных по умолчанию с привязками переменных к *эстетикам* (aesthetics);
- одним или более слоями, состоящими из геометрического объекта, статистической трансформации, настройки позиции, а также в качестве необязательных элементов из набора данных и привязки эстетики;
- одной *шкалой* (scale) для каждой привязки эстетики;
- системой координат;
- параметрами фасетирования (faceting).

Это определение в полной мере описывает принципы многослойной грамматики графиков. Скоро вы узнаете, как использовать описанную выше концепцию для построения прекрасных визуализаций в Power BI при помощи пакета *ggplot2*.

Несмотря на то что Power BI предоставляет возможности создания визуальных элементов с использованием самых разных библиотек и пакетов Python и R, в данной книге мы главным образом сосредоточимся на применении пакета *ggplot2* и нескольких вспомогательных пакетов. Причина в том, что пакет *ggplot2* находится вне конкуренции в сравнении с другими инструментами из арсенала R и Python. Принципы многослойной грамматики графиков, на которых базируется пакет *ggplot2*, позволяют значительно сократить время от рождения идеи определенной визуализации данных до ее воплощения на практике. Этот пакет обладает богатейшим набором возможностей, так что подавляющее большинство ваших идей в области визуализации может быть легко реализовано при помощи него.

ПОШАГОВОЕ СОЗДАНИЕ ВИЗУАЛИЗАЦИИ В POWER BI ПРИ ПОМОЩИ R



В данном разделе будет подробно рассказано о процессе создания пользовательского элемента визуализации в Power BI посредством языка R. Следующие действия вам необходимо будет выполнить вне зависимости от того, какой тип визуализации вы собираетесь использовать.

Шаг 1. Настройте Power BI

Во вводной части книги мы уже говорили о том, как настроить Power BI для работы с R. Убедитесь, что вы выполнили все инструкции и рекомендации по подготовке к работе.

Шаг 2. Перенесите визуальный элемент R в рабочую область Power BI

На вашей панели выбора элементов визуализации справа есть кнопка с латинской буквой «R», показанная на рис. 1.1, при наведении на которую мышью появляется подсказка **Визуальный элемент скрипта R** (R custom visual). Для создания визуализации при помощи этого элемента необходимо нажать на иконку R, в результате чего будет создана заготовка в рабочей области.

Элемент визуализации R предустановлен в Power BI по умолчанию. В рабочей области вы можете изменить его размер и позицию, как и у любого другого визуального элемента в Power BI.

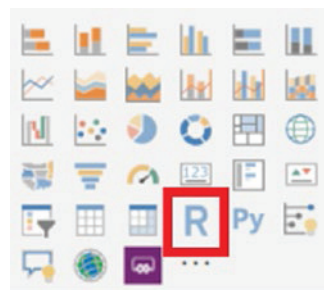


Рис. 1.1 ❖ Иконка создания визуального элемента R в Power BI

Шаг 3. Определитесь с набором данных

Как и с любым элементом визуализации, в данном случае вам также необходимо определить источник данных. Вы можете собрать набор данных путем переноса полей из вашей модели данных и необязательных мер на панель **Значения** (Values). Полученный в результате набор данных будет доступен в R в виде объекта с именем *data frame*. Вы можете воспринимать этот объект как обычную таблицу Excel с расширенными возможностями для анализа данных.

Одной из особенностей такого объекта, о которой не стоит забывать при создании визуальных элементов R, является обязательная уникальность



строк. При создании объекта *data frame* R автоматически будет проверять его на уникальность строк, и вам необходимо позаботиться о том, чтобы это условие выполнялось в исходном наборе данных.

Шаг 4. Спроектируйте визуальный элемент в среде разработки R

Теперь, когда вы определились с источником данных для своей визуализации, можно приступить к ее проектированию в вашей любимой среде разработки. Когда вы настраивали Power BI для работы со скриптами R, на одном из шагов вы, если помните, указывали предпочтительную среду разработки. Это делалось для того, чтобы вы могли создавать свой элемент визуализации в привычном для вас инструменте вроде *R Studio* вместо того, чтобы пользоваться встроенным в Power BI редактором скриптов R (R script editor).

В конечном счете код для вашего визуального элемента должен быть вставлен в редактор скриптов R, показанный на рис. 1.2. Этот редактор открывается в нижней части рабочей области в Power BI при нажатии на кнопку создания визуального элемента скрипта R.

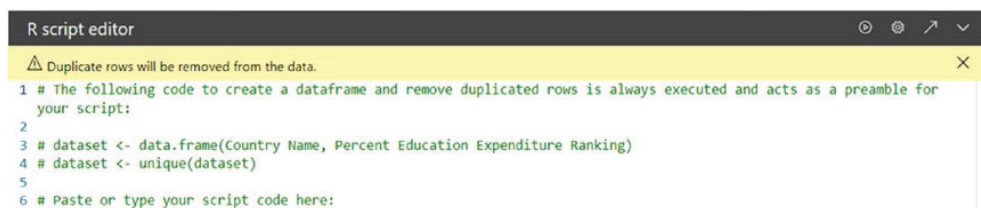


Рис. 1.2 ❖ Редактор скриптов R в Power BI

Как видите, встроенный редактор – не идеальное место для написания скриптов на языке R. Здесь нет *автоматического дополнения ввода* (Intel-lisense), нет консоли, а пространство для кода весьма и весьма ограничено. К счастью, Power BI позволяет вам портировать набор данных, который вы определили для своего визуального элемента, в вашу среду разработки. Рекомендованной средой является *R Studio*, и в данной книге мы будем обсуждать работу именно в ней.

Чтобы портировать набор данных для вашего пользовательского визуального элемента в *R Studio*, необходимо выбрать элемент в рабочей области и нажать в заголовке редактора скриптов на стрелку, направленную вверх и направо. Это приведет к запуску оболочки *R Studio* с базовым шаблоном скрипта, включающим в себя код с созданием переменной для *датафрейма* (data frame) с именем *dataset* на основе данных, переданных элементу визуализации в Power BI. Также в оболочку будет перенесен весь код, ассоциированный с вашим визуальным элементом. В общем, в Microsoft решили

не изобретать велосипед, а дать нам возможность разрабатывать свои элементы там, где нам удобно. Лично я рекомендую вам проектировать все свои разработки на языке R именно в *R Studio*, а не пользоваться для этой цели встроенным редактором в Power BI.

Шаг 5. Используйте следующий шаблон для разработки элемента на R

Представленный ниже шаблон очень удобно использовать при разработке визуального элемента скрипта R в Power BI:

```
if (<"data test">) {
  #<"Code for R Visual">
} else {
  plot.new()
  title(main = "<predefined message>")
}
```

В первой строке шаблона выполняется проверка набора данных на соответствие всем требованиям визуализации на языке R. Если все в порядке, визуализация создается успешно, в противном случае будет сгенерирован пустой элемент с заранее определенным текстом. За создание пустого элемента отвечает следующий фрагмент кода:

```
plot.new()
title(main = " <predefined message> ")
```

В первой строке создается пустая диаграмма, а во второй к ней добавляется заголовок с заранее определенным сообщением, которое будет показано пользователю.

Шаг 6. Добавьте скрипту функциональности

После написания скрипта для вашего элемента визуализации в сторонней среде разработки перенесите его в редактор скриптов R в Power BI – целиком, за исключением строки с созданием переменной набора данных с именем *dataset*. Этот набор будет автоматически создан в Power BI. Заметьте, что созданный вами элемент визуализации на языке R будет обладать полной интерактивностью и реагировать на изменение фильтров и других визуальных элементов. В следующей главе мы рассмотрим несколько примеров. Единственное, чего вы лишитесь при создании элемента скрипта, – это *двухнаправленной фильтрации* (bidirectional filtering). Таким образом, изменение внешних элементов визуализации в Power BI будет распространять свое действие на ваш элемент R, но фильтровать другие элементы непосредственно из созданного вами не получится.

РЕКОМЕНДОВАННЫЕ ШАГИ ПО СОЗДАНИЮ ВИЗУАЛЬНОГО ЭЛЕМЕНТА НА R ПРИ ПОМОЩИ GGPLOT2

В предыдущем разделе мы рассмотрели базовый шаблон, с которого стоит начинать разработку собственного элемента визуализации на языке R для Power BI. По сути, он состоит из проверки набора данных на удовлетворение всем необходимым требованиям и перехвата ошибок. В данной секции мы сосредоточимся на фрагменте кода для создания самого элемента, который в представленном листинге обозначен как #<»Code for R Visual»>.

Далее мы рассмотрим, как можно изменить базовый шаблон для построения диаграммы, показанной на рис. 1.3.

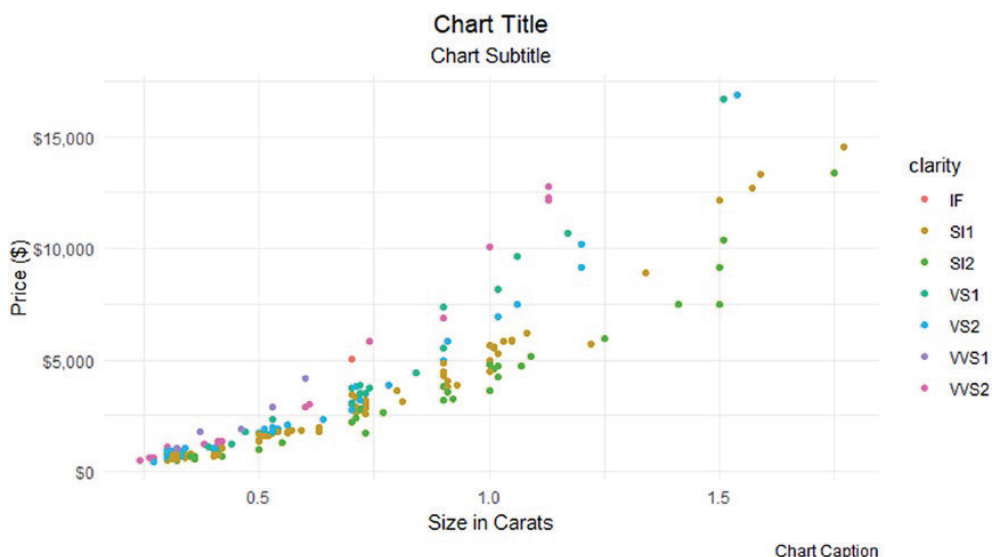


Рис. 1.3 ❖ Визуальный элемент для демонстрации в Power BI

В качестве источника для этой диаграммы используется набор данных об алмазах в части зависимости цены от веса камней в каратах, при этом анализ проводится только по алмазам с цветовой классификацией D. Это довольно подходящий пример для нас, поскольку он позволяет пройти по порядку все шаги при создании элемента визуализации скрипта R. Давайте начнем с самого начала.

Шаг 1. Импортируйте нужные для скрипта пакеты

Для данного примера мы импортируем два пакета, как показано в представленном ниже листинге:

```
library(tidyverse)
library(scales)
```

Мы будем использовать два пакета *ggplot2* и *dplyr* из набора пакетов *tidyverse*.

Примечание. Набор пакетов *tidyverse* объединяет в себе несколько библиотек, значительно облегчающих процесс анализа данных в языке R. Указание на весь набор пакетов (или метапакет) *tidyverse* позволяет загрузить все нужные вам ключевые пакеты одной строкой, вместо того чтобы подгружать каждый пакет по отдельности.

Пакет *ggplot2* мы будем использовать для построения визуализаций, тогда как пакет *dplyr* поможет нам при обработке исходных данных. Загруженный также пакет *scales* поможет нам с форматированием числовых значений.

Шаг 2. Выполните необходимое преобразование исходных данных

Зачастую, перед тем как «скормить» исходные данные пакету *ggplot2*, требуется привести их в удобный для анализа вид. Один из примеров такой трансформации данных приведен ниже:

```
plot.data <-
  diamonds %>%
  filter(color == "D") %>%
  sample_n(200)
```

Набор данных, переданный на вход элементу визуализации R, помимо полезных показателей содержит информацию, не предназначенную для вывода. Так что мы воспользовались пакетом *dplyr*, чтобы очистить исходные данные от лишнего наполнения. Мы знаем, что наша визуализация должна показывать информацию только по алмазам цветовой классификации D. Таким образом, нам необходимо отфильтровать оригинальный датафрейм *diamonds*, чтобы он включал только нужный нам цветовой диапазон. В листинге выше мы делаем именно это в третьей строке кода.

После фильтрации в наборе данных остается информация только об алмазах нужной нам цветовой классификации. Всего о таких алмазах насчитывается 6775 записей, что довольно много для нашей *диаграммы рассеяния* (scatter plot). Вывод на график чересчур большого количества исследований может привести к тому, что визуализация станет совершенно нечитаемой. Чтобы избежать этого, можно случайным образом выбрать из всей информации какое-то ее подмножество и отобразить только его. Подмножество данных будет обладать тем же распределением, что и исходный набор, что позволит отобразить на графике правильные тенденции и при этом сделать его легким для восприятия. В представленном примере мы воспользовались функцией *sample_n()* из пакета *dplyr*, оставив в итоговом наборе всего 200 записей, которые вполне приемлемо смотрятся на диаграмме рассеяния.

Шаг 3. Создайте визуализацию при помощи функции `ggplot()`

Создание диаграммы при использовании пакета `ggplot2` осуществляется путем вызова функции `ggplot()`. На вход функции подаются параметры, которые будут доступны всем без исключения слоям нашей визуализации. В данном случае мы передали в качестве параметра `data` содержимое датафрейма `plot.data`, а значения переменных, хранящих информацию о весе алмазов и их цене, передали аргументам `x` и `y` функции `aes()`, которая также поступила на вход функции `ggplot()`, как показано ниже:

```
ggplot(data = plot.data, aes(x = carat, y = price))
```

Очень важно понимать, что собой представляет функция `aes()` и как ее следует использовать. Функция `aes()` описывает правила привязки ваших данных к визуальным атрибутам, или *эстетикам* (aesthetics). Среди распространенных эстетик можно выделить следующие:

- координаты `x` и `y` на выбранном виде диаграммы;
- цвет выбранных вами геометрических фигур;
- размер геометрических фигур;
- цветовое заполнение фигур на диаграмме.

В нашем простом примере функция `aes()` используется для описания того, какие показатели должны быть отмечены на горизонтальной и вертикальной осях. При этом функция `aes()` может быть объявлена как в функции `ggplot()`, как в нашем случае, так и в функции `geom()`, о которой вы узнаете на следующем шаге. Разница в том, что если функция `aes()` объявлена внутри функции `ggplot()`, ее действие будет распространяться на все слои диаграммы, а если указана в функции `geom()`, действие будет ограничиваться конкретным слоем. В следующей главе мы рассмотрим множество примеров применения функции `aes()` на практике.

Шаг 4. Добавьте нужные геометрии

Пример добавления геометрии (`geom`) к визуализации показан в следующем фрагменте кода:

```
ggplot(plot.data, aes(x = carat, y = price)) +  
  geom_point()
```

При инициализации функции `ggplot()` на предыдущем шаге наша визуализация не содержала ни одного слоя, а значит, и визуализировать ничего не могла. Механизм добавления слоев в функции `ggplot()` реализуется через указание соответствующих функций `geom`. Геометрия, которую мы используем для диаграммы рассеяния, добавляется в функцию `ggplot()` посредством вы-

зова функции `geom_point()`. В представленном выше фрагменте кода функция `geom_point()` наследуется от предшествующей ей функции `ggplot()`, поэтому источник данных и координаты x и y , обычно обязательные для указания, можно явно не определять. Результирующий график показан на рис. 1.4.

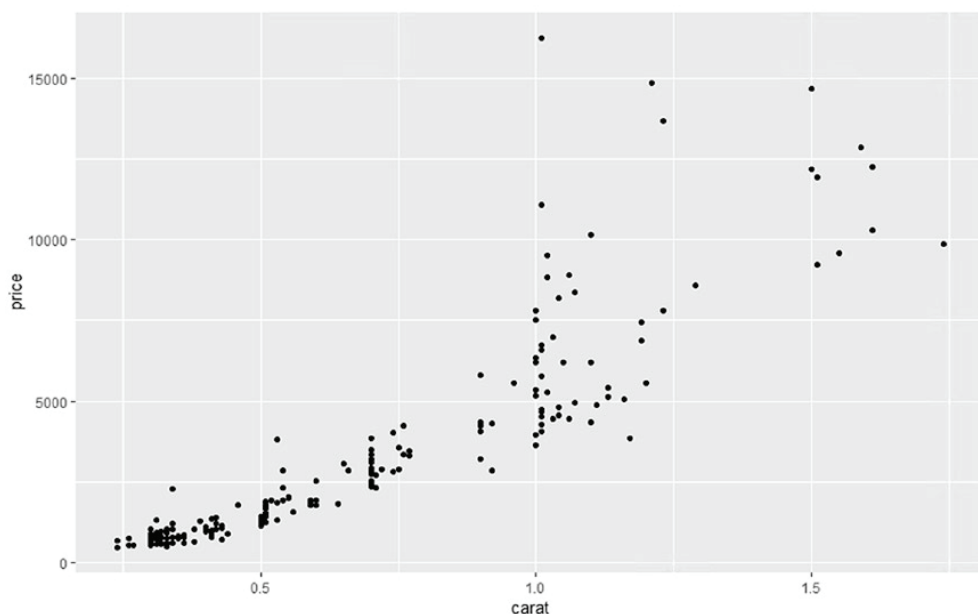


Рис. 1.4 ❖ Базовая диаграмма рассеяния

Теперь нам необходимо добавить еще один слой детализации, а именно мы добавим цветовую дифференциацию по *чистоте* (*clarity*) алмаза. Для этого необходимо присвоить эстетике *color* значение переменной *clarity* в функции `geom_point()`, как показано ниже:

```
ggplot(plot.data, aes(x = carat, y = price)) +  
  geom_point(aes(color = clarity))
```

Как видите, мы установили эстетику *color* только в созданной нами геометрии, поскольку она будет использоваться лишь в этом слое.

Примечание. Очень важно помнить, что, когда вы устанавливаете источники данных или эстетики в функции `ggplot()`, они будут распространяться на все слои в вашей визуализации, а когда делаете это в конкретной геометрии, их область доступа будет ограничена только ассоциированным с этой геометрией слоем.

Результирующая диаграмма с параметром *clarity* для каждой точки показана на рис. 1.5.

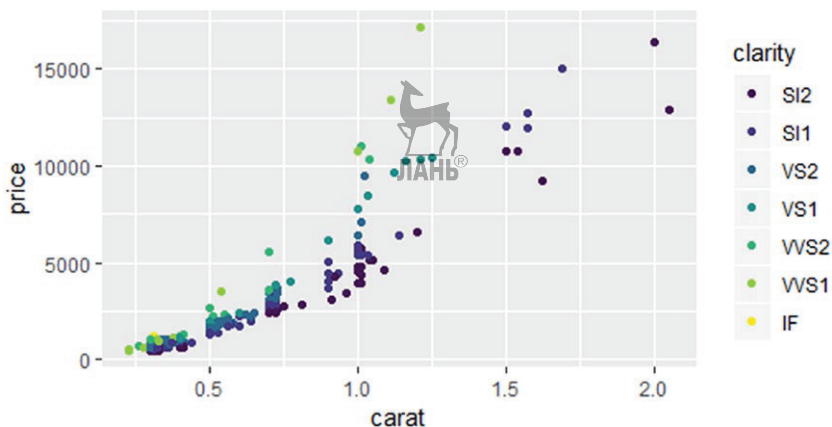


Рис. 1.5 ❖ Базовая диаграмма рассеяния с детализацией

Шаг 5. Определите заголовки, подзаголовки и подписи

Используя следующий код, можно добавить на визуализацию заголовки:

```
ggplot(plot.data, aes(x = carat, y = price)) +
  geom_point(aes(color = clarity))+
  labs(
    title = "Chart Title",
    subtitle = "Chart Subtitle",
    caption = "Chart Caption"
  )
```

Пакет *ggplot2* позволяет очень легко добавлять на диаграммы заголовки, подзаголовки и подписи при помощи функции *labs()*. Название этой функции сокращено от «labels» (метки). При этом титульные надписи на вашей визуализации могут быть как статическими строками, так и динамическими. В следующей главе мы рассмотрим примеры использования динамических заголовков. На рис. 1.6 показано, как будет выглядеть наша диаграмма с заголовками.

Шаг 6. Приведите в порядок оси

Зачастую имена, присвоенные осям *x* и *y* по умолчанию при построении диаграммы, нормально подходят и не требуют каких-то изменений. Но если вам необходимо внести правки в названия осей, придется воспользоваться функциями семейства *scale*.

Давайте вспомним весь код, который мы написали на данный момент:

```
library(tidyverse)
library(scales)
```



```
plot.data <-
  diamonds %>%
    filter(color == "D") %>%
    sample_n(200)

ggplot(plot.data, aes(carat, price)) +
  geom_point(aes(color = clarity)) +
  labs(
    title = "Chart Title",
    subtitle = "Chart Subtitle",
    caption = "Chart Caption"
  )
)
```

Запуск этого фрагмента кода приведет к построению диаграммы рассеяния, показанной на рис. 1.6. Как видите, на ней осталось несколько надписей, вставленных по умолчанию, которые нам хотелось бы изменить. Список наших будущих изменений:

- название оси x;
- название оси y;
- способ форматирования чисел на оси y.

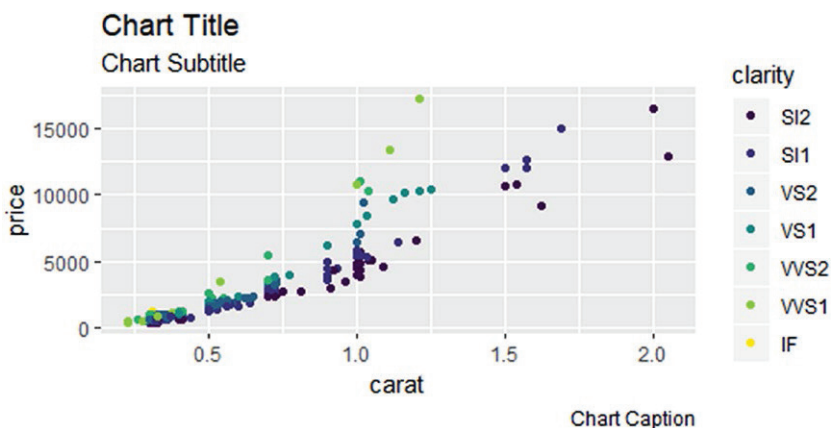


Рис. 1.6 ❖ Визуализация с установленными заголовками и подписями

Эти изменения можно внести, воспользовавшись функциями *scale_x_continuous()* и *scale_y_continuous()*, как показано ниже:

```
library(tidyverse)
library(scales)

plot.data <-
  diamonds %>%
    filter(color == "D") %>%
    sample_n(200)

ggplot(plot.data, aes(x = carat, y = price)) +
  geom_point(aes(color = clarity)) +
```



```
labs(
  title = "Chart Title",
  subtitle = "Chart Subtitle",
  caption = "Chart Caption"
) +
scale_x_continuous(name = "Size in Carats") +
scale_y_continuous(
  name = "Price ($)",
  labels = dollar_format()
)
```



Получившаяся в результате запуска этого кода диаграмма показана на рис. 1.7.

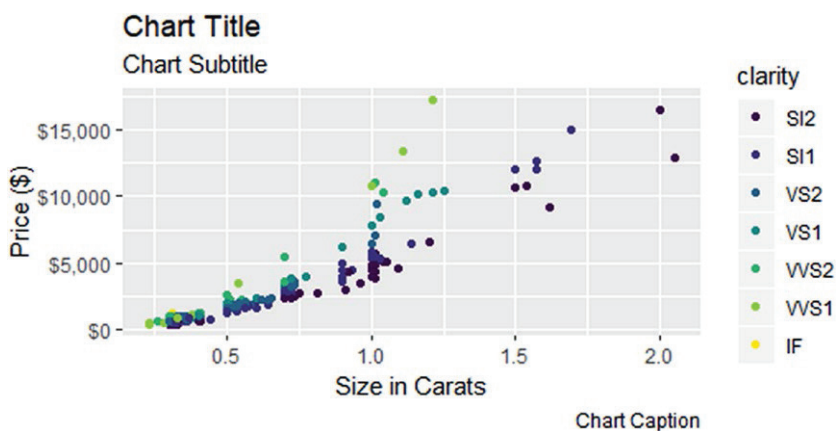


Рис. 1.7 ❖ Диаграмма с настроенными осями

Как видите, при помощи функций `scale_x_continuous()` и `scale_y_continuous()` можно достаточно быстро внести необходимые нам корректировки.

Примечание. Заметьте, что вы добавили шкалы на график так же точно, как до этого добавляли слои. Здесь важно понять, что добавленные шкалы заменяют собой шкалы по умолчанию, но не добавляют к вашей диаграмме слои.

Поскольку оси x и y базируются на *непрерывных данных* (continuous data), для них будет использоваться именно непрерывная шкала. Единственное, что нам необходимо изменить в отношении оси x , – это ее подпись, что, как видите, легко делается путем передачи аргумента `name`. Для оси y мы изменим не только подпись, но и способ форматирования числовых меток на оси. Подпись также меняется при помощи аргумента `name`, а для меток предусмотрен свой аргумент с названием `labels`. Мы передали этому аргументу функцию `dollar_format()` из пакета `scales`, что привело к форматированию меток на оси в виде долларов. Функция `dollar_format()` наследует параметр `u` от функции `ggplot()`, которая была определена в начале кода, так что явно объявлять его не нужно.

Шаг 7. Примените тему при необходимости

В пакете *ggplot2* есть предустановленные *темы* (themes), отвечающие за оформление диаграмм в той части, где нет данных. Сюда относится и расположение подписей и заголовков на графике, и цвет фона, и размер шрифтов для заголовков, и многое другое. Применить одну из предустановленных тем к вашему визуальному элементу довольно просто. Темы добавляются так же точно, как устанавливаются слои и модифицируются шкалы. Вы просто ставите знак + (плюс), следом за которым указываете название функции, символизирующей выбранную вами тему, как показано во фрагменте кода ниже:

```
ggplot(plot.data, aes(x = carat, y = price)) +
  geom_point(aes(color = clarity)) +
  labs(
    title = "Chart Title",
    subtitle = "Chart Subtitle",
    caption = "Chart Caption"
  ) +
  scale_x_continuous(name = "Size in Carats") +
  scale_y_continuous(
    name = "Price ($)",
    labels = dollar_format()
  ) +
  theme_minimal()
```

Результат показан на рис. 1.8.

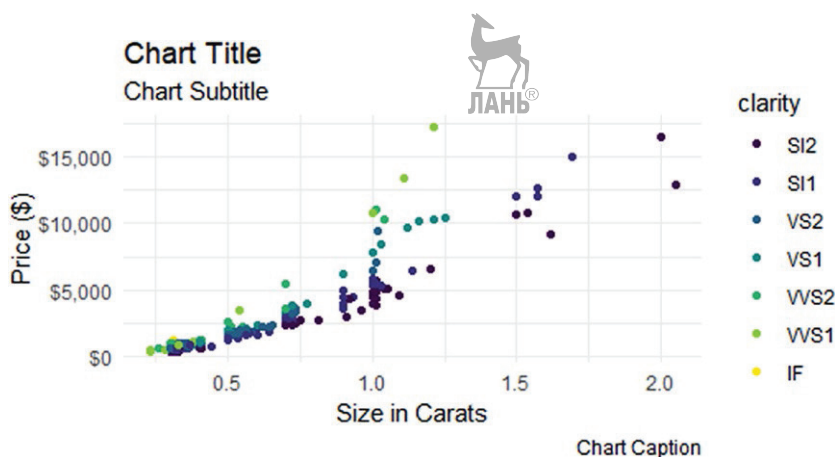


Рис. 1.8 ❖ Установка темы для визуализации

В данном примере мы выбрали предустановленную тему *theme_minimal()*. Считается хорошим тоном снабжать графики дополнительными данными только в случае особой необходимости. Белый фон под диаграммой также выглядит довольно приятно. Тема *theme_minimal()*, как ясно из названия, базируется на основных принципах минимализма. Подробно о темах, предустановленных в пакете *ggplot2*, мы поговорим далее в этой главе.

Шаг 8. Используйте функцию `theme()` для настройки оформления

Если вы остались не полностью удовлетворены примененной к визуализации предустановленной темой, то можете провести ее тонкую настройку при помощи специальной функции `theme()`. В следующем фрагменте кода мы вызовем эту функцию с целью форматирования заголовков:

```
ggplot(plot.data, aes(carat, price)) +
  geom_point(aes(color = clarity)) +
  labs(
    title = "Chart Title",
    subtitle = "Chart Subtitle",
    caption = "Chart Caption"
  ) +
  scale_x_continuous(name = "Size in Carats") +
  scale_y_continuous(
    name = "Price ($)",
    labels = dollar_format()
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5),
    plot.subtitle = element_text(hjust = 0.5),
    plot.caption = element_text(hjust = 1)
  )
```

Результат запуска этого кода показан на рис. 1.9.

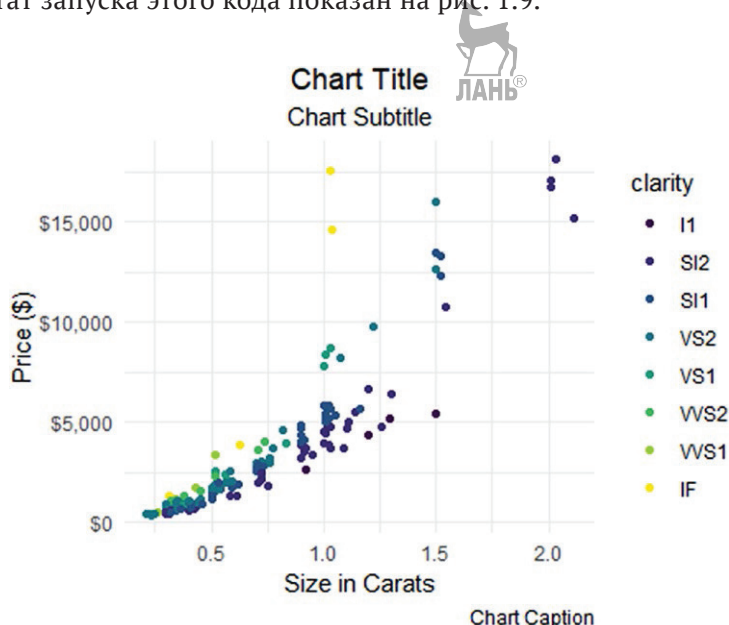


Рис. 1.9 ❖ Смещение заголовков и подписей с помощью функции `theme()`

Функция *theme()* открывает вам доступ к форматированию более чем 80 различных элементов на графике. Для большинства этих элементов существуют функции семейства *element*, предназначенные специально для выполнения подобных изменений.

В представленном выше фрагменте кода мы изменили положение заголовка диаграммы, подзаголовка и подписи. Как видите, элементу, ассоциированному с заголовком графика, соответствует объект *plot.title*, к подзаголовку мы обращаемся как к *plot.subtitle*, а к нижней подписи – как к *plot.caption*. При этом изменение всех трех элементов выполняется при помощи функции *element_text()*. Заголовок и подзаголовок графика были центрированы путем передачи аргументу *hjust* функции *element_text()*, отвечающему за горизонтальное позиционирование элемента, значения 0.5. Этот аргумент может принимать значения в диапазоне от 0 до 1, где 0 означает расположение элемента по левому краю, 0.5 – по центру, а 1 – по правому краю.

Ниже представлены функции семейства *element* со всеми своими аргументами:

```
element_blank()
```

```
element_rect(fill = NULL, colour = NULL, size = NULL, linetype = NULL, color = NULL,
inherit.blank = FALSE)
```

```
element_line(colour = NULL, size = NULL, linetype = NULL, lineend = NULL, color = NULL,
arrow = NULL, inherit.blank = FALSE)
```

```
element_text(family = NULL, face = NULL, colour = NULL, size = NULL, hjust = NULL, vjust =
NULL, angle = NULL, lineheight = NULL, color = NULL, margin = NULL, debug = NULL, inherit.
blank = FALSE)
```

По адресу <https://ggplot2.tidyverse.org/reference/element.html> можно найти полную документацию по функции *theme()*.

Дополнительный шаг: задайте цвета точек на диаграмме рассеяния

Иногда вам будет необходимо, чтобы все точки на диаграмме рассеяния были одного цвета. Давайте посмотрим, что будет, если присвоить эстетике *color* конкретный цвет, а не привязывать ее к определенному полю в наборе данных. Во фрагменте кода ниже мы присвоим значение *blue* этой эстетике в функции *aes()*:

```
library(tidyverse)
library(scales)

plot.data <-
  diamonds %>%
  filter(color == "D") %>%
  sample_n(200)
```



```
ggplot(plot.data, aes(carat, price)) +
  geom_point(aes(color = "blue")) +
  labs(
    title = "Chart Title",
    subtitle = "Chart Subtitle",
    caption = "Chart Caption"
  ) +
  scale_x_continuous(name = "Size in Carats") +
  scale_y_continuous(
    name = "Price ($)",
    labels = dollar_format()
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5),
    plot.subtitle = element_text(hjust = 0.5),
    plot.caption = element_text(hjust = 1)
  )
```



Результирующая диаграмма показана на рис. 1.10.

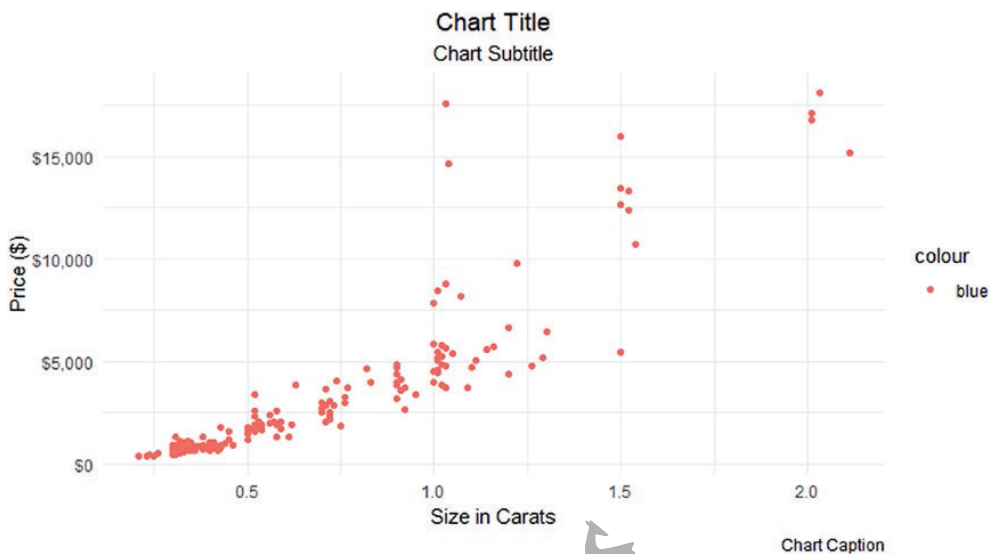


Рис. 1.10 ❖ Установка эстетики *color* в функции *aes()*



Заметьте, что мы не добились желаемого результата. Точки на нашей диаграмме рассеяния получились не синего цвета, как мы планировали, а красного. Причина в том, что мы попытались присвоить цвет точкам внутри функции *aes()*. Если вы хотите присвоить цвету конкретное значение, делать это необходимо за пределами функции *aes()*, как показано в исправленном листинге ниже:

```
library(tidyverse)
library(scales)

plot.data <-
  diamonds %>%
    filter(color == "D") %>%
    sample_n(200)

ggplot(plot.data, aes(carat, price)) +
  geom_point(color = "blue") +
  labs(
    title = "Chart Title",
    subtitle = "Chart Subtitle",
    caption = "Chart Caption"
  ) +
  scale_x_continuous(name = "Size in Carats") +
  scale_y_continuous(
    name = "Price ($)",
    labels = dollar_format()
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5),
    plot.subtitle = element_text(hjust = 0.5),
    plot.caption = element_text(hjust = 1)
  )
)
```



Теперь мы получим ожидаемый результат с синими точками по диаграмме рассеяния, как видно на рис. 1.11.

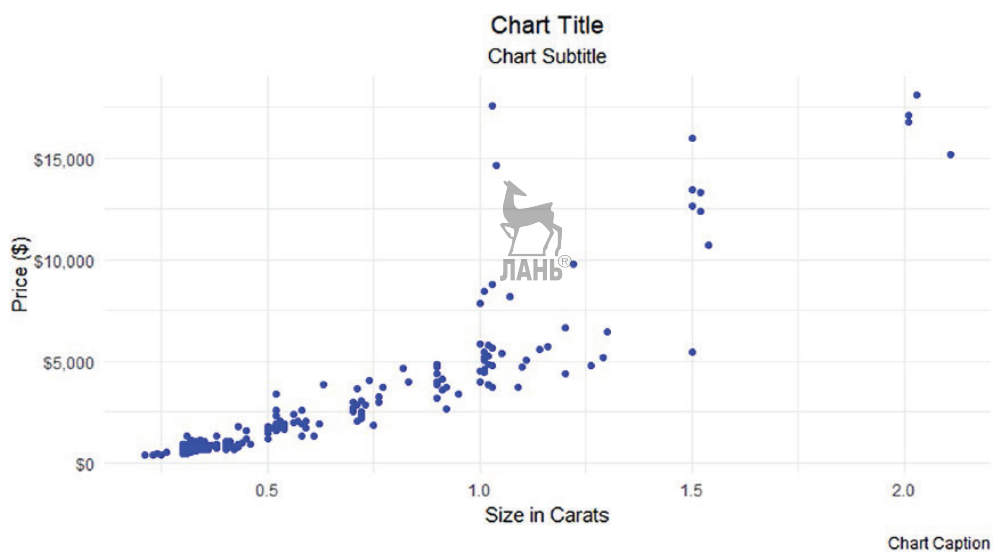


Рис. 1.11 ❖ Определение цвета точек за пределами функции `aes()`

Важность оперирования «чистыми» данными

Разработчики, знакомые с языком запросов DAX, понимают всю важность того, чтобы модель данных была приведена к канонической схеме «звезда». При выполнении этого условия даже сложные меры на DAX будет писать очень легко и просто. В то же время простые меры без должной организации данных могут оказаться очень сложными для написания.

Подобный феномен встречается и применительно к построению визуализаций и выполнению анализа данных в R. Когда данные приведены к «чистому» (tidy) формату, как его назвал Хэдли Уикхэм, анализ и визуализация данных не составляют большого труда.

Примечание. «Чистым» называется набор данных, в котором каждой строке соответствует ровно одно наблюдение, а каждый столбец представляет собой одну переменную. Это утверждение перекликается с приведением структуры базы данных к третьей нормальной форме.

Если по той или иной причине данные, которые вы передаете визуальному элементу R в Power BI, оказались не в «чистом» виде, ничего страшного. Вы всегда можете воспользоваться пакетами *dplyr* и *tidyr* в скриптах своего элемента визуализации для приведения данных к нужному вам формату.

Концепция приведения исходной информации к «чистому» формату занимает важное место при анализе и визуализации данных, и это касается не только языка R. Подробное освещение этой темы выходит за рамки данной книги. К счастью, Хэдли Уикхэм написал прекрасную инструкцию по «очищению» данных, которую можно загрузить по адресу <https://vita.had.co.nz/papers/tidy-data.pdf>.

Популярные геометрии

Как мы уже раньше говорили, пакет *ggplot2* основан на теории многослойной грамматики графиков. Слои добавляются в визуальные элементы посредством вызова функций семейства *geom*. Эта особенность позволяет строить поистине прекрасные визуализации с минимумом кода, довольно простого для понимания. Выбранный подход способствует гораздо более легкому воплощению своих идей при помощи визуализации в сравнении с другими языками программирования. В пакете *ggplot2* вашему вниманию представлено великое множество геометрий, и в данной главе мы рассмотрим и приведем примеры наиболее популярных из них.

- `geom_bar()`. Функция `geom_bar()` используется для построения привычных всем нам вертикальных и горизонтальных *столбчатых диаграмм* (bar chart). Посмотрим на примере, приведенном ниже, как можно создать простую столбчатую диаграмму при помощи функции `geom_bar()`:

```
library(tidyverse)

plot.data <-
  data.frame(Titanic) %>%
  group_by(Class) %>%
  summarize(`Total Freq` = sum(Freq))

ggplot(plot.data, aes(x = Class, y = `Total Freq`)) +
  geom_bar(stat = "identity") +
  labs(title = "geom_bar") +
  theme_minimal()
```

Визуализация, полученная в результате запуска этого скрипта, показана на рис. 1.12.

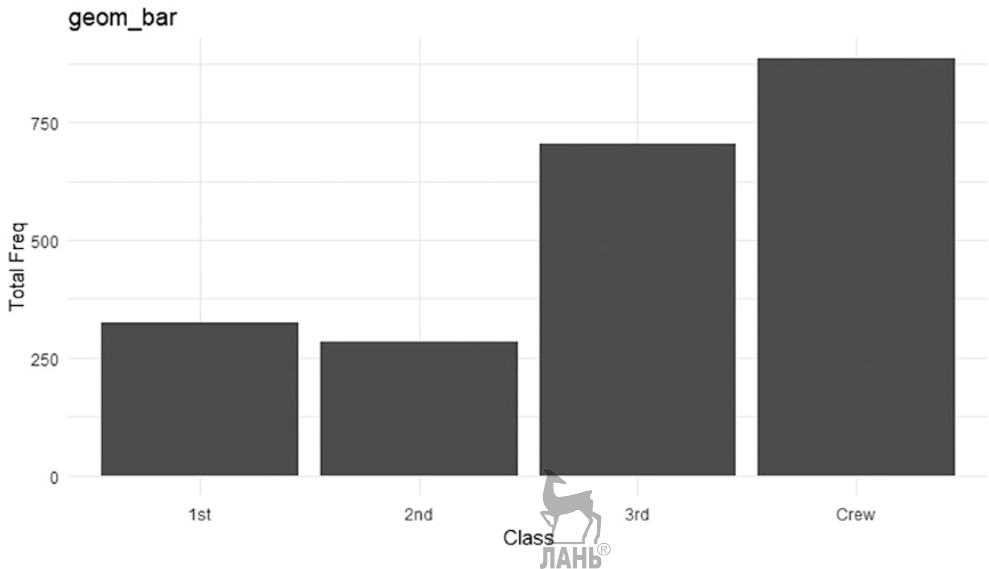


Рис. 1.12 ❖ Геометрия `geom_bar()`

В первой строке кода происходит загрузка набора пакетов *tidyverse*. Далее выполняется подготовка набора данных для визуализации при помощи функции `geom_bar()`. Техника преобразования данных, используемая в этом примере, будет подробно обсуждаться далее в книге.

После этого следует блок создания визуализации. По сути, первых двух строк этого блока достаточно, чтобы построить столбчатую диаграмму. Последние две строки блока призваны навести красоту в визуализации. В них на диаграмму выводится нужный нам заголовок и меняется тема на минимальную.

Обратите внимание, что функции `geom_bar()` достаточно передать единственный аргумент *stat* для корректного создания геометрии. Остальные аргументы (набор данных, а также оси *x* и *y*) в данном случае наследуются от функции `ggplot()`. Присвоение параметру *stat* значения

«identity» говорит пакету *ggplot2* о том, что высоты столбиков должны быть ассоциированы со значениями полей, связанных с эстетикой у в функции *aes()*. По умолчанию используется количество элементов в каждой группе. Подробнее об аргументе *stat* и его значениях мы поговорим в следующих примерах.

- *geom_histogram()*. Функция *geom_histogram()*, как ясно из названия, используется для построения гистограмм. Гистограммы представляют собой диаграммы распределения значений. Взгляните на код, используемый для создания гистограммы с помощью функции *geom_histogram()*:

```
library(tidyverse)

set.seed(50)
probs <- c(0.0033, 0.0033, 0.0033, 0.0033, 0.0033, 0.0033, 0.0033, 0.0033, 0.0033,
0.0033, 0.0033, 0.0033, 0.0033, 0.0033, 0.0033, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01,
0.01, 0.01, 0.01, 0.01, 0.01, 0.015, 0.015, 0.015, 0.015, 0.015, 0.015, 0.015, 0.015,
0.015, 0.015, 0.0133, 0.0133, 0.0133, 0.0133, 0.0133, 0.0133, 0.0133, 0.0133, 0.0133,
0.0133, 0.0133, 0.0133, 0.0133, 0.0133, 0.0133, 0.0133, 0.0133, 0.0133, 0.0133, 0.0133,
0.0133, 0.0133, 0.0133, 0.0133, 0.0133, 0.0133, 0.0133, 0.0133, 0.0133, 0.0133, 0.0133,
0.0133, 0.0133, 0.017, 0.015, 0.015, 0.015, 0.015, 0.015, 0.015, 0.015, 0.015, 0.015,
0.015, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01,
0.0033, 0.0033, 0.0033, 0.0033, 0.0033, 0.0033, 0.0033, 0.0033, 0.0033, 0.0033, 0.0033,
0.0033, 0.0033, 0.0033)

weights <- 265:364

names <- c("Liam Galles", "Noah Raymond", "William Hammontree", "James Zaremba",
"Oliver Zurcher", "Benjamin Hilker", "Elijah Loken", "Lucas Lewter", "Mason
Straus", "Logan Work", "Alexander Jarret", "Ethan Wey", "Jacob Adolphsen",
"Michael Solt", "Daniel Welcome", "Henry Portman", "Jackson Tichenor",
"Sebastian Free", "Aiden Papp", "Matthew Lenzi", "Samuel Rinaldo", "David
Goines", "Joseph Asuncion", "Carter Philhower", "Owen Freeborn", "Wyatt Ice",
"John McGuckin", "Jack Soden", "Luke Humfeld", "Jayden Natera", "Dylan Galles",
"Grayson Raymond", "Levi Hammontree", "Isaac Zaremba", "Gabriel Zurcher",
"Julian Hilker", "Mateo Loken", "Anthony Lewter", "Jaxon Straus", "Lincoln
Work", "Joshua Jarret", "Christopher Wey", "Andrew Adolphsen", "Theodore Solt",
"Caleb Welcome", "Ryan Portman", "Asher Tichenor", "Nathan Free", "Thomas
Papp", "Leo Lenzi", "Isaiah Rinaldo", "Charles Goines", "Josiah Asuncion",
"Hudson Philhower", "Christian Freeborn", "Hunter Ice", "Connor McGuckin", "Eli
Soden", "Ezra Humfeld", "Aaron Natera", "Landon Galles", "Adrian Raymond",
"Jonathan Hammontree", "Nolan Zaremba", "Jeremiah Zurcher", "Easton Hilker",
"Elias Loken", "Colton Lewter", "Cameron Straus", "Carson Work", "Robert
Jarret", "Angel Wey", "Maverick Adolphsen", "Nicholas Solt", "Dominic Welcome",
"Jaxon Portman", "Greyson Tichenor", "Adam Free", "Ian Papp", "Austin Lenzi",
"Santiago Rinaldo", "Jordan Goines", "Cooper Asuncion", "Brayden Philhower",
"Roman Freeborn", "Evan Ice", "Ezekiel McGuckin", "Xavier Soden", "Jose
Humfeld", "Jace Natera", "Jameson Adolphsen", "Leonardo Solt", "Bryson Welcome",
"Axel Portman", "Everett Tichenor", "Parker Free", "Kayden Papp", "Miles
Lenzi", "Sawyer Rinaldo", "Jason Goines")
```

```
linemen_weights <- sample(weights, 100, replace = TRUE, prob= probs)
```

```
plot.data <- data.frame(names, linemen_weights)

ggplot(plot.data, aes(linemen_weights)) +
  geom_histogram(binwidth = 20) +
  labs(title = "geom_histogram") +
  theme_minimal()
```

Полученная в результате запуска этого кода гистограмма показана на рис. 1.13.

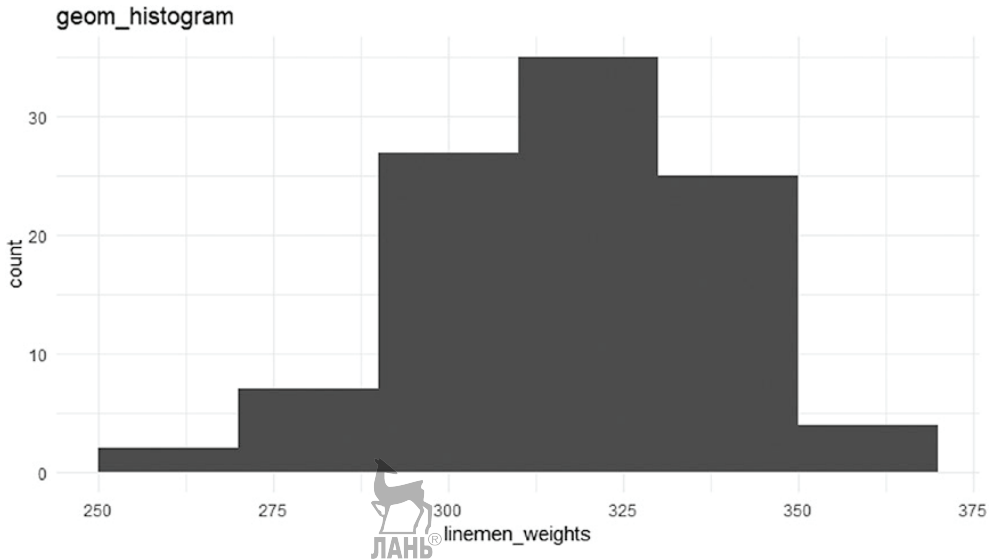


Рис. 1.13 ❖ Гистограмма, построенная при помощи функции *geom_histogram()*

Как и в предыдущем примере, здесь первые строки кода посвящены подготовке данных для отображения, чтобы вы также могли построить визуализацию в своем окружении. Мы использовали фиктивные данные об антропометрических данных нападающих в американском футболе на уровне колледжа. Мы же сконцентрируемся на последних четырех строках кода, где, собственно, и строится визуализация.

Как вы могли догадаться, вся основная работа происходит в функции *geom_histogram()*. Заметьте, что на вход функции мы передаем единственный параметр *binwidth*, поскольку остальные параметры, как и раньше, были унаследованы от функции *ggplot()*. Как понятно из названия, параметр *binwidth* отвечает за установку ширины столбиков. Рекомендуется поиграться со значениями этого параметра, поскольку при использовании значения по умолчанию чаще всего создается неоптимальное количество столбиков. Заметьте, что для построения гистограммы нам бы хватило всего двух строк кода, показанных ниже:

```
ggplot(plot.data, aes(linemen_weights)) +
  geom_histogram(binwidth = 20)
```

Остальные инструкции призваны сделать визуализацию более привлекательной.

- `geom_line()`. Геометрия `geom_line()` используется для создания линейных диаграмм, которые идеально подходят для отображения трендов показателей во времени. Представленный ниже пример демонстрирует создание линейной диаграммы:

```
library(tidyverse)
library(lubridate)

players <- c("Kobe Bryant", "Pau Gasol", "Lamar Odom")
plot.data <- lakers %>%
  filter(
    result == "made" & team == "LAL" & player %in% players
  ) %>%
  mutate(date = ymd(date)) %>%
  group_by(player, date) %>%
  summarize(points = sum(points))

ggplot(plot.data, aes(x=date, y=points, color=player)) +
  geom_line() +
  labs(title = "geom_line") +
  theme_minimal()
```

Полученный в результате запуска этого кода график показан на рис. 1.14.

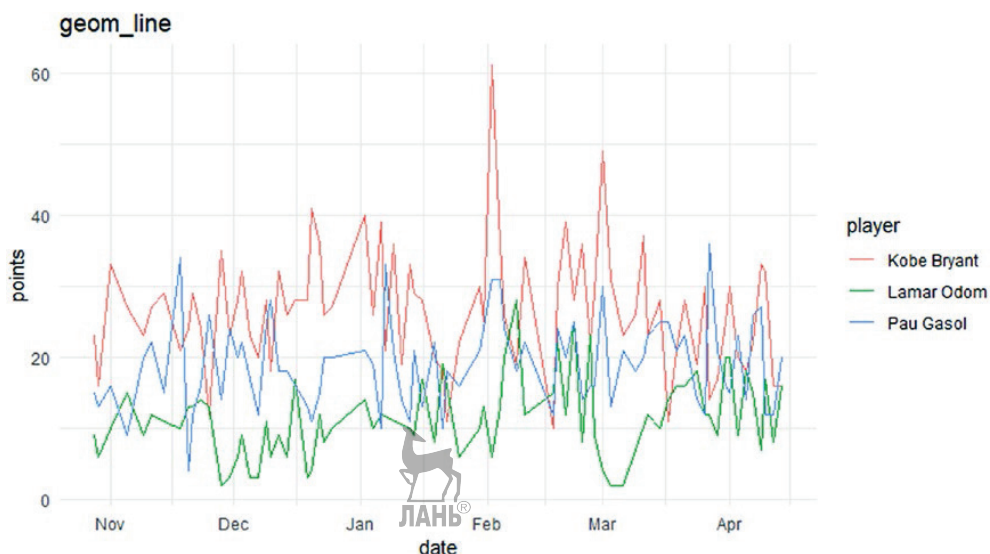


Рис. 1.14 ❖ Линейная диаграмма, построенная при помощи функции `geom_line()`

Как и до этого, сначала здесь происходит подготовка набора данных, чтобы вы могли воспроизвести построение графика в своем окружении. В нашей визуализации мы использовали набор данных *lakers*, идущий вместе с пакетом *lubridate*. В этом наборе данных содержится подробная информация о результатах сезона НБА 2008 года.

Заметьте, что мы использовали эстетику *color* для создания множества линий. В пакете *ggplot2* линии создаются на основании каждого уникального элемента поля, определенного в эстетике *color*. И, по сути, нам снова нужны всего две строки кода, чтобы вывести график на экран: это строка с вызовом функции *ggplot()* и следующая за ней, определяющая геометрию визуализации.

- *geom_point()* и *geom_smooth()*. В следующем примере мы посмотрим, как можно при помощи пакета *ggplot2* строить диаграммы с использованием двух слоев. А именно мы добавим *линию регрессии* (regression line) на диаграмму рассеяния. В общем случае пакет *ggplot2* позволяет накладывать поверх диаграммы рассеяния различные линии тренда, не ограничиваясь только лишь линиями регрессии. Также пакет допускает добавление *доверительных интервалов* (confidence intervals) с выбранным уровнем доверительной вероятности. На момент написания книги такие возможности недоступны при помощи стандартных средств Power BI.

В следующем фрагменте кода показано, как можно создать диаграмму рассеяния и поверх нее наложить линию регрессии:

```
library(tidyverse)

set.seed(1)

plot.data <-
  diamonds %>%
    filter(color == "D") %>%
    sample_n(200)

ggplot(plot.data, aes(x=carat, y=price)) +
  geom_point() +
  geom_smooth(method = "lm") +
  labs(title = "geom_point & geom_smooth") +
  theme_minimal()
```

Полученная в результате диаграмма показана на рис. 1.15.

Представленная на рисунке визуализация основана сразу на двух геометриях: *geom_point()* и *geom_smooth()*. Геометрия *geom_point()* используется для создания диаграммы рассеяния, а *geom_smooth()* – для наложения линии регрессии с доверительным интервалом. По умолчанию доверительный интервал составляет 95 %. Изменить его можно путем модификации параметра *se*. Присвоение параметру *method* значения «*lm*» указывает функции *geom_smooth()* построить именно линию регрессии.

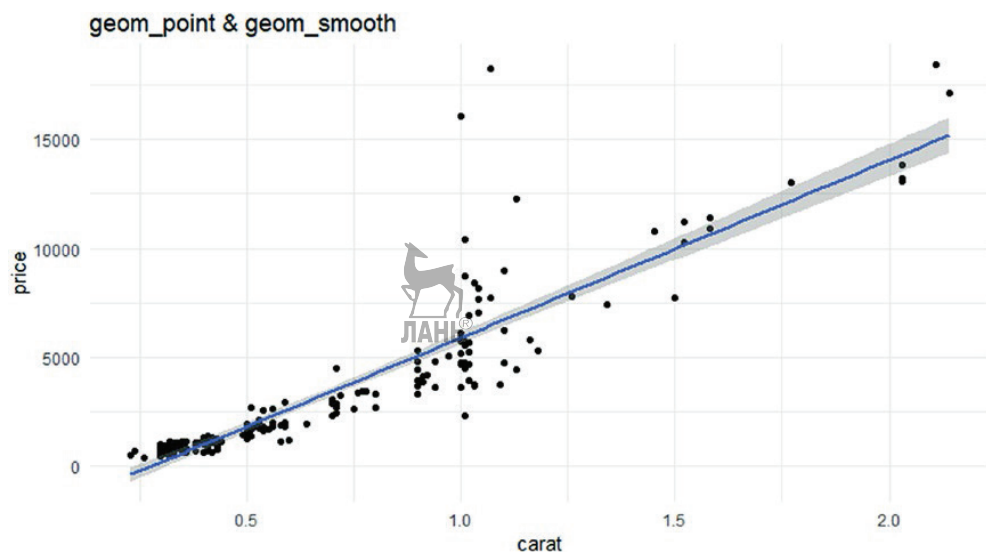


Рис. 1.15 ❖ Визуализация с двумя геометриями: *geom_point()* и *geom_smooth()*

УПРАВЛЕНИЕ ЭСТЕТИКАМИ ЧЕРЕЗ ШКАЛЫ

Ранее вы узнали, как сопоставлять данные с эстетиками при помощи функции *aes()*. Однако лежащий в основе этого действия процесс на самом деле контролируется функциями семейства *scale*. В данном разделе мы подробно поговорим о том, что происходит за сценой.

Начнем с объяснения того, что имеется в виду под *шкалами* (*scale*). Шкалы необходимы каждой эстетике на диаграмме (координатам *x* и *y*, цветовой составляющей и т. д.). Именно они определяют, как именно данные будут шкалированы на графике. Нет никакой необходимости явно объявлять шкалы. Если вы этого не сделаете, пакет *ggplot2* просто воспользуется шкалами по умолчанию. Давайте взглянем на следующий код для примера:

```
ggplot(plot.data, aes(x = carat, y = price)) +  
  geom_point(aes(color = clarity))
```

Здесь шкалы не были определены явно. Таким образом, этот код «под капотом» пакета *ggplot2* будет транслирован в следующий с использованием шкал по умолчанию:

```
ggplot(plot.data, aes(x = carat, y = price)) +  
  geom_point(aes(color = clarity)) +  
  scale_x_continuous() +  
  scale_y_continuous() +  
  scale_color_discrete()
```

Как видите, пакет *ggplot2* использовал значения по умолчанию для всех трех шкал.

Внимательный читатель наверняка заметит определенную зависимость в принципах именования функций шкалирования. И действительно, все названия функций состоят из трех частей: первым идет кодовое слово *scale*, следом за которым располагается имя эстетики для шкалирования, а затем тип шкалы. Все три части разделены символами подчеркивания. Таким образом, две первые шкалы в показанном выше коде предназначены для эстетик *x* и *y* соответственно. Тип шкалы (третье слово) в обоих случаях указан *continuous*, поскольку числовые данные, ассоциированные с этими эстетиками, носят непрерывный характер. Последняя шкала соответствует эстетике *color*. Данные, ассоциированные с этой эстетикой, берутся из поля *clarity*, содержащего дискретные значения, и именно поэтому третьим словом в имени функции, написанной в последней строке кода, является *discrete*.

Если вас устраивают шкалы, выбранные *ggplot2* по умолчанию, нет необходимости вносить в этот аспект какие-то изменения. В то же время у вас всегда есть возможность настроить шкалы вручную, как показано ниже:

```
library(tidyverse)
library(scales)

plot.data <-
  diamonds %>%
  filter(color == "D") %>%
  sample_n(200)

ggplot(plot.data, aes(x = carat, y = price)) +
  geom_point(aes(color = clarity)) +
  labs(
    title = "Chart Title",
    subtitle = "Chart Subtitle",
    caption = "Chart Caption"
  ) +
  scale_x_continuous(name = "Size in Carats") +
  scale_y_continuous(
    name = "Price ($)",
    labels = dollar_format()
  )
```

Заметьте, что мы изменили имя шкалы эстетики *x* при помощи функции *scale_x_continuous()*, а также имя шкалы *y* и формат меток на ней при помощи функции *scale_y_continuous()*. В следующей главе мы рассмотрим больше примеров, демонстрирующих изменение эстетик посредством настройки шкал.

ВСТРОЕННЫЕ В ПАКЕТ GGPLOT2 ТЕМЫ

Пакет *ggplot2* поставляется с предустановленным набором тем, способных разнообразить не содержащие данных элементы на вашей диаграмме. Представляем вам список имеющихся в наличии тем с описаниями автора пакета Хэдли Уикхэма:

- *theme_bw()*: разновидность темы *theme_gray()* с белым фоном и тонкими серыми линиями сетки;
- *theme_linedraw()*: тема с линиями черного цвета различной толщины на белом фоне, напоминающая чертеж;
- *theme_light()*: разновидность темы *theme_linedraw()* со светло-серыми линиями и осями для лучшей видимости значимых данных;
- *theme_dark()*: темнокожий брат темы *theme_light()* – с похожими линиями и темным фоном. Подходит для отображения тонких цветных линий на графиках;
- *theme_minimal()*: минималистичная тема без примечаний на фоне;
- *theme_classic()*: классическая тема с осями *x* и *y* и отсутствием сетки;
- *theme_void()*: пустая тема.

ИСПОЛЬЗОВАНИЕ ВИЗУАЛЬНЫХ ЭЛЕМЕНТОВ R В СЛУЖБЕ POWER BI

Разворачивая элементы визуализации R в рамках службы Power BI, необходимо убедиться в том, что вы используете те же версии пакетов, которые доступны службе. По следующей ссылке можно посмотреть доступные версии пакетов на сервере: <https://docs.microsoft.com/en-us/power-bi/connect-data/service-r-packages-support>.

ВСПОМОГАТЕЛЬНЫЕ ПАКЕТЫ GGPLOT2

Пакет *ggplot2* обладает очень богатым функционалом и содержит в себе все необходимые инструменты визуализации данных. Однако иногда приходится выполнять дополнительные операции, как то преобразование данных, передаваемых Power BI скрипту R перед их визуализацией, особое форматирование информации, недоступное при помощи стандартных средств R и пакета *ggplot2*, или использование тем, не входящих в пакет *ggplot2*.

К счастью, вы не ограничены использованием одного пакета *ggplot2* при построении визуализаций на языке R. Вот лишь несколько вспомогательных пакетов, которые могут быть вам полезны при разработке графиков с использованием пакета *ggplot2* в Power BI. Этот список отнюдь не является исчерпывающим:

- *tidyverse*: метапакет, содержащий коллекцию популярных пакетов R, использующих один фреймворк. В состав этого набора входят пакеты *ggplot* и *dplyr*. Использование пакетов из коллекции *tidyverse* значительно облегчает анализ данных средствами языка R. Больше информации о метапакете *tidyverse* можно найти по адресу www.tidyverse.org;
- *ggthemes*: вспомогательный пакет для *ggplot2*, содержащий дополнительные шкалы, геометрии и темы. Две темы из этого пакета, которые следует отметить особо, – это *theme_few()* и *theme_tufte()*. Они базируются

на принципах признанных экспертов в области визуализации данных Стивена Фью (Stephen Few) и Эдварда Тафти (Edward Tufte) и даже названы в их честь. Больше информации о пакете *ggthemes* можно найти по адресу <https://cran.r-project.org/web/packages/ggthemes/ggthemes.pdf>;

- *scales*: пакет *scales* обеспечивает внутреннюю инфраструктуру шкал для пакета *ggplot2*, а также предоставляет множество функций для преобразования и форматирования данных. Подробнее о пакете *scales* можно почитать по адресу <https://cran.r-project.org/web/packages/scales/scales.pdf>;
- *ggrepel*: в пакете *ggrepel* представлены геометрии, позволяющие разместить метки на диаграмме с минимальными перекрытиями. Этим пакетом можно, например, воспользоваться, если вы хотите добавить метки на диаграмму рассеяния. Больше информации о пакете *ggrepel* можно найти по адресу <https://cran.r-project.org/web/packages/ggrepel/vignettes/ggrepel.html>.

ЗАКЛЮЧЕНИЕ

В данной главе мы поговорили о следующих вещах:

- о фреймворке, который необходимо использовать при создании элементов визуализации R в Power BI;
- о многослойной грамматике графиков, на основе которой построен пакет *ggplot2*;
- о том, как использовать геометрии для добавления слоев на визуализацию;
- об изменении эстетик при помощи функций шкалирования;
- о том, как менять внешний вид диаграмм при помощи тем и функций семейства *theme*.

Целью этой главы было познакомить читателя с концепциями, лежащими в основе пакета *ggplot2*, чтобы подготовить его к нюансам построения более сложных графиков, которыми мы займемся в следующей главе. Теперь, когда вы знаете принципы создания визуальных элементов R в Power BI при помощи *ggplot2*, давайте приступим к построению действительно полезных диаграмм и расскажем историю на основе данных в Power BI!

Глава 2



Создание пользовательских визуализаций на R в Power BI при помощи ggplot2

В предыдущей главе вы познакомились с пакетом *ggplot2* и базовым шаблоном для создания элементов визуализации R в Power BI. В данной главе вы узнаете, насколько выразительными могут быть средства построения графиков с помощью этого мощнейшего пакета. Мы рассмотрим основные принципы создания следующих типов диаграмм:

- *диаграмма с аннотацией* (callout chart), характеризующаяся динамическими заголовками и примечаниями;
- *пузырьковая диаграмма* (bubble chart) с динамическими заголовками, отображающими пять измерений данных;
- *детализированная визуализация прогноза* (forecast visualization), предсказывающая просмотры страниц Википедии Ламара Джексона (Lamar Jackson) и Дешона Уотсона (Deshaun Watson);
- *линейная диаграмма* (line chart), показывающая динамику валового внутреннего продукта по годам с *фоновой заливкой* (background shade), основанной на том, какая политическая партия находилась у власти в то или иное время;
- *географическая карта* (map) с демонстрацией выбранного штата и закрашиванием округов в соответствии с квинтилем численности населения, к которому относится округ;
- *диаграмма квадрантов* (quad chart), сравнивающая игроков LA Lakers на основании набранных очков и сделанных подборов в сезоне 2008–2009 годов;
- *диаграмма рассеяния* (scatter plot) со средним весом и ростом американок с наложенной линией регрессии.

Вы можете с небольшими изменениями использовать представленные здесь наработки в своих сценариях. Кроме того, если вы проявите чуть больше творческого подхода и соберете все лучшее из показанных визуализаций, то сможете построить собственный пользовательский визуальный элемент, отвечающий всем вашим требованиям. Пакет *ggplot2* представляет настолько богатые возможности для креатива, что вам никогда не будет тесно в его рамках. Давайте начнем с создания диаграммы с аннотацией.



ДИАГРАММА С АННОТАЦИЕЙ

Горизонтальная столбчатая диаграмма (horizontal bar chart) – один из самых популярных видов диаграмм. Этот вид подходит для ранжирования категориальных данных на основе метрики. В Power BI столбчатая диаграмма также является одним из основных встроенных элементов визуализации, но обладает при этом довольно базовым функционалом. Вы можете делать определенные улучшения, например выделять столбцы цветом, но серьезных доработок в виде тех же динамических подзаголовков и примечаний без серьезных вмешательств извне вам сделать не удастся. К счастью, все это возможно с пакетом *ggplot2*!

В данном разделе мы построим горизонтальную столбчатую диаграмму, показанную на рис. 2.1.

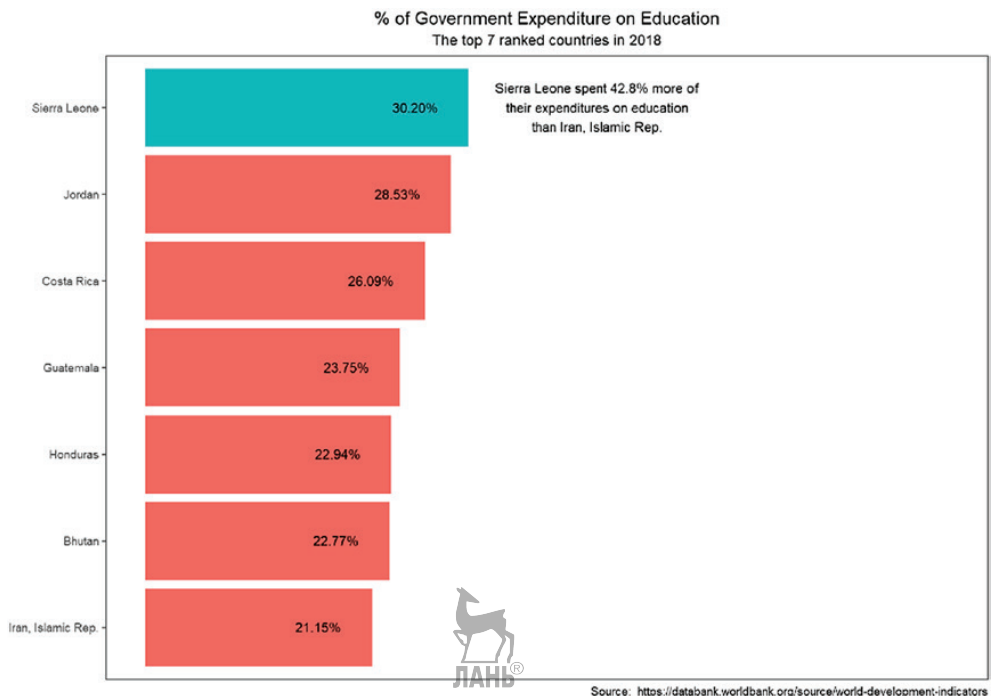


Рис. 2.1 ❖ Горизонтальная столбчатая диаграмма с аннотацией и подписью

На представленной диаграмме показаны семь ведущих стран по проценту выделяемых государством субсидий на образование. Кроме того, на диаграмме присутствует динамическое примечание, сравнивающее первую в списке страну с последней. Наконец, под графиком есть подпись, сообщающая об источнике данных. Давайте посмотрим, какой код необходимо написать, чтобы создать такую визуализацию.

Шаг 1. Загрузите исходные данные

Набор данных, используемый в данном примере, был загружен с источника *World Bank Open Data*. В наборе присутствует информация о процентах государственных отчислений на образование по всем странам мира. Выполните следующие шаги, чтобы получить требуемый набор данных.

1. Откройте сайт data.worldbank.org.
2. Перейдите в поле поиска, расположенное по центру в верхней части сайта, и введите следующий текст: «Government expenditure on education, total (% of government expenditure)». После нажатия на клавишу **Enter** появится список возможных вариантов, один из которых будет в точности совпадать с тем, что вы искали. Выберите этот пункт, и вы будете перенаправлены на страницу по этой теме.
3. В правой части страницы вы увидите раздел для скачивания, в котором вам будут предложены на выбор три формата загружаемого файла: *csv*, *xml* или *Excel*. Выберите вариант *csv*.
4. В результате на ваш компьютер будет загружен архив. На момент написания книги скачиваемый архив содержал три файла: файл с данными, файл с метаданными, несущий информацию о стране, и файл с метаданными, содержащий информацию о показателе. Нам для нашего примера понадобится файл с данными. Файл, который был загружен на мой компьютер в результате этих действий, назывался *API_SE.XPD.TOTL.GB.ZS_DS2_en_csv_v2_715566.csv*.

Примечание. Примите к сведению, что агентства, предоставляющие исходные данные сайту, имеют право менять формат хранения информации, так что в будущем он вполне может измениться.

5. Теперь нам нужно загрузить полученный набор данных в Power BI и привести его к виду, приемлемому для представления на визуализации. Для этого мы используем инструмент Power BI под названием **Получить данные** (GetData). Чтобы загрузить содержимое файла в Power BI, перейдите на вкладку **Главная** (Home), затем нажмите на выпадающую кнопку **Получить данные** и выберите пункт **Текстовый или CSV-файл** (Text/CSV). Перейдите в папку, где хранится загруженный файл с названием *API_SE.XPD.TOTL.GB.ZS_DS2_en_csv_v2_715566.csv*. Это самый большой файл из загруженного архива. Файл также можно скачать из репозитория к этой главе.

- Теперь необходимо выполнить преобразование данных при помощи инструмента Power Query, чтобы привести их к виду, пригодному для визуализации. Результирующий набор данных должен содержать следующие столбцы с ассоциированными типами данных:

Имя столбца	Тип данных
Country	Text
Country Code	Text
Year	Whole number
Total Expend %	Decimal number

В репозитории к данной главе есть файл с расширением *pbix*, содержащий шаги Power Query, необходимые для получения данных и приведения их в указанный вид. Все, что вам необходимо, – это изменить файл-источник. Помните, что шаги преобразования данных в будущем могут потребовать доработки, если в источнике изменится формат хранения информации.

Шаг 2. Создайте срез на основании года на панели фильтров

Перетащите значок фильтра в верхний правый угол холста. Значок фильтра показан на рис. 2.2.

После этого перетащите поле *Year* из набора данных *ExpenditureOnEducationByCountryByYear* на панель **Поле** (Field), как показано на рис. 2.3.



Рис. 2.2 ❖ Иконка, соответствующая визуальному элементу фильтра

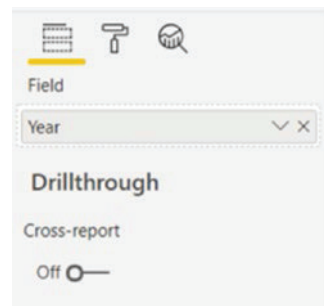


Рис. 2.3 ❖ Панель **Поле** в Power BI

Шаг 3. Настройте визуальный элемент R в Power BI

Перейдите на панель визуализаций и перетащите элемент R в рабочую область. Измените размер визуального элемента под ваши требования. Перетащите столбцы *Country*, *Total Expend %* и *Year* из таблицы *EducationExpenditures* на панель **Поля** (Fields). Если эта панель недоступна, щелкните по добавленному визуальному элементу, и она появится.

Шаг 4. Экпортируйте данные в R Studio для дальнейшей разработки

Выделите в рабочей области элемент визуализации R, чтобы внизу появился **Редактор R-скриптов** (R script editor). Вы увидите окно редактора в нижней части рабочей области. Если редактор не показывается полностью, нажмите на стрелку вверх в правой его части, чтобы раскрыть окно. В результате вы должны увидеть в редакторе код, подобный тому, что показан на рис. 2.4.

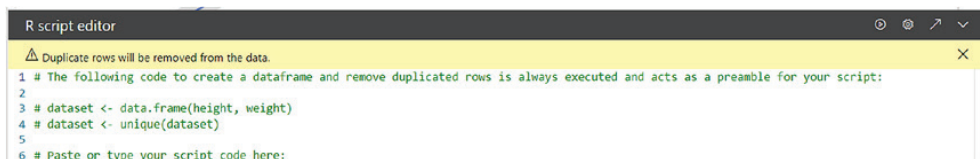


Рис. 2.4 ❖ Редактор R-скриптов в Power BI с комментариями по умолчанию

По комментариям в листинге можно понять, что R взял набор данных, который вы создали путем перетаскивания столбцов на панель **Поля**, и присвоил его объекту *датафрейм* (data frame) с именем *dataset*. Датафрейм представляет собой двумерный набор данных с особыми свойствами. Этот объект создается в представленном выше коде в строке 3. Power BI позволяет передавать в R только наборы данных с уникальными строками, и это обеспечивается вызовом соответствующей функции *unique()* в строке 4 листинга.

Но, как мы уже говорили ранее, редактор R-скриптов в Power BI является далеко не самым комфортным местом для проектирования элементов визуализации R. К счастью, в Microsoft подумали о том, чтобы вы могли разрабатывать скрипты там, где вам привычно, – например, в *R Studio*. Для переноса скрипта в комфортную для вас среду разработки нажмите на значок с диагональной стрелкой в правой части окна редактора, подсвеченный на рис. 2.5.

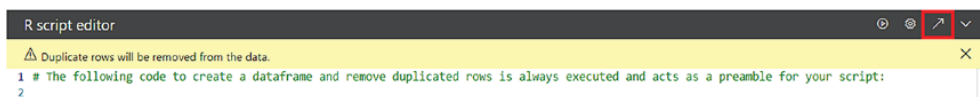


Рис. 2.5 ❖ Нажатие на кнопку со стрелкой позволит открыть комфортную для вас среду разработки

Итак, если вы заранее установили *R Studio* и выбрали эту среду разработки в Power BI, то при нажатии на стрелку будет открыта эта программа, а набор данных будет сохранен в виде временного файла на вашем компьютере. Кроме того, автоматически будет сгенерирован код, создающий переменную *dataset* и загружающий в нее набор данных, переданный в R из Power BI. Если какой-то код был создан вами во встроенном в Power BI редакторе R-скриптов, он также будет перенесен в *R Studio*. На листинге ниже показано, какой код был сгенерирован в *R Studio* в моем случае:

```
# Input load. Please do not change #
`dataset` =
  read.csv('<path to input file>',
    check.names = FALSE,
    encoding = "UTF-8",
    blank.lines.skip = FALSE
  );
# Original Script. Please update your script content here
# and once completed copy below section back to the original
# editing window. The following code to create a dataframe and
# remove duplicated rows is always executed and acts as a
# preamble for your script:
# dataset <-
#   data.frame(
#     Country Name,
#     `Percent Education Expenditure Ranking`
#   )
# dataset <- unique(dataset)
# Paste or type your script code here:
```

Код был отформатирован так, чтобы помещался на страницу. В оригинале строки со второй по седьмую, в которых происходит считывание набора данных, сохраненного во временный файл на вашем компьютере, находились на одной строке. Переменной с полученным в результате датафреймом было присвоено имя *dataset*. Заметьте, что именно это имя использовалось и в Power BI, что не случайно. Сохранение имени переменной позволит вам использовать код, написанный в *R Studio*, в Power BI напрямую – без необходимости выполнять его рефакторинг или переработку.

Шаг 5. Загрузите необходимые пакеты

Три пакета, которые мы будем активно использовать в данном примере, – это *tidyverse*, *scales* и *ggthemes*. Пакет *dplyr* из коллекции *tidyverse* будет использован для преобразования исходных данных, *ggplot2* – для создания визуализации, а *forcats* – для определения правил сортировки. Кроме того, мы воспользуемся некоторыми функциями из пакета *scales* для форматирования чисел, а из пакета *ggthemes* возьмем тему с названием *theme_few()*. Следующим кодом необходимо предварить ваш листинг, чтобы загрузить все необходимые пакеты:

```
library(tidyverse)
library(scales)
library(ggthemes)
```

Шаг 6. Создайте переменные для проверки данных

Следующий код можно использовать для проверки корректности загруженных данных:

```
currentColumns <- sort(colnames(dataset))
requiredColumns <- c("Country", "Total Expend %", "Year")
columnTest <- isTRUE(all.equal(currentColumns, requiredColumns))
reportYear <- unique(dataset$Year)
```

Вам следует убедиться в том, что данные, переданные из Power BI в R, отвечают всем требованиям для их корректной визуализации. В представленном коде выполняются сразу две проверки. Во-первых, мы проверяем, что в R были переданы столбцы, которые нас интересуют, во-вторых – что мы имеем дело с одним переданным годом.

Проверка на столбцы выполняется в первых трех строках представленного кода. Сначала создается вектор с именем *currentColumns*, содержащий отсортированный в алфавитном порядке список столбцов из исходного датафрейма. Сами имена столбцов при этом извлекаются при помощи функции *colnames()*. Функция *sort()* применяется для сортировки итогового перечня столбцов, что необходимо для последующего сравнения. В следующей строке кода создается символьный вектор с именем *requiredColumns*, содержащий список имен столбцов, которые ожидает получить на вход наш визуальный элемент. Здесь имена колонок также необходимо перечислять в алфавитном порядке.

Чтобы проверить идентичность этих списков, необходимо не только убедиться в том, что они содержат одинаковые элементы, но и расположить эти элементы в одном порядке. Для проверки равенства списков мы воспользовались стандартной функцией языка R *all.equal()*. Она возвращает TRUE, если сравниваемые символьные векторы являются идентичными. В противном случае возвращается информация о причине их неидентичности. Нам необходимо вернуть булево выражение, и чтобы обеспечить это, достаточно обернуть функцию *all.equal()* в другую функцию, носящую имя *isTRUE()*. В последней строке представленного кода собираются уникальные годы, представленные в датафрейме. Эту информацию мы используем на следующем шаге.

Шаг 7. Выполните проверку данных

Таким образом выглядит шаблон для проверки данных:

```
if (length(reportYear) == 1 & columnTest) {
  <code to produce visual>
} else {
  plot.new()
  title("The data supplied did not meet the requirements of the chart.")
}
```

Если в нашем наборе данных представлен только один год и проверка идентичности столбцов прошла успешно, создаем наполнение нашего элемента визуализации. В противном случае будет создана пустая диаграмма с сообщением о том, что исходные данные не отвечают требованиям визуализации.

Шаг 8. Добавьте столбцы к набору данных, необходимые для нашей визуализации

Теперь начнем писать скрипт создания визуализации при условии, что все требования к набору данных соблюдены. Для начала расширим наш датафрейм за счет дополнительных столбцов, необходимых для нашей визуализации, написав следующий код:

```
plotdata <-
  dataset %>%
  mutate(
    rank = dense_rank(desc(`Total Expend %`)),
    callout = ifelse(rank == 1, TRUE, FALSE)
    Country = fct_reorder(Country, rank, .desc = TRUE),
  ) %>%
  filter(rank <= 7)
```

Этот код необходимо вставить на место заполнителя `<code to produce visual>` из предыдущего листинга. Здесь мы воспользовались пакетом *dplyr* из коллекции *tidyverse* для добавления нужных нам столбцов. В первой строке кода происходит объявление переменной *plotdata* и присвоение ей дальнейшего выражения. В следующей строке говорится о том, что за основу мы берем наш датафрейм, который передаем в качестве первого аргумента в функцию *mutate()* при помощи оператора конвейера (pipe operator) `%>%`.

Примечание. Оператор конвейера находится в пакете *magrittr*, который загружается вместе с пакетом *dplyr*. При помощи него можно неявно передать результат выражения или значение на вход следующей функции в качестве первого аргумента. Такие цепочки функций делают код более легким для восприятия.

Функция *mutate()* используется для добавления двух новых столбцов – *rank* и *callout* – и изменения исходного порядка сортировки столбца *Country*. При создании столбца *rank* применяется функция *dense_rank()* из пакета *dplyr* для ранжирования строк на основании столбца *Total Expend %* в порядке убывания.

Столбец *callout* мы будем использовать в качестве индикатора для первой страны в списке. При создании этого столбца мы применили функцию *ifelse()*, возвращающую значение TRUE только в случае, если значение столбца *rank* равно единице, и FALSE во всех остальных случаях. Для столбца *Country* мы использовали функцию *fct_reorder()* из пакета *forcats*, чтобы настроить сортировку на основании столбца *rank*. Это возможно по причине того, что тип данных столбца *Country* – *factor*.

Примечание. Фактор (*factor*) представляет собой особый тип данных в языке R, используемый для категориальных данных. У этого типа данных много преимуществ, включая возможность выполнять сортировку на основании другого столбца.

В результате столбец *Country* будет отсортирован по полю *rank*. Определенный в функции порядок сортировки будет использован в визуализации.

Наконец, функция *filter()*, как ясно из названия, используется для осуществления фильтрации нашего датафрейма. Фильтр применяется к столбцу *rank* для ограничения итогового списка семью ведущими странами.

Шаг 9. Создайте переменные для динамических составляющих диаграммы

Одним из преимуществ визуализаций скрипта R над встроенными визуальными элементами Power BI является возможность добавлять на диаграмму динамические подписи и примечания. Более того, вы можете не просто динамически создать текст, который разместите на диаграмме, но и полностью контролировать его позицию в рамках визуализации. На этом шаге мы создадим переменные, которые будут использоваться при создании динамической аннотации и определении ее позиции на экране. Еще мы позаботимся о переменных, которые помогут нам в создании заголовка и подзаголовка диаграммы, а также подписи. Взгляните на следующий код:

```
countryToAnnotate <- plotdata$`Country`[plotdata$rank == 1]
minExpenditure <- min(plotdata$`Total Expend %`)
maxExpenditure <- max(plotdata$`Total Expend %`)

minCountry <-
  plotdata$`Country`[
    which(plotdata$`Total Expend %` == minExpenditure)
  ]
minCountry <- paste(minCountry, collapse = " & ")

maxCountry <-
  plotdata$`Country`[
    which(plotdata$`Total Expend %` == maxExpenditure)
  ]
maxCountry <- paste(maxCountry, collapse = " & ")

mainTitle <- "% of Government Expenditure on Education"

subTitle <-
  paste(
    "The top 7 ranked countries in",
    reportYear,
    sep = " "
  )

caption <- "Source: https://databank.worldbank.org/source/world-development-indicators"

label_val <-
  str_wrap(
    paste(
      maxCountry,
      "spent",
      percent((maxExpenditure/minExpenditure-1)),
```



```

        "more of their expenditures on education than",
        minCountry,
        sep = " "
    ),
    width = 35
)

```

Ниже приведем описание каждой из созданных переменных:

- *countryToAnnotate*: здесь у нас будет первая в списке страна. Именно к этой стране мы будем добавлять динамическую аннотацию;
- *minExpenditure*: в данной переменной хранится результат вызова функции *min()* для получения минимального значения из столбца *Total Expend %*. Позже мы используем эту информацию для определения страны с минимальными расходами на образование;
- *maxExpenditure*: в этой переменной хранится результат вызова функции *max()* для получения максимального значения из столбца *Total Expend %*. Эту информацию мы используем для определения страны с максимальными расходами на образование;
- *minCountry*: в переменную *minExpenditure* мы сохранили минимальный процент расходов государства на образование среди ведущих семи стран по этому показателю. Теперь нам остается найти, какой стране соответствует этот процент. Для этого мы возьмем подмножество элементов из столбца *Country* датафрейма *plotdata*, включающее страны, для которых значение поля *Total Expend %* равно переменной *minExpenditure*. В этом нам поможет функция *which()*. Эта функция возвращает индексы TRUE для тех элементов, для которых выполняется наше условие, – таким образом мы можем получить страны с минимальными расходами на образование. Да, в датафрейме *plotdata* таких стран может оказаться несколько, и в следующей строке кода мы решим эту проблему путем объединения их в единый скалярный символьный вектор при помощи функции *paste()*. При этом страны в этом векторе будут разделяться символом амперсанда (&). Вы можете рассмотреть пример с несколькими странами с одинаковым минимальным уровнем затрат на образование, если выберете в отчете Power BI 2005 год;
- *maxCountry*: функцию *which()* мы также используем для получения списка стран с максимальными расходами, как и в предыдущей переменной;
- *mainTitle*: в этой переменной будет храниться статическая строка для заголовка диаграммы;
- *subtitle*: здесь мы будем динамически собирать строку для подзаголовка визуализации с использованием выбранного пользователем года;
- *caption*: в данной переменной мы просто будем хранить название сайта, послужившего нам в качестве источника информации;
- *label_val*: эта переменная будет хранить динамически собранную аннотацию, сравнивающую расходы правительства на образование для первой и последней стран в представленном списке. Строка в переменной собирается с помощью знакомой уже нам функции *paste()* путем объединения ранее определенных переменных *maxCountry* и *minCountry*, а также расчета разницы в процентах между первой страной и по-

следней. Аннотация получится довольно длинной и не поместится на элемент визуализации, если выводить ее в одну строку. Решим эту проблему при помощи функции `str_wrap()` из пакета `stringr`. Эта функция предлагает очень богатые возможности форматирования текста, одной из которых – контролем максимальной ширины текста – мы и воспользовались. Присвоив атрибуту `width` значение 35, мы тем самым гарантируем, что аннотация будет смотреться на диаграмме нормально.

Шаг 10. Постройте диаграмму при помощи функции `ggplot()`

Теперь воспользуемся основной функцией создания графиков – `ggplot()`:

```
R >-
ggplot(
  data = plotdata,
  aes(x = `Country`, y = `Total Expend %`, fill = callout,
      label = percent(`Total Expend %`))
)
```

Эстетики `x` и `y` не нуждаются в дополнительных пояснениях. Эстетике `fill` мы присваиваем столбец (переменную), на основании которого будет выполняться цветовая заливка столбиков, а эстетика `label` говорит R о том, какой столбец будет использован для добавления меток на диаграмму.

Шаг 11. Добавьте слой со столбчатой диаграммой на визуальный элемент

На этом шаге пришла пора добавить на наш элемент визуализации первый слой, представляющий собой столбчатую диаграмму, при помощи функции `geom_col()`. Также вы можете использовать функцию `geom_bar()`. Мы выбрали функцию `geom_col()`, поскольку для нее значение аргумента `stat` по умолчанию равно `stat_identity`, а значит, отображаться будут именно значения эстетики `y`, а не количество элементов. Если бы мы использовали функцию `geom_bar()`, нам пришлось бы явным образом устанавливать аргумент `stat` в значение `stat_identity`:

```
R >-
ggplot(
  data = plotdata,
  aes(x = `Country`, y = `Total Expend %`, fill = callout,
      label = percent(`Total Expend %`))
) +
geom_col()
```

Шаг 12. Добавьте текстовый слой на визуальный элемент

Следующим шагом мы добавим на наш визуальный элемент слой с метками:

```
p <-
  ggplot(
    data = plotdata,
    aes(x = `Country`, y = `Total Expend %`, fill = callout,
        label = percent(`Total Expend %`))
  ) +
  geom_bar(stat="identity", aes(fill = callout)) +
  geom_text(nudge_y = -0.05)
```

Здесь мы добавили на диаграмму еще один слой с метками при помощи функции `geom_text()`. Обязательными аргументами для этой функции являются `x`, `y` и `label`, и все они были унаследованы от соответствующих эстетик функции `ggplot()`. Аргументы `x` и `y` отвечают за положение меток, а `label` определяет, что именно будет выводиться. В данном случае мы выводим значение поля `Total Expend %`, отформатированное в виде процентов. По умолчанию метки немного выходят за границы столбиков на диаграмме, а нам требуется, чтобы они полностью попадали в их границы, – так метки гарантированно не будут накладываться на аннотацию, которую мы впоследствии наложим. Воспользуемся аргументом `nudge_y`, чтобы сместить метки внутрь столбиков. На рис. 2.6 показано, как будут размещены метки без явного указания аргумента `nudge_y`.

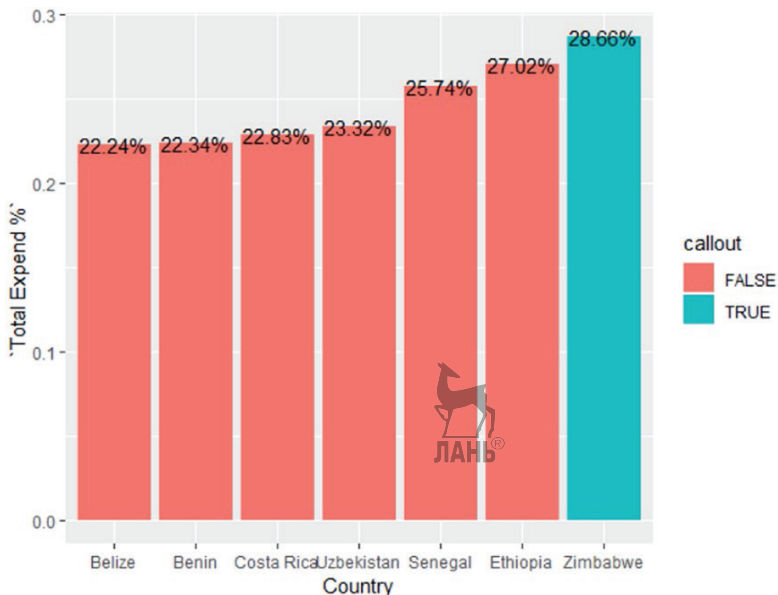


Рис. 2.6 ❖ Размещение меток на столбчатой диаграмме по умолчанию

На рис. 2.7 видно, как метки переместились внутрь столбиков после присвоения аргументу *nudge_y* значения -0.05 .



Рис. 2.7 ❖ Смещение меток на столбчатой диаграмме

Шаг 13. Измените ось y

В следующем фрагменте кода используется функция *scale_y_continuous()* для модификации оси y:

```
p <-
  ggplot(
    data = plotdata,
    aes(x = 'Country', y = 'Total Expend %', fill = callout,
        label = percent('Total Expend %'))
  ) +
  geom_bar(stat="identity", aes(fill = callout)) +
  geom_text(nudge_y = -0.05) +
  scale_y_continuous(
    limits = c(0,0.75),
    labels = NULL,
    breaks = NULL
  )
```

Здесь мы явно задаем границы для оси y как диапазон от 0 до 0.75. Мы делаем это, чтобы пользователю было удобнее. По умолчанию все оси на диаграммах масштабируются автоматически в зависимости от данных. Это способно исказить общую картину при сравнительном анализе лет. Таким

образом, мы предпочли жестко зафиксировать границы оси, чтобы избежать ее автоматического масштабирования. Аргументы *labels* и *breaks* используются для указания формата вставки меток на ось. Поскольку мы указали эти характеристики явно, нет необходимости повторно задавать эти параметры. Поэтому мы присвоили им значение NULL.

Шаг 14. Преобразуйте вертикальную столбчатую диаграмму в горизонтальную

Развернуть столбики диаграммы на 90° можно, воспользовавшись функцией *coord_flip()*, как показано в следующем фрагменте кода:

```
p <-
  ggplot(
    data = plotdata,
    aes(x = `Country`, y = `Total Expend %`, fill = callout,
        label = percent(`Total Expend %`))
  )
  ) +
  geom_bar(stat="identity", aes(fill = callout)) +
  geom_text(nudge_y = -0.05) +
  scale_y_continuous(
    limits = c(0,0.75),
    labels = NULL,
    breaks = NULL
  ) +
  coord_flip()
```

Функция *coord_flip()* меняет местами оси *x* и *y*, в результате чего мы получим горизонтальную столбчатую диаграмму, как показано на рис. 2.8.

Шаг 15. Добавьте на диаграмму динамическую аннотацию

Снабдить график аннотацией можно при помощи функции *annotate()*, как показано в следующем фрагменте кода:

```
p <-
  ggplot(
    data = plotdata,
    aes(x = `Country`, y = `Total Expend %`, fill = callout,
        label = percent(`Total Expend %`))
  )
  ) +
  geom_bar(stat="identity", aes(fill = callout)) +
  geom_text(nudge_y = -0.05) +
  scale_y_continuous(
    limits = c(0,0.75),
```

```

    labels = NULL,
    breaks = NULL
  ) +
  coord_flip() +
  annotate(
    "text",
    label = label_val,
    x = countryToAnnotate[1],
    y = maxExpenditure + 0.12
  )

```

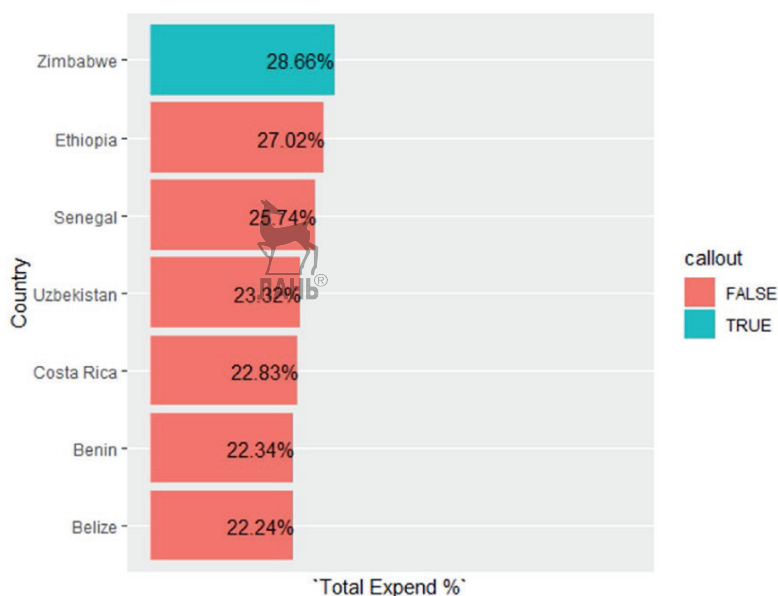


Рис. 2.8 ❖ Превращение вертикальной столбчатой диаграммы в горизонтальную

Мы хотим добавить текстовую аннотацию на диаграмму, так что первым аргументом в функцию *annotate()* передаем «text». Затем мы присваиваем значение переменной *label_val*, в которой хранится текст аннотации, аргументу *label*. Наконец, нам необходимо определить местоположение аннотации, что делается путем присвоения значений аргументам *x* и *y*. Аргументу *x* мы присвоим имя страны, для которой собираемся сделать аннотацию, что позволит расположить ее на нужном уровне. Для аргумента *y* мы выбираем значение, на несколько единиц превосходящее значение с максимальными расходами (*maxExpenditure*), чтобы аннотация не перекрывала столбец. Обратите внимание, что, поскольку мы ранее уже вызывали функцию *coord_flip()*, сейчас при работе с осью *x* кажется, что мы работаем с осью *y*, и наоборот. Результат добавления слоя с аннотацией показан на рис. 2.9.

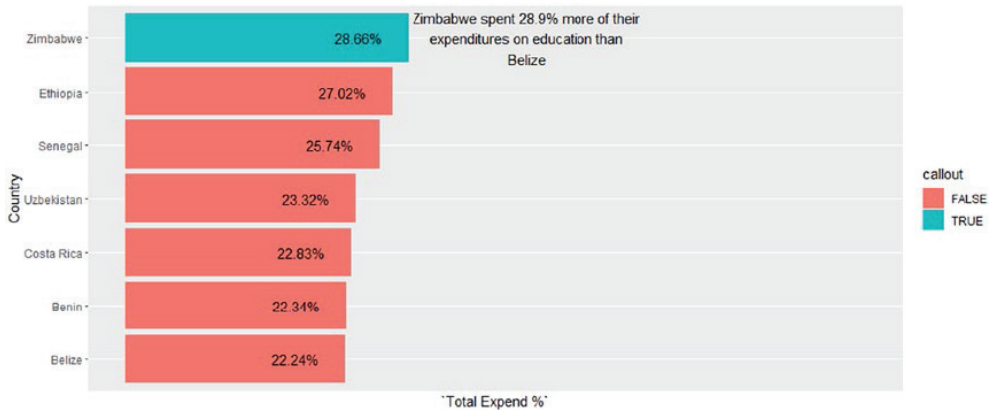


Рис. 2.9 ❖ Горизонтальная столбчатая диаграмма с динамической аннотацией

Шаг 16. Добавьте динамические заголовки и подпись на визуальный элемент

Теперь вам необходимо снабдить диаграмму динамическими заголовками и подписью, что можно сделать при помощи функции *labs()*. Переменные *mainTitle*, *subTitle* и *caption* мы инициализировали заранее.

```
p <-
  ggplot(
    data = plotdata,
    aes(x = `Country`, y = `Total Expend %`, fill = callout,
        label = percent(`Total Expend %`))
  ) +
  geom_bar(stat="identity", aes(fill = callout)) +
  geom_text(nudge_y = -0.05) +
  scale_y_continuous(
    limits = c(0,0.75),
    labels = NULL,
    breaks = NULL
  ) +
  coord_flip() +
  annotate(
    "text",
    label = label_val,
    x = countryToAnnotate[1],
    y = maxExpenditure + 0.12
  ) +
  labs(
    title = mainTitle,
    subtitle = subTitle,
    caption = caption
  )
```

С функцией *labs()* все совершенно понятно, так что мы не будем тратить на нее много времени.

Шаг 17. Удалите метки с осей x и y

Нам нет необходимости явно задавать названия осей *x* и *y*, поскольку мы снабдили диаграмму полноценными метками. И чтобы избавиться от них, передадим в виде аргумента *label* в функции *xlab()* и *ylab()* значение *NULL*, как показано во фрагменте кода ниже:

```
p <-
  ggplot(
    data = plotdata,
    aes(x = `Country`, y = `Total Expend %`, fill = callout,
        label = percent(`Total Expend %`))
  ) +
  geom_bar(stat="identity", aes(fill = callout)) +
  geom_text(nudge_y = -0.05) +
  scale_y_continuous(
    limits = c(0,0.75),
    labels = NULL,
    breaks = NULL
  ) +
  coord_flip() +
  annotate(
    "text",
    label = label_val,
    x = countryToAnnotate[1],
    y = maxExpenditure + 0.12
  ) +
  labs(
    title = mainTitle,
    subtitle = subTitle,
    caption = caption
  ) +
  xlab(label = NULL) +
  ylab(label = NULL)
```

Шаг 18. Удалите легенду с диаграммы

Если еще раз взглянуть на рис. 2.9, несложно обнаружить на нем легенду для столбца *callout*. Этот столбец ассоциирован у нас с эстетикой *fill*, отвечающей за цвет заливки наших столбцов. Мы используем эту эстетику для выделения цветом страны, лидирующей по объемам государственных расходов на образование. Таким образом, легенда в данном случае будет просто избыточна, и нам необходимо от нее избавиться. Сделаем это, передав аргументу *fill* функции *guides()* значение *FALSE*, как показано ниже:

```

p <-
  ggplot(
    data = plotdata,
    aes(x = `Country`, y = `Total Expend %`, fill = callout,
        label = percent(`Total Expend %`))
  )
  ) +
  geom_bar(stat="identity", aes(fill = callout)) +
  geom_text(nudge_y = -0.05) +
  scale_y_continuous(
    limits = c(0,0.75),
    labels = NULL,
    breaks = NULL
  ) +
  coord_flip() +
  annotate(
    "text",
    label = label_val,
    x = countryToAnnotate[1],
    y = maxExpenditure + 0.12
  ) +
  labs(
    title = mainTitle,
    subtitle = subTitle,
    caption = caption
  ) +
  xlab(label = NULL) +
  ylab(label = NULL) +
  guides(fill = FALSE)

```



Шаг 19. Измените внешний вид диаграммы при помощи темы `theme_few()`

Осталось подкорректировать внешний вид нашей визуализации, чтобы она отвечала базовым принципам признанного эксперта в этой области Стивена Фью. Для этого применим тему `theme_few()` из пакета `ggthemes` к нашей диаграмме следующим образом:

```

p <-
  ggplot(
    data = plotdata,
    aes(x = `Country`, y = `Total Expend %`, fill = callout,
        label = percent(`Total Expend %`))
  )
  ) +
  geom_bar(stat="identity", aes(fill = callout)) +
  geom_text(nudge_y = -0.05) +
  scale_y_continuous(
    limits = c(0,0.75),
    labels = NULL,

```



```

    breaks = NULL
  ) +
  coord_flip() +
  annotate(
    "text",
    label = label_val,
    x = countryToAnnotate[1],
    y = maxExpenditure + 0.12
  ) +
  labs(
    title = mainTitle,
    subtitle = subTitle,
    caption = caption
  ) +
  xlab(label = NULL) +
  ylab(label = NULL) +
  guides(fill=FALSE) +
  theme_few()

```



Шаг 20. Расположите заголовки по центру

Воспользуемся функцией *theme()*, чтобы центрировать на визуальном элементе заголовков и подзаголовков диаграммы:

```

p <-
  ggplot(
    data = plotdata,
    aes(x = `Country`, y = `Total Expend %`, fill = callout,
        label = percent(`Total Expend %`)
    )
  ) +
  geom_bar(stat="identity", aes(fill = callout)) +
  geom_text(nudge_y = -0.05) +
  scale_y_continuous(
    limits = c(0,0.75),
    labels = NULL,
    breaks = NULL
  ) +
  coord_flip() +
  annotate(
    "text",
    label = label_val,
    x = countryToAnnotate[1],
    y = maxExpenditure + 0.12
  ) +
  labs(
    title = mainTitle,
    subtitle = subTitle,
    caption = caption
  ) +
  xlab(label = NULL) +
  ylab(label = NULL) +

```





```
guides(fill=FALSE) +
theme_few() +
theme(
  plot.title = element_text(hjust = 0.5),
  plot.subtitle = element_text(hjust = 0.5)
)
```

Функция *theme()* используется для индивидуальной настройки выбранной темы, то есть элементов, не представляющих сами данные. Вы можете ввести команду *?theme()* в консоли, чтобы получить доступ к полной документации по этой функции и всем ее возможностям. *element_text()* представляет собой функцию семейства *element*, ассоциированную с компонентами *plot.title* и *plot.subtitle*. Присвоение аргументу *hjust* функции *element_text()* значения 0.5 позволяет разместить соответствующий заголовок по центру.

Шаг 21. Перенесите код в Power BI

Ниже в полном объеме представлен код, который у нас получился. Скопируйте его и перенесите во встроенный редактор R-скриптов Power BI. Если вы все шаги выполнили корректно, вам не потребуется менять код при переносе.

```
library(tidyverse)
library(scales)
library(ggthemes)

currentColumns <- sort(colnames(dataset))
requiredColumns <- c("Country", "Total Expend %", "Year")
columnTest <- isTRUE(all.equal(currentColumns, requiredColumns))
reportYear <- unique(dataset$Year)

if (length(reportYear) == 1 & columnTest) {
  plotdata <-
    dataset %>%
    mutate(
      rank = dense_rank(desc(`Total Expend %`)),
      `Country` = fct_reorder(`Country`, rank, .desc = TRUE),
      callout = ifelse(rank == 1, TRUE, FALSE)
    ) %>%
    filter(rank <= 7)

  countryToAnnotate <- plotdata$`Country`[plotdata$rank == 1]
  minExpenditure <- min(plotdata$`Total Expend %`)
  maxExpenditure <- max(plotdata$`Total Expend %`)

  minCountry <-
    plotdata$`Country`[
      which(plotdata$`Total Expend %` == minExpenditure)]
  minCountry <- paste(minCountry, collapse = " & ")

  maxCountry <-
    plotdata$`Country`[
      which(plotdata$`Total Expend %` == maxExpenditure)]
}
```

```

maxCountry <- paste(maxCountry, collapse = " & ")

mainTitle = "% of Government Expenditure on Education"
subTitle =
  paste(
    "The top 7 ranked countries in",
    reportYear,
    sep = " ")
caption = "Source: https://databank.worldbank.org/source/world-development-indicators"
label_val <-
  str_wrap(
    paste(maxCountry, "spent",
          percent((maxExpenditure/minExpenditure-1)),
          "more than", minCountry, sep = " "),
    width = 25
  )

p <-
  ggplot(
    data = plotdata,
    aes(x = `Country`, y = `Total Expend %`, fill = callout,
        label = percent(`Total Expend %`))
  ) +
  geom_bar(stat="identity", aes(fill = callout)) +
  geom_text(nudge_y = -0.05) +
  scale_y_continuous(
    limits = c(0,0.75), labels = NULL, breaks = NULL) +
  coord_flip() +
  annotate(
    "text",
    label = label_val,
    x = countryToAnnotate[1],
    y = maxExpenditure + 0.1) +
  labs(
    title = mainTitle,
    subtitle = subTitle,
    caption = caption
  ) +
  xlab(label = NULL) +
  ylab(label = NULL) +
  guides(fill = FALSE) +
  theme_few() +
  theme(
    plot.title = element_text(hjust = 0.5),
    plot.subtitle = element_text(hjust = 0.5))
p
} else {
  plot.new()
  title("The data supplied did not meet the requirements of the chart.")
}

```

В случае если данные будут содержать структурные ошибки, будет создана пустая диаграмма с заголовком **The data supplied did not meet the requirements of the chart** (Представленные данные не отвечают требованиям диаграммы).

Пузырьковая диаграмма

Пузырьковая диаграмма (bubble chart) является еще одним очень распространенным типом визуализации. Можно думать о ней как об уже знакомой вам диаграмме рассеяния с дополнительным измерением-атрибутом, представляющим собой размер точек. Таким образом, при помощи стандартной пузырьковой диаграммы вы можете анализировать сразу пять различных измерений:

- координаты x и y каждого пузырька представляют два измерения;
- цвет (цветовая заливка внутренней области пузырька);
- размер пузырька;
- метка пузырька.

С пакетом *ggplot2* вы можете пойти еще дальше. Например, вы можете раскрасить границы пузырьков отдельным цветом, что позволит включить в анализ еще одно измерение. В примере из этого раздела мы будем использовать данные из вымышленной профессиональной футбольной лиги по игрокам амплуа раннинбек или бегущий (running back) в разрезе шести измерений: имя, рост, вес, общее количество преодоленных ярдов, конференция и дивизион. Диаграмма, которую мы будем строить в данном разделе, показана на рис. 2.10. Давайте пройдем по шагам.

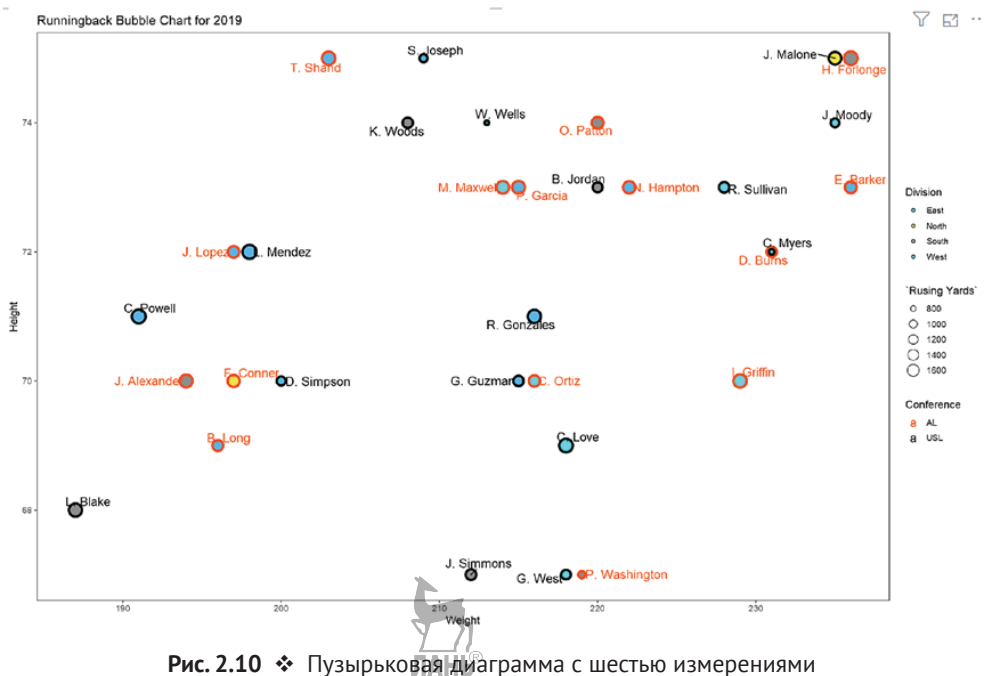


Рис. 2.10 ❖ Пузырьковая диаграмма с шестью измерениями

Шаг 1. Загрузите исходные данные

Набор данных, который мы будем использовать в этом примере, взят из вымышленной футбольной лиги. В лиге присутствуют две конференции: AL (American League) и USL (United States League). Каждая конференция разбита на четыре дивизиона: North, South, East и West. В качестве данных мы, помимо конференции и дивизиона, будем использовать имена игроков, их рост и вес, а также общее количество преодоленных ярдов. Файл с исходными данными, который можно загрузить из репозитория этой главы, называется *FakeFootballLeagueData.csv*.

Шаг 2. Загрузите данные в Power BI

Загрузите полученную информацию в модель данных Power BI посредством инструмента получения данных (GetData). В табл. 2.1 показаны названия столбцов в таблице и соответствующие им типы данных.

Таблица 2.1. Столбцы таблицы с типами данных

Имя столбца	Тип данных
ID	Whole Number
Year	Whole Number
Key	Whole Number
FN	Text
LN	Text
Weight	Whole Number
Height	Whole Number
Rushing Yards	Whole Number
Division	Text
Conference	Text

Шаги с преобразованием исходных данных к требуемому виду содержатся в файле *PBI_FakeFootballLeague.pbix*.

Шаг 3. Создайте срез на основании года

Для начала создайте срез в отчете по полю *Year*, сначала перетащив его на рабочую область, а затем нажав на элемент среза на панели визуализации. Убедитесь, что поле *Year* выбрано перед созданием среза. Иконка среза на панели визуализации содержит изображение воронки.

Шаг 4. Выполните базовую настройку визуального элемента R

Перейдите на панель визуализации и перетащите в рабочую область визуальный элемент R. Измените его размер до желаемого. Перенесите поля *ID*, *Year*, *FN*, *LN*, *Weight*, *Height*, *Rushing Yards*, *Division* и *Conference* из таблицы *FakeFootballLeague* на панель **Поля** (Fields). Если эта панель недоступна, выделите созданный элемент визуализации, и она появится.

Шаг 5. Экпортируйте данные в R Studio для разработки элемента

Нажмите на диагональную стрелку в правой части встроенного в Power BI редактора скриптов R. Это приведет к экспорту данных и базового кода на языке R в *R Studio*. Обратитесь к четвертому шагу инструкции по созданию диаграммы с аннотацией за дополнительной информацией.

Шаг 6. Загрузите требуемые пакеты

Следующий код позволит вам загрузить все необходимые для создания диаграммы пакеты:

```
library(tidyverse)
library(ggrepel)
library(ggthemes)
```

Пакеты *dplyr* и *ggplot2*, которые мы будем использовать в нашем скрипте, загружаются вместе с коллекцией пакетов *tidyverse*. В частности, пакет *dplyr* применяется для преобразования данных, полученных из Power BI, в подходящий для анализа вид, а с помощью пакета *ggplot2* будет произведен вывод диаграммы. Пакет *ggrepel* понадобится нам для изменения позиции меток для точек с данными, а *ggthemes* – для настройки визуального оформления диаграммы.

Шаг 7. Создайте переменные для проверки данных

```
currentColumns <- sort(colnames(dataset))
requiredColumns <- c("Conference", "Division", "FN", "Height",
                    "ID", "LN", "Rushing Yards", "Weight",
                    "Year")
columnTest <- isTRUE(all.equal(currentColumns, requiredColumns))
reportYear <- unique(dataset$Year)
```

В переменной *currentColumns* мы сохраним имена столбцов в датафрейме, переданном в R из Power BI. При этом имена будут автоматически отсортированы по возрастанию для дальнейшего сравнения. В переменной *requiredColumns* будем хранить требуемый нам список столбцов в виде символьного вектора. Имена этих столбцов также будут упорядочены по возрастанию. Переменная *columnTest* будет содержать результат сравнения колонок. Здесь мы использовали функцию *all.equal()* для проверки содержимого переменных *currentColumns* и *requiredColumns* на равенство. Если списки столбцов равны, будет возвращено значение TRUE, в противном случае вернется описание несоответствия. Нам необходимо обработать булево значение на выходе, поэтому мы обернем функцию *all.equal()* в другую функцию *isTRUE()*, в результате чего в случае равенства списков получим значение TRUE, а иначе – FALSE. После этого мы используем функцию *unique()* для получения уникального списка элементов из столбца *Year*.

Шаг 8. Создайте код для проверки

Шаблон, используемый для выполнения проверки исходных данных, показан ниже:

```
if(length(reportYear) == 1 & columnTest) {
  <code to create the visual>
} else{
  plot.new()
  title("The data supplied did not meet the requirements of the
        chart.")
}
```

В условном выражении *if* выполняется проверка двух условий: в векторе *reportYear* должен содержаться единственный элемент, а значение переменной *columnTest* должно быть равно TRUE. Если оба условия выполняются, запускается код для создания визуального элемента. В противном случае будет создана пустая диаграмма с заголовком о том, что исходные данные не отвечают требованиям визуализации.

Шаг 9. Определите цвета для конференций и дивизионов

Одним из преимуществ пакета *ggplot2* является возможность задавать собственные цвета, которые будут использоваться при визуализации данных. Вы можете определять цветовую схему понятными пакету *ggplot2* названиями цветов или использовать распространенную шестнадцатеричную систему представления цветовых оттенков. В данном случае мы воспользуемся второй опцией и с ее помощью определим именованные символьные векторы

(named character vector). В таких векторах мы будем хранить цветовые схемы для дивизионов и конференций. Ниже представлен код, который позволит определить цвета для всех элементов в нашем наборе данных:

```
divisionColors <- c("East"="#56B4E9", "West"="#33FAFF", "North"="#F0E442",
"South"="#8B8D8D")
conferenceColors <- c("USL"="#000000", "AL"="#FC4E07")
```

Вы можете подобрать цвета и получить их шестнадцатеричные значения, воспользовавшись сайтом www.hexcolortool.com. Этот инструмент может быть использован и при подборе цветовых схем для визуальных элементов в Power BI.

Шаг 10. Динамически определите заголовок диаграммы

При помощи следующего кода определите переменную, в которой будет храниться заголовок нашей диаграммы:

```
chartTitle <- paste("Runningback Quad Chart for",
                    reportYear,
                    sep = " ")
```

Здесь мы просто объединяем строку «Runningback Quad Chart for» с выбранным для визуализации годом через пробел.

Шаг 11. Создайте набор данных для диаграммы

Набор данных, который был передан в Power BI, еще не окончательно готов для визуализации. Нам необходимо снабдить точки данных метками, но при этом если использовать имя и фамилию игрока, метки окажутся чересчур длинными, так что одновременно нам нужно найти способ как-то сократить их. Неплохо было бы взять лишь первую букву из имени игрока и дополнить ее фамилией. Это можно легко сделать при помощи следующего кода:

```
chartData <-
  dataset %>%
  mutate(Name = paste0(str_sub(FN,1,1),". ", LN))
```

Здесь мы к датафрейму *dataset* добавляем столбец с именем *Name*, базирующийся на трех объединенных при помощи функции *paste0()* строках. Первая строка образуется вызовом функции *str_sub()*, извлекающей из содержимого столбца *FN* первую букву. Далее идет точка и полная фамилия, хранящаяся в столбце *LN*. Таким образом, если в поле *FN* стоит *John*, а в поле *LN* – *Doe*, результатом операции будет строка *J. Doe*.

Шаг 12. Создайте диаграмму при помощи функции ggplot

Пузырьковая диаграмма, которую мы собираемся создать, будет охватывать шесть измерений:

- координата *x* будет отражать вес атлета;
- координата *y* – его рост;
- размер пузырька будет характеризовать количество преодоленных ярдов;
- внутренняя заливка пузырька – дивизион футболиста;
- цвет границы пузырька – конференцию;
- метка – сокращенное имя спортсмена.

Вы можете установить эти измерения при помощи функции *aes()* в рамках функции *ggplot()*, как показано ниже:

```
p <- ggplot(
  chartData,
  aes(
    x = Weight, y = Height, size = `Rushing Yards`,
    color = Conference, fill = Division, stroke = 2,
    label = Name
  )
)
```

Шаг 13. Добавьте слой для пузырьковой диаграммы при помощи геометрии geom_point

В вашем распоряжении есть 26 разных форм, которые вы можете использовать при построении пузырьковой диаграммы посредством функции *geom_point()*. На рис. 2.11 представлены все доступные фигуры.

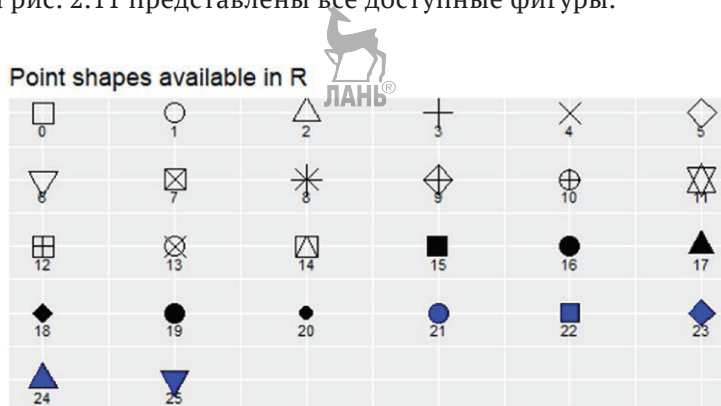


Рис. 2.11 ❖ Фигуры, доступные в функции *geom_point()*. Визуализация построена при помощи пакета *ggplot*

В представленном списке есть три формы, которые могут быть использованы при создании пузырьковых диаграмм, – это 1, 16 и 21. При этом только в последней форме можно отдельно устанавливать цвет заливки пузырька и цвет границы. В форме под номером 1 допустимо устанавливать только цвет границы, а в 16 – лишь цвет заливки. Ниже приведен код добавления слоя с геометрией `geom_point()` и типом фигуры, равным 21:

```
p <- ggplot(
  chartData,
  aes(
    x = Weight, y = Height, size = `Rushing Yards`,
    color = Conference, fill = Division, stroke = 2,
    label = Name
  )
) +
geom_point(shape = 21)
```

Результат показан на рис. 2.12.

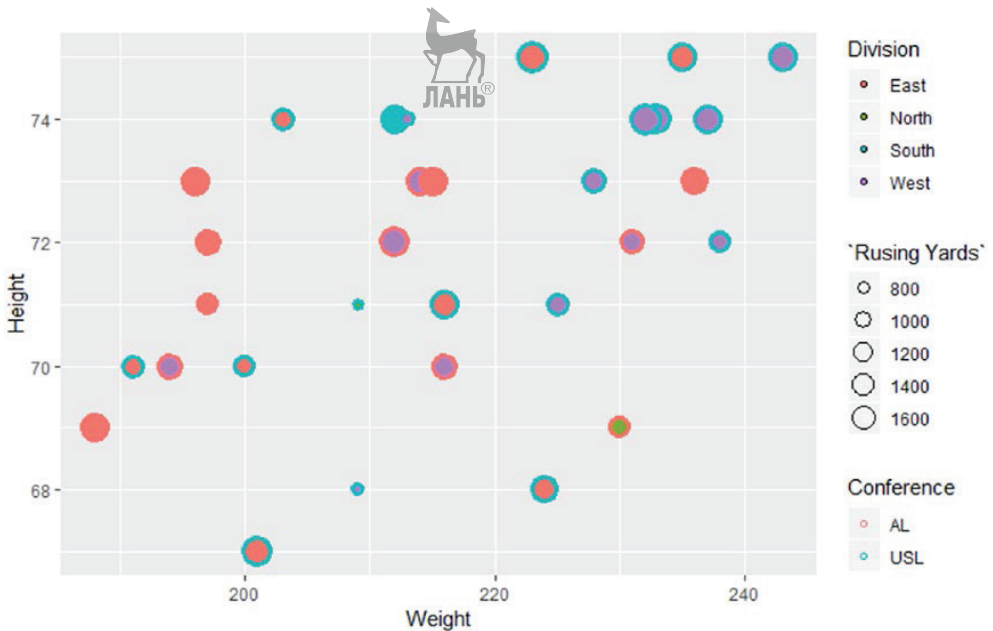


Рис. 2.12 ❖ Исходный вариант пузырьковой диаграммы с фигурой номер 21

Итак, мы построили базовую пузырьковую диаграмму. Обратите внимание, что геометрия `geom_point()` наследует эстетики у функции `ggplot()`, так что явно объявлять их нет никакой необходимости. Теперь нужно навести общую косметику, чтобы диаграмма смотрелась прилично.

Шаг 14. Добавьте метки на диаграмму

У нас есть пузырьки, разбросанные по диаграмме, но мы не знаем, какие из них каким спортсменам соответствуют. Можно добавить текст на визуализацию при помощи геометрии `geom_text_repel()` из пакета `ggrepel`. Геометрия `geom_text_repel()` при отрисовке делает все возможное, чтобы не возникало перекрытия и наложения меток. Ниже представлен код добавления слоя с геометрией `geom_text_repel()`:

```
p <- ggplot(
  chartData,
  aes(
    x = Weight, y = Height, size = `Rushing Yards`,
    color = Conference, fill = Division, stroke = 2,
    label = Name
  )
) +
geom_point(shape = 21) +
geom_text_repel(size = 5)
```

Результат добавления слоя при помощи функции `geom_text_repel()` показан на рис. 2.13.

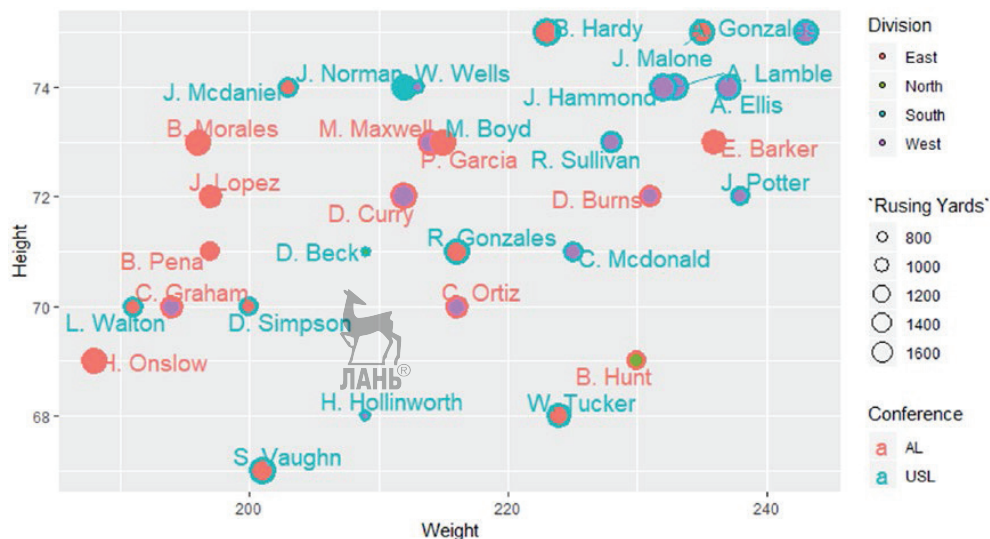


Рис. 2.13 ❖ Пузырьковая диаграмма с метками

Заметьте, что мы установили размер шрифта меток при помощи аргумента `size` функции `geom_text_repel`. Если этот аргумент не указывать, размер шрифта меток будет варьироваться в зависимости от размера пузырька.

Шаг 15. Измените цвет границ и заливок пузырьков на диаграмме

Следующим шагом необходимо изменить цвета границ и заливки точек данных. Для цвета границ мы используем инициализированную ранее переменную *conferenceColors*, в которой хранится именованный символьный вектор с цветовой схемой для конференций. Это можно сделать, присвоив переменную *conferenceColors* аргументу *values* функции *scale_color_manual()*. Также необходимо задать цвета заливки пузырьков при помощи созданной переменной *divisionColors*. Для этого присвоим переменную *divisionColors* аргументу *values* функции *scale_fill_manual()*. Фрагмент кода, в котором это сделано, представлен ниже:

```
p <- ggplot(
  chartData,
  aes(
    x = Weight, y = Height, size = `Rushing Yards`,
    color = Conference, fill = Division, stroke = 2,
    label = Name
  )
) +
  geom_point(shape = 21) +
  geom_text_repel(size = 5) +
  scale_color_manual(values = conferenceColors) +
  scale_fill_manual(values = divisionColors)
```

Результат показан на рис. 2.14.

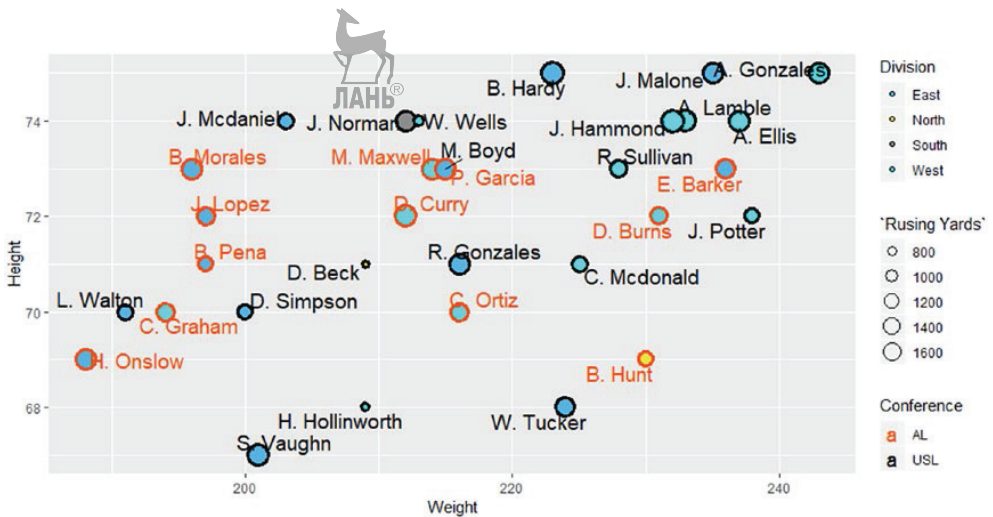


Рис. 2.14 ❖ Пузырьковая диаграмма с цветовым наполнением

Шаг 16. Создайте заголовок диаграммы

Теперь нужно передать созданную ранее переменную *chartTitle* в функцию *ggtitle()* для создания заголовка диаграммы, как показано в последней строке следующего фрагмента кода:

```
p <- ggplot(
  chartData,
  aes(
    x = Weight, y = Height, size = `Rushing Yards`,
    color = Conference, fill = Division, stroke = 2,
    label = Name
  )
) +
geom_point(shape = 21) +
geom_text_repel(size = 5) +
scale_color_manual(values = conferenceColors) +
scale_fill_manual(values = divisionColors) +
ggtitle(chartTitle)
```



Шаг 17. Задайте тему

Осталось установить тему для нашей визуализации, и мы выберем тему *theme_few()* из пакета *ggthemes*:

```
p <- ggplot(
  chartData,
  aes(
    x = Weight, y = Height, size = `Rushing Yards`,
    color = Conference, fill = Division, stroke = 2,
    label = Name
  )
) +
geom_point(shape = 21) +
geom_text_repel(size = 5) +
scale_color_manual(values = conferenceColors) +
scale_fill_manual(values = divisionColors) +
ggtitle(chartTitle) +
theme_few()
```



Шаг 18. Перенесите код в Power BI

Мы написали полноценный скрипт на языке R, удовлетворяющий всем нашим требованиям. Осталось лишь перенести его в полном объеме во встроенный редактор скриптов R в Power BI. При этом визуальный элемент должен корректно реагировать на выбор фильтра по полю **Year**.

```
library(tidyverse)
library(ggrepel)
```

```

library(ggthemes)

currentColumns <- sort(colnames(dataset))
requiredColumns <-
  c("Conference", "Division", "FN", "Height", "ID", "LN", "Rushing Yards", "Weight", "Year")
columnTest <- isTRUE(all.equal(currentColumns, requiredColumns))
reportYear <- unique(dataset$Year)

if(length(reportYear) == 1 & columnTest) {
  chartData <-
    dataset %>%
    mutate(Name = paste0(substring(FN,1,1),". ", LN))

  divisionColors <-
    c("East"="#56B4E9", "West"="#009E73", "North"="#F0E442", "South"="#0072B2")
  conferenceColors <- c("USL"="#000000", "AL"="#FC4E07")

  chartTitle <-
    paste("Runningback Quad Chart for", reportYear, sep = " ")

  p <- ggplot(
    chartData,
    aes(
      x = Weight, y = Height, size = `Rushing Yards`,
      color = Conference, fill = Division, stroke = 2,
      label = Name
    )
  ) +
  geom_point(shape = 21) +
  geom_text_repel(size = 5) +
  scale_color_manual(values = conferenceColors) +
  scale_fill_manual(values = divisionColors) +
  ggtitle(chartTitle) +
  theme_few()

  p
} else{
  plot.new()
  title("The data supplied did not meet the requirements of the chart.")
}

```

Визуализация прогнозирования

Техники *прогнозирования* (forecasting) результатов деловой деятельности применяются в компаниях уже много лет. Большинство инструментов бизнес-аналитики обладают встроенными возможностями включения в дашборды прогнозных визуализаций, но при этом подобные визуализации обладают весьма ограниченным функционалом. Эти ограничения в первую очередь связаны с тем, что пользователь видит исключительно прогноз показателей, тогда как важные факторы, влияющие на него, включая тенденции и сезон-

ность, обычно остаются вне поля его зрения. Тенденции демонстрируют долгосрочный тренд роста или падения показателей, а сезонность описывает характер изменчивости данных в зависимости от времени года. К счастью, аналитическим сообществом Facebook был разработан специальный пакет для R под названием *Prophet*, позволяющий не просто строить прогноз, но и отдельно визуализировать каждый из входящих в него компонентов.

В данной главе мы рассмотрим пример из документации библиотеки *Prophet*. Он будет посвящен прогнозированию количества просмотров страницы в Википедии, посвященной бывшему игроку NFL Пейтону Мэннингу. После этого мы построим интерактивную визуализацию, в которой можно будет переключаться между двумя действующими квотербеками NFL Ламаром Джексон (Lamar Jackson) и Дешоном Уотсоном (Deshaun Watson). Визуализация будет не только включать в себя прогноз анализируемых показателей, но и показывать тенденции прогнозирования, а также учитывать недельную и годовую сезонность. На рис. 2.15 проиллюстрирована визуализация, которую мы будем строить.

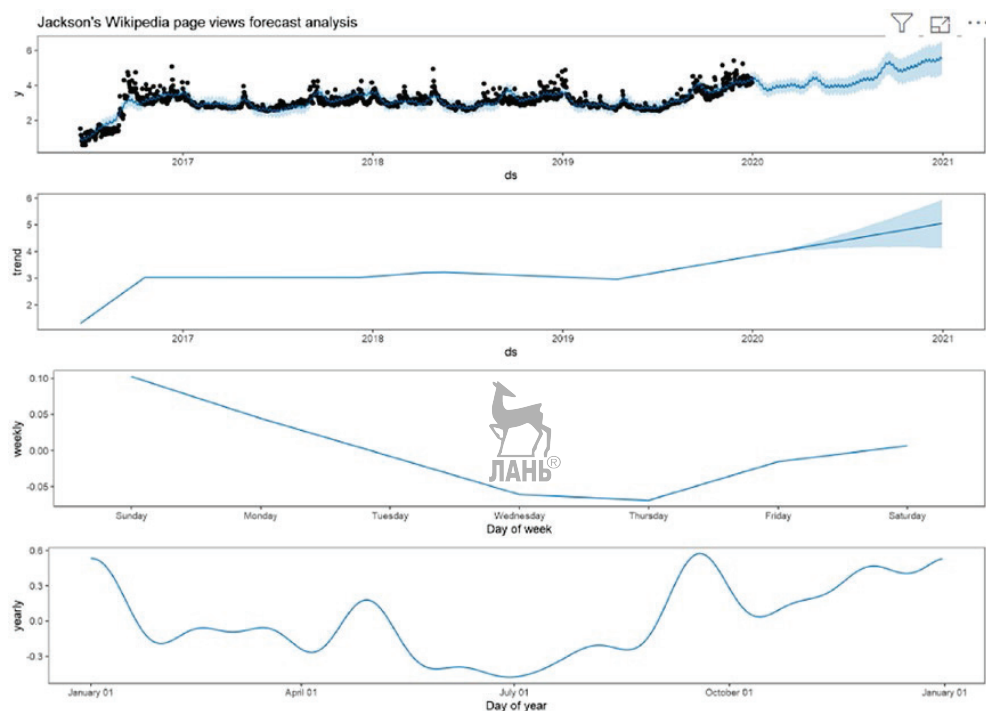


Рис. 2.15 ❖ Визуализация прогноза при помощи пакета *Prophet* и библиотеки *ggplot2*

Давайте рассмотрим пошаговую инструкцию создания такой визуализации.

Шаг 1. Загрузите исходные данные

Первое, что вам необходимо сделать, – это получить информацию о просмотрах страниц в Википедии, посвященных Ламару Джексону и Дешону Уотсону. Ниже приведена подробная инструкция для извлечения этих данных. Обратите внимание, что вам нужно будет дважды выполнить эту инструкцию – для обоих игроков.

1. Перейдите по адресу https://en.wikipedia.org/wiki/Wikipedia:Web_statistics_tool, где располагается инструмент веб-статистики Википедии (Wikipedia Web Statistics Tool).
2. В разделе *Current tools* выберите пункт *toolforge:pageviews*.
3. Очистите содержимое фильтра по умолчанию соответствующей кнопкой, чтобы ввести интересующие вас страницы. При первом анализе введите фамилию первого игрока (Lamar Jackson), при втором – второго (Deshaun Watson).
4. Щелкните на поле *Даты* (Dates) и выберите пункт *Настраиваемый диапазон* (Custom range).
5. Выберите диапазон дат, включающий как минимум два года. Это необходимо, чтобы у библиотеки *Prophet* было достаточно информации для определения годовой сезонности. Я выбрал период с 1 января 2018 года по 31 декабря 2019-го.
6. Нажмите на кнопку *Загрузить* (Download), располагающуюся ниже поля с выбором страниц, и выберите пункт *CSV*. В результате выбранные вами данные в фильтрах будут загружены на ваш компьютер в виде CSV-файла. Имя файла по умолчанию мало что может сказать вам о его содержимом, так что вам лучше сразу его переименовать. Я решил сопроводить имена загруженных файлов префиксами, соответствующими обоим игрокам: *lj* для Ламара Джексона и *dw* – для Дешона Уотсона.
7. Теперь нам необходимо загрузить полученные данные в Power BI, выполнить некоторые преобразования для облегчения визуализации, после чего объединить в единый набор данных. Для этого мы используем инструмент *Power Query*. Итоговый набор данных должен содержать поля, перечисленные в табл. 2.2, с соответствующими им типами данных. Шаги с преобразованиями, выполненными при помощи *Power Query*, можно найти в файле *pbix* в репозитории к данной главе.

Таблица 2.2. Поля и типы данных в итоговом наборе

Имя столбца	Тип данных
Date	Text
Page Views	Whole number
Quarterback	Text
Page Views (log10)	Decimal number

Шаг 2. Создайте срез по квотербекам на панели фильтров

Перенесите поле *Quarterback* из полученной таблицы в правую верхнюю часть рабочей области, после чего на панели визуализаций выберите элемент со *срезом* (Slicer). Это один из стандартных элементов визуализации в Power BI, кнопка которого содержит изображение воронки.

Шаг 3. Настройте визуальный элемент R в Power BI

Перейдите на панель визуализаций и перенесите в рабочую область знакомый уже вам визуальный элемент R. Измените его размеры на желаемые. Перенесите поля *Date*, *Page Views*, *Page Views (Log10)* и *Quarterback* из таблицы *WikipediaPageViews* на панель **Поля** (Fields). Если эта панель не отображается, выделите элемент визуализации R, и она появится.

Шаг 4. Экспортируйте данные в R Studio для дальнейшей разработки

Нажмите на диагональную стрелку в правой части открывшегося в нижней части экрана редактора R-скриптов. Это позволит передать датафрейм вашего визуального элемента вместе с шаблоном исходного кода в редактор *R Studio*. Если вам необходимо больше информации, обратитесь к четвертому шагу в инструкции к построению диаграммы с аннотацией.

Шаг 5. Загрузите необходимые пакеты

Для создания диаграммы прогнозирования вам потребуется загрузить следующие пакеты:

```
library(tidyverse)
library(prophet)
library(ggthemes)
library(gridExtra)
library(lubridate)
```

Пакет *dplyr* из коллекции *tidyverse* понадобится нам, как и раньше, для выполнения необходимых преобразований данных. Пакет *ggplot2* мы используем для построения графической визуализации. Пакет *prophet* позволит нам реализовать логику прогноза для нашей визуализации, при помощи библиотеки *gridExtra* мы разместим несколько диаграмм на одном визуальном элементе, а пакет *lubridate* используем для манипуляции с датами.

Шаг 6. Создайте переменные для проверки данных

Код с созданием переменных, необходимых для проверки целостности данных, приведен ниже:

```
currentColumns <- sort(colnames(dataset))
requiredColumns <- c("Date", "Page Views", "Page Views (Log10)",
"Quarterback")
columnTest <- isTRUE(all.equal(currentColumns, requiredColumns))
qb <- unique(dataset$Quarterback)
```

В переменной *currentColumns* в виде символьного вектора хранится список названий столбцов набора данных, переданного в R из Power BI. При этом элементы вектора отсортированы в алфавитном порядке для облегчения сравнения. Переменная *requiredColumns* служит для хранения списка желаемых столбцов также в виде упорядоченного по алфавиту символьного вектора. Результат выполнения сравнения этих двух переменных мы будем хранить в переменной *columnTest*. В ней, как и в предыдущих примерах, используется функция *all.equal()* для проверки на равенство векторов *currentColumns* и *requiredColumns*. В случае их равенства будет возвращено значение TRUE, иначе – информация о том, в чем состоят различия между содержимым этих переменных. Нам необходимо получить булево значение, поэтому мы оборачиваем выражение в функцию *isTRUE()*, которая гарантированно вернет TRUE или FALSE в зависимости от результата выполнения операции сравнения. Последнее, что необходимо сделать на данном шаге, – получить уникальные значения из столбца *Quarterback* и сохранить их в переменной *qb*.

Шаг 7. Выполните проверку данных

Код, запускающий на выполнение проверку данных, представлен ниже:

```
if (length(qb) == 1 & columnTest) {
  <code to produce visual>
} else {
  plot.new()
  title("The data supplied did not meet the requirements of the chart.")
}
```

Нам необходимо, чтобы в наборе данных, переданном элементу визуализации, был выбран единственный игрок, а столбцы соответствовали заранее подготовленному списку. Проверку удовлетворения набора первому требованию мы осуществляем путем сравнения длины переменной *qb* с единицей. Если условие выполняется, значит, в переданном наборе содержится информация только об одном спортсмене, что нам и нужно. Также нам необходимо убедиться, что список столбцов в переданном в Power BI наборе данных

в точности соответствует желаемому перечню столбцов. Если переменная *columnTest* равна TRUE, значит, тест прошел успешно. И лишь при выполнении обоих условий R приступит к созданию визуализации с прогнозированием. В противном случае будет создана пустая диаграмма с заголовком в виде сообщения о том, что предоставленные данные не отвечают нашим требованиям.

Шаг 8. Создайте динамический заголовок для визуализации

В следующем фрагменте кода мы создадим переменную для хранения динамического заголовка визуализации на основании имени выбранного квортебека:

```
chartTitle <- paste0(qb, "'s Wikipedia page views forecast analysis")
```

Здесь мы просто объединяем имя выбранного спортсмена со строкой «'s Wikipedia page views forecast analysis» (Прогноз количества просмотров страницы в Википедии) без разделителей. Мы воспользовались функцией *paste0()*, которая отличается от функции *paste()* лишь тем, что объединяет содержимое строк без разделителей.

Шаг 9. Создайте набор данных, необходимый для составления прогноза

```
dfPageViews <-  
  dataset %>%  
  transmute(  
    Quarterback,  
    ds = ymd(Date),  
    y = `Page Views (Log10)`  
  )
```



Пакет *prophet* требует лишь двух переменных: *ds* и *y*. Первая из них отвечает за дату. При этом даты должны быть непрерывными и включать в себя как минимум двухлетний интервал. Переменная *y* представляет собой показатель, по которому вы собираетесь строить прогноз, – в нашем случае это логарифм по полю *Page Views*.

Примечание. Обратите внимание, что мы пользуемся именно логарифмом количества просмотров страницы, а не самим этим показателем в чистом виде. Это обычная практика в подобных ситуациях, когда данные обладают большими разбросами. Применение логарифма позволяет снизить дисперсию и облегчить процесс визуализации данных. Кроме того, логарифмирование широко применяется для *конструирования признаков* (feature engineering) при создании моделей анализа данных.

Шаг 10. Постройте прогноз

Всего трех строк кода, представленных ниже, достаточно, чтобы создать простой прогноз:

```
m <- prophet(dfPageViews)
future <- make_future_dataframe(
  m,
  periods = 365,
  freq = "day",
  include_history = TRUE
)
forecast <- predict(m, future)
```

В первой строке кода создается объект с моделью прогнозирования на базе датафрейма *dfPageViews*. Во второй строке создается новый датафрейм, в котором добавляется определенное количество периодов, переданное при помощи переменной *periods*, к списку дат в исходном датафрейме *dfPageViews*. Частота периодов задается при помощи аргумента *freq*, а если в качестве аргумента *include_history* передать значение TRUE, в датафрейм будут также включены прошлые даты. В третьей строке кода мы воспользовались функцией *predict()* для построения прогноза с использованием ранее созданной модели и нового датафрейма *future*.

Шаг 11. Постройте диаграмму

На данном этапе мы собираемся построить четыре диаграммы на одном элементе визуализации:

- график прогноза;
- график тенденций;
- график недельной сезонности;
- график годовой сезонности.

Код для создания всех четырех диаграмм показан ниже:

```
p1 <- plot(m, forecast) + ggtitle(chartTitle) + theme_few()
p <- prophet_plot_components(m, forecast)
p2 <- p[[1]] + theme_few()
p3 <- p[[2]] + theme_few()
p4 <- p[[3]] + theme_few()
p5 <- grid.arrange(p1, p2, p3, p4, nrow = 4)
p5
```

Итоговая визуализация была показана ранее на рис. 2.15. Такое представление с несколькими графиками бывает крайне полезным, поскольку вы не только видите сам прогноз, но можете тут же проанализировать все факторы, влияющие на него. На графике с трендом вы видите общую тенденцию прогнозируемого показателя, а на диаграммах с недельной и годовой се-

зонностью легко можно различить рост и падение спроса в зависимости от фактора времени.

Первое, что вам необходимо сделать, – это определить каждый график отдельно. Ниже приведено описание каждой переменной:

- переменная *p1* служит для визуализации актуального прогноза. Она создается при помощи функции *plot()* из пакета *prophet*, который в своей основе использует библиотеку *ggplot*. А раз так, вы можете спокойно применить к ней тему *theme_few()*;
- в переменной *p* хранятся графики компонентов, сгенерированные при помощи функции *prophet_plot_components()*. Эта функция возвращает список диаграмм *ggplot*, представляющих разные компоненты прогноза. В нашем случае это график тенденций, а также диаграммы недельной и годовой сезонности;
- переменные *p2*, *p3* и *p4* предназначены для хранения отдельных графиков компонентов, входящих в состав переменной *p*. Как мы уже говорили, переменная *p* представляет собой список графиков *ggplot*, и нам необходимо выделить его составляющие в виде подмножеств, что можно сделать при помощи двойных скобок (*[[*). Когда вы выделяете подмножество из списка в языке R при помощи одинарной квадратной скобки, вы получаете на выходе список с единственным элементом, а не объект нужного вам типа. Так что если написать просто *p[1]*, вы получите не график *ggplot*, а список из одного элемента, представляющего собой этот самый график. Если вам нужно вернуть сам элемент с желаемым типом данных, используйте для извлечения подмножества двойные квадратные скобки, как было показано в предыдущем фрагменте кода. Помните, что, поскольку все объекты извлекаются из списка в виде графиков *ggplot*, вы можете к каждому из них применить тему *theme_few()*.

Теперь, когда вы получили график прогноза в виде объекта *ggplot*, а также все компоненты в том же виде, вы можете вывести их все на одну визуализацию при помощи функции *grid.arrange()* из пакета *gridExtra*. Это происходит в шестой строке кода. Нам необходимо, чтобы графики располагались в столбик – один над другим. Этого можно добиться, передав в качестве аргумента *nrow* значение 4. Именно при помощи этого аргумента осуществляется контроль за тем, как именно графики будут размещаться на элементе визуализации. К примеру, если передать аргументу *nrow* значение 2, графики будут выведены по два в одной строке.

Шаг 12. Перенесите код в Power BI

Полный скрипт для создания визуализации прогнозирования на языке R представлен ниже. Скопируйте код и вставьте его во встроенный редактор R-скриптов в Power BI. В результате визуализация должна реагировать на выбор квотербека в добавленном ранее фильтре.

```
library(tidyverse)
library(prophet)
```

```
library(ggthemes)
library(gridExtra)
library(lubridate)
library(rlang)

# Мне понадобилось загрузить пакет rlang. Посмотрите его.

currentColumns <- sort(colnames(dataset))
requiredColumns <- c("Date", "Page Views", "Page Views (Log10)", "Quarterback")
columnTest <- isTRUE(all.equal(currentColumns, requiredColumns))
qb <- unique(dataset$Quarterback)

if (length(qb) == 1 & columnTest) {
  chartTitle <- paste0(qb, "'s Wikipedia page views forecast analysis")

  dfPageViews <-
    dataset %>%
    mutate(Date = ymd(Date)) %>%
    rename(ds = Date, y = `Page Views (Log10)`)

  m <- prophet(dfPageViews, yearly.seasonality=TRUE)

  future <- make_future_dataframe(m, periods = 365)

  forecast <- predict(m, future)

  p1 <- plot(m, forecast) + ggtitle(chartTitle) + theme_few()
  p <- prophet_plot_components(m, forecast)

  p2 <- p[[1]] + theme_few()
  p3 <- p[[2]] + theme_few()
  p4 <- p[[3]] + theme_few()
  p5 <- grid.arrange(p1, p2, p3, p4, nrow =4)
  p5
} else {
  plot.new()
  title("The data supplied did not meet the requirements of the chart.")
}
```

ЛИНЕЙНАЯ ДИАГРАММА С ЗАТЕНЕНИЕМ

Линейная диаграмма является одним из наиболее популярных способов визуализировать данные графическим способом. Ее очень удобно применять тогда, когда нужно отобразить тенденции во времени. И в связи с такой популярностью линейная диаграмма, конечно, по умолчанию входит в базовый пакет любой BI-системы.

В данном разделе мы рассмотрим процесс построения линейной диаграммы в Power BI при помощи языка R. Но это будет не совсем обычный график. Он будет включать в себя функционал, который на момент написания книги в стандартном пакете Power BI не предусмотрен. Идею такого визуального

элемента я почерпнул из книги Хэдли Уикхэма «Ggplot2: Elegant Graphics for Data Analysis» («Ggplot2: Изящные графики в анализе данных»). В своей визуализации автор использовал технику фонового затенения, чтобы показать, представитель какой партии – Демократической или Республиканской – был у власти в тот или иной момент времени. Мы с вами сделаем нечто подобное в Power BI с использованием языка R, даже с некоторыми улучшениями. Визуализация, к которой мы будем стремиться, показана на рис. 2.16.

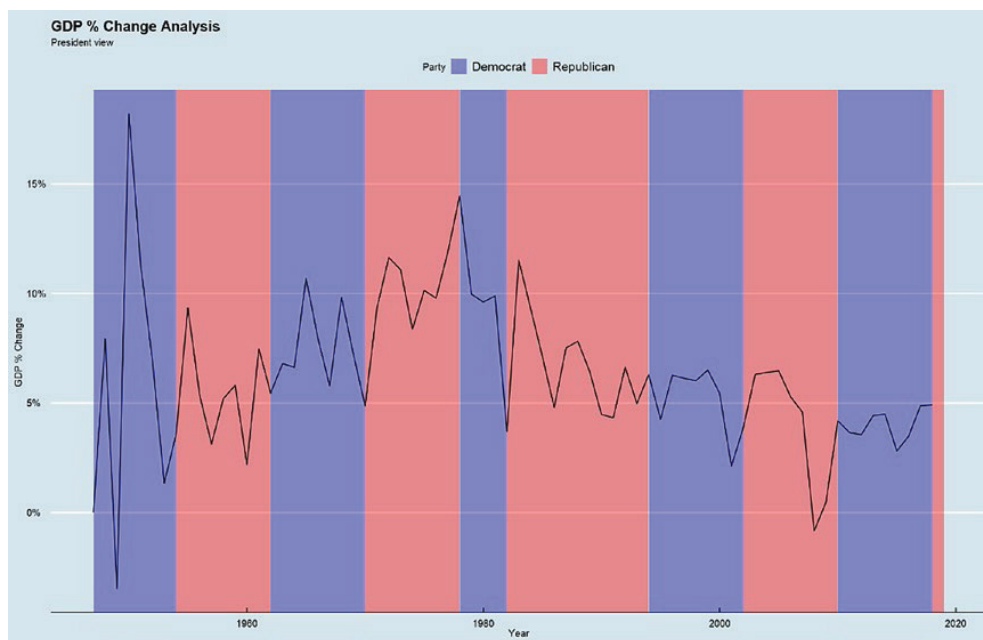


Рис. 2.16 ❖ Линейная диаграмма, показывающая изменение ВВП на фоне партии власти

На этой диаграмме показано изменение *валового внутреннего продукта* (GDP) в процентах с 1947 по 2018 год. Особенностью этого графика является то, что на фоне вы видите зоны затенения, цвет которых зависит от того, какая в тот или иной момент времени партия была у власти. Но мы пойдем чуть дальше и воспользуемся интерактивными возможностями Power BI, чтобы улучшить нашу визуализацию. Дадим пользователю возможность выбирать между партией власти, партиями, представляющими большинство в верхней (сенате) и нижней палате конгресса, а также партией, имеющей абсолютное большинство. В дополнение к этому пользователь сможет переключаться между *текущим значением ВВП* (Actual GDP) и его *процентным изменением* (% Change in GDP). На рис. 2.17 представлен график по текущему ВВП для абсолютного большинства.

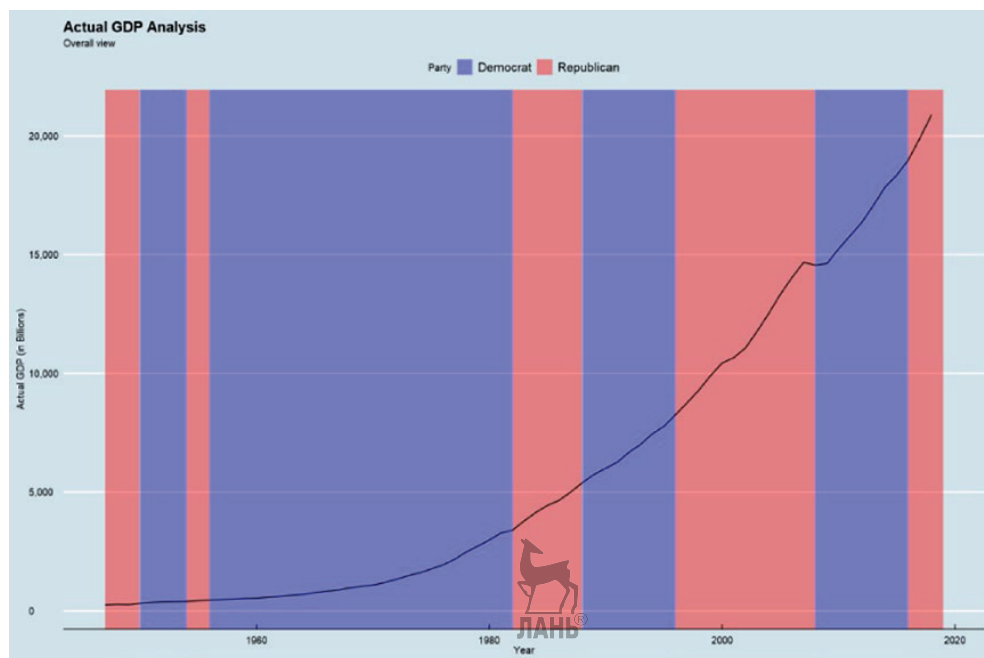


Рис. 2.17 ❖ Линейная диаграмма, показывающая текущий ВВП с фоном по партиям, представляющим абсолютное большинство в тот или иной период времени

Теперь, когда вы знаете, к чему мы будем двигаться, давайте пройдем по шагам и реализуем нашу задумку в Power BI.

Шаг 1. Загрузите исходные данные

1. Откройте сайт <https://fred.stlouisfed.org/series/GDP>, на котором хранится архивная информация о валовом внутреннем продукте. Нажмите на кнопку **EDIT GRAPH**, поменяйте значение в выпадающем списке *Modify frequency* на *Annual*, а *Aggregation method* – на *End of Period*. Нажмите на крестик в верхнем правом углу формы, чтобы закрыть ее и применить изменения.
2. Нажмите на кнопку **DOWNLOAD** и выберите формат файла для загрузки CSV. На момент написания книги это приводило к скачиванию на локальный компьютер файла с именем *GDP.csv*.
3. Откройте репозиторий данной главы и скачайте файл *PoliticalInfoWithGDP.xlsx*. В этой рабочей книге содержатся исходные данные и запросы Power Query, которые использовались при создании набора данных *PoliticalInfoWithGDP* для нашего примера. *PoliticalInfoWithGDP* представляет собой набор данных с исходной информацией для элемента

визуализации, который мы вместе построим, читая эту главу. Я сделал за вас большую часть работы, но вы можете изучить загруженную рабочую книгу, чтобы понять, как именно исходные данные были преобразованы в целевой набор данных *PoliticalInfoWithGDP*. Ниже приведено подробное описание всех запросов, которые были созданы посредством Power Query в рабочей книге *PoliticalInfoWithGDP.xlsx*:

- *GDP*: в этом запросе мы загружаем данные о ВВП, полученные на шаге 2, и добавляем к набору данных столбец % *GDP Change*;
- *Presidents*: в этом запросе строится набор данных со всеми президентами США за анализируемый период времени вместе со столбцом, в котором указана их политическая принадлежность;
- *Political Info*: этот запрос подключается к странице Википедии по адресу https://en.wikipedia.org/wiki/Party_divisions_of_United_States_Congresses. На данной странице содержится вся необходимая информация о партиях большинства в верхней и нижней палатах конгресса, а также о действующих в разные периоды времени президентах США. При помощи Power Query мы преобразуем полученные данные в вид, пригодный для анализа и визуализации. Вы можете рассмотреть все шаги в Power Query, чтобы понять, как это реализовано;

Примечание. Power Query представляет собой инструмент для преобразования данных, встроенный в продукты *Microsoft*, такие как *Microsoft Excel*, *Power BI*, *Power BI Dataflows* и *SQL Server Analysis Services*. Интерфейс Power Query значительно снижает количество ручного труда при выполнении сложных операций над данными. Разработчикам на R и Python, незнакомым с экосистемой *Microsoft*, я бы настоятельно рекомендовал присмотреться к этому прекрасному инструменту.

- *PoliticalInfoWithGDP*: объединяет полученные ранее наборы данных и приводит их к виду, пригодному для визуализации.
4. Как мы уже упоминали ранее, *PoliticalInfoWithGDP* представляет собой набор данных, который и будет использоваться для визуализации. Он был создан при помощи Power Query и сохранен в виде файла *PoliticalInfoWithGDP.csv*. Этот файл вы можете найти в репозитории к данной главе.

Шаг 2. Загрузите данные в Power BI

Чтобы загрузить файл *PoliticalInfoWithGDP.csv* в Power BI, необходимо на вкладке **Главная** (Home) нажать на выпадающую кнопку **Получить данные** (GetData), выбрать пункт **Текстовый или CSV-файл** (Text/CSV) и указать файл, который вы ранее сохранили. Нажмите на кнопку **Преобразовать данные** (Transform Data), чтобы перейти в редактор Power Query. Убедитесь, что поля обладают типами данных, указанными в табл. 2.3.

Таблица 2.3. Столбцы и типы данных

Столбец	Тип данных
Index	Whole Number
Year	Whole Number
GDP	Decimal Number
% GDP Change	Decimal Number
Senate Majority	Text
House Majority	Text
President's Majority	Text
Overall Majority	Text



Если типы данных в столбцах не соответствуют приведенным в таблице, измените их соответствующим образом. Проще всего щелкнуть правой кнопкой мыши по столбцу, раскрыть выпадающее меню **Тип изменения** (Change Type) и выбрать нужный тип. После окончания работы с типами данных нажмите на кнопку **Закрыть и применить** (Close & Apply), чтобы загрузить данные в модель Power BI.

Перейдите к таблице и убедитесь, что поведением по умолчанию для полей *Year* и *Index* стоит **Не суммировать** (Don't Summarize). Чтобы сделать это, выделите поле на панели **Fields** (Поля), перейдите на вкладку **Моделирование** (Modeling) и в группе **Свойства** (Properties) выберите вариант **Не суммировать** (Don't Summarize).

Шаг 3. Создайте срезы в отчете

Наш отчет будет содержать два среза. В первом пользователь может выбрать область изучения политических взглядов (нижняя палата конгресса (House), верхняя палата (Senate), президент (President) или суммарный показатель (Overall)), а во втором – тип показателя (текущий ВВП (Actual GDP) или процентное изменение (GDP % Change)). Этот шаг был выполнен за вас в шаблоне *pbix*, который вы также можете найти в репозитории к данному примеру.

Шаг 4. Настройте визуальный элемент R в Power BI

Перейдите на *панель визуализаций* (Visualization pane) и перенесите визуальный элемент R в рабочую область. Измените размер перенесенного элемента под свои требования. Добавьте поля *Year*, *GetPoliticalLevel*, *GetPoliticalLevelName*, *GetGDPStat* и *GetGDPStatName* на панель **Поля** (Fields), расположенную под панелью визуализаций. Иногда Microsoft Power BI по умолчанию применяет функцию агрегации к числовым полям. Проверьте, не попытался ли он сделать это с полем *Year*. Если функция агрегации стоит, удалите ее, нажав на стрелку вниз рядом с полем и выбрав пункт **Не суммировать** (Don't Summarize).

Шаг 5. Экпортируйте данные в R Studio для дальнейшей разработки

Нажмите на диагональную стрелку в правой части встроенного в Power BI редактора скриптов R. Это приведет к экспорту данных и базового кода на языке R в *R Studio*. Обратитесь к четвертому шагу инструкции по созданию диаграммы с аннотацией за дополнительной информацией.

Шаг 6. Загрузите необходимые пакеты

Для этого примера вам потребуется загрузить следующие пакеты:

```
library(tidyverse)
library(ggthemes)
library(scales)
```

Пакет *dplyr* из коллекции *tidyverse* понадобится нам для выполнения необходимых преобразований данных. Пакет *ggplot2* из той же коллекции мы используем для построения графической визуализации. Из библиотеки *ggthemes* мы возьмем тему для визуализации в стиле журнала *The Economist*, а функции из пакета *scales* используем для форматирования чисел.

Шаг 7. Создайте переменные для проверки данных

Ниже представлен код для проверки данных:

```
currentColumns <- sort(colnames(dataset))
requiredColumns <- c("GetGDPStat", "GetGDPStatName", "GetPoliticalLevel",
"GetPoliticalLevelName", "Year")
columnTest <- isTRUE(all.equal(currentColumns, requiredColumns))
politicalLevelName <- unique(dataset$GetPoliticalLevelName)
gdpStatName <- unique(dataset$GetGDPStatName)
```

Здесь мы выполняем сразу три проверки:

- проверку того, что переданный визуализации набор данных содержит нужные нам поля;
- проверку того, что выбрана единственная область анализа в срезе;
- проверку того, что выбран единственный тип отображения ВВП в срезе.

Давайте сначала внимательно посмотрим на код первой проверки:

```
currentColumns <- sort(colnames(dataset))
requiredColumns <- c("GetGDPStat", "GetGDPStatName", "GetPoliticalLevel",
"GetPoliticalLevelName", "Year")
columnTest <- isTRUE(all.equal(currentColumns, requiredColumns))
```

В переменной *currentColumns* хранится список имен столбцов набора данных, переданного в R из Power BI, в виде символьного вектора. При этом элементы отсортированы в алфавитном порядке для облегчения сравнения. Переменная *requiredColumns* служит для хранения списка желаемых столбцов также в виде упорядоченного по алфавиту вектора. Результат выполнения сравнения этих двух переменных мы будем хранить в переменной *columnTest*. В ней мы использовали функцию *all.equal()* для проверки на равенство векторов *currentColumns* и *requiredColumns*. В случае их равенства будет возвращено значение TRUE, иначе – информация о том, в чем состоят различия между содержимым этих переменных. Нам нужно получить булево значение, поэтому мы оборачиваем выражение в функцию *isTRUE()*, которая гарантированно вернет TRUE или FALSE в зависимости от результата выполнения операции сравнения. Последнее, что необходимо сделать на данном шаге, – это получить уникальные значения из столбцов *GetPoliticalLevelName* и *GetGDPStatName* и сохранить их в соответствующих переменных при помощи следующего кода:

```
politicalLevelName <- unique(dataset$GetPoliticalLevelName)
gdpStatName <- unique(dataset$GetGDPStatName)
```

В идеале в каждой переменной должно оказаться ровно по одному значению, и эту проверку мы выполним на следующем шаге.

Шаг 8. Выполните проверку данных

Используем следующий шаблон для проверки данных:

```
if(length(politicalLevelName) == 1 &
    length(gdpStatName) == 1 & columnTest) {
  <code to produce visual>
} else {
  plot.new()
  title("The data supplied did not meet the requirements of the chart.")
}
```

Если в нашем наборе данных представлена только одна область для анализа (*politicalLevelName*) и один тип ВВП (*gdpStatName*), а также проверка идентичности столбцов прошла успешно, создаем наполнение нашего элемента визуализации. В противном случае создаем пустую диаграмму с сообщением о том, что исходные данные не отвечают требованиям визуализации.

Шаг 9. Создайте новый датафрейм на основании датафрейма dataset

Датафрейм с именем *dataset*, переданный в R из Power BI, изменять не принято. Вместо этого хорошим тоном считается создать его копию, которую уже можно модифицировать по своему усмотрению. Делается это легко и просто:

```
dfPI <- dataset
```

Шаг 10. Создайте переменные для динамических составляющих диаграммы

Ниже представлен код для создания нужных нам вспомогательных переменных:

```
politicalLevelName <- unique(dfPI$GetPoliticalLevelName)
gdpStatName <- unique(dfPI$GetGDPStatName)
yAxisName <-
  paste(
    gdpStatName,
    ifelse(gdpStatName == "Actual GDP", "(in Billions)", ""),
    sep = " "
  )
chartTitle <- paste(gdpStatName, "Analysis", sep = " ")
chartSubtitle <- paste(politicalLevelName, "view", sep = " ")
```

Здесь мы динамически собираем название метки для оси y, а также заголовков и подзаголовков для нашей диаграммы. В первых двух строках кода мы получаем информацию о выбранных значениях в наших срезах и сохраняем их в переменных *politicalLevelName* и *gdpStatName* для дальнейшего использования. В следующих шести строках отформатированного кода создается метка для оси y. Здесь мы используем уже знакомую нам функцию *paste()* для объединения значения переменной *gdpStatName* со строкой «in billions» (в миллиардах), если мы собираемся выводить на диаграмму актуальные значения ВВП, то есть если в переменной *gdpStatName* находится строка «Actual GDP». Результат присваивается переменной *yAxisName*. Далее мы динамически собираем заголовок диаграммы из значения переменной *gdpStatName* и слова «Analysis» с пробелом в качестве разделителя. В заключительной строке кода мы определим, каким будем подзаголовок диаграммы путем объединения значения переменной *politicalLevelName* со словом «view» с разделителем пробелом.

Шаг 11. Создайте наборы данных, необходимые для наложения тени

На данном шаге мы преобразуем наш датафрейм *dfPI* в набор данных, пригодный для отрисовки затенения на нашей визуализации. Посмотрите на датафрейм, показанный на рис. 2.18.

Используя этот датафрейм, нам необходимо создать набор данных с датами начала и конца периодов нахождения партий у власти. Иными словами, нужно преобразовать датафрейм, который вы видели ранее, в датафрейм, представленный на рис. 2.19.

Year	GetPoliticalLevel	GetPoliticalLevelName	GetGDPStat	GetGDPStatName
1947	Republican	Overall	259.745	Actual GDP
1948	Republican	Overall	280.366	Actual GDP
1949	Republican	Overall	270.627	Actual GDP
1950	Democrat	Overall	319.945	Actual GDP
1951	Democrat	Overall	356.178	Actual GDP
1952	Democrat	Overall	380.812	Actual GDP
1953	Democrat	Overall	385.970	Actual GDP
1954	Republican	Overall	399.734	Actual GDP
1955	Republican	Overall	437.092	Actual GDP

Рис. 2.18 ❖ Фрагмент датафрейма dfPI

group_id	Party	start	end
1	Republican	1947	1949.99
2	Democrat	1950	1953.99
3	Republican	1954	1955.99
4	Democrat	1956	1981.99
5	Republican	1982	1987.99
6	Democrat	1988	1995.99
7	Republican	1996	2007.99
8	Democrat	2008	2015.99
9	Republican	2016	2018.99

Рис. 2.19 ❖ Требуемый формат датафрейма

Чтобы совершить такую трансформацию, необходимо иметь возможность однозначно идентифицировать каждый непрерывный период правления одной из двух партий. На данный момент такого идентификатора у нас нет, так что придется его создать. После этого мы сможем использовать традиционные методы агрегации для получения начала и конца периода правления той или иной партии. Давайте пройдемся по шагам, которые необходимо выполнить для добавления уникального идентификатора с именем *group_id*. Ниже приведен код для создания этого поля.

```
dfPI$GetPoliticalLevel <- as.character(dfPI$GetPoliticalLevel)
runs <- rle(dfPI$GetPoliticalLevel)
group_id <- rep(seq_along(runs$lengths), runs$lengths)
```

В первой строке кода выполняется необходимое преобразование типов. Типом данных по умолчанию для столбцов в датафрейме, содержащих строки, в R версии ниже 4.0 является *фактор* (factor).

Примечание. Во время написания данной книги служба Power BI не поддерживала R версии 4.0. Если на момент выполнения вами данного упражнения ничего в этом отношении не изменилось, вам также необходимо выполнить преобразование типов. В случае появления в службе Power BI поддержки R версии 4.0 и выше этот шаг можно пропустить.

Поскольку в столбце *GetPoliticalLevel* содержатся исключительно строковые данные, R дал этому столбцу тип данных фактор. Чтобы обработать информацию так, как нам нужно, необходимо привести поле *GetPoliticalLevel* к символьному (character) типу данных. Этот тип данных можно сравнить с типом *Text* в DAX. И это преобразование мы выполняем в первой строке кода. Следующее, что мы делаем, – это вызываем стандартную функцию *rle()*, позволяющую определить группы в столбце *GetPoliticalLevel*. Название этой функции расшифровывается как *run length encoding* (кодирование длин серий). Передавая на вход функции *rle()* столбец *GetPoliticalLevel*, мы получаем вывод, показанный на рис. 2.20.

```
Run Length Encoding
lengths: int [1:9] 3 4 2 26 6 8 12 8 3
values : chr [1:9] "Republican" "Democrat" "Republican" "Democrat" "Republican" "Democrat" "Republican"
```

Рис. 2.20 ❖ Вывод функции *rle()*

Функция *rle()* возвращает список, состоящий из двух векторов. Первый вектор с именем *lengths* в нашем случае содержит продолжительности нахождения у власти той или иной партии, а второй вектор – *values* – название партии, которая была у власти в каждый конкретный промежуток времени. Возвращенную функцией *rle()* информацию сохраним в переменной *runs*.

После этого содержимое переменной *runs* используется для создания вектора *group_id* следующим образом:

```
group_id <- rep(seq_along(runs$lengths), runs$lengths)
```

Первым делом здесь вызывается функция *seq_along()*, используемая для генерирования последовательности целых чисел, которую будет представлять переменная *group_id*. Поскольку в векторе *runs\$lengths* у нас содержится девять значений, вектор целых чисел также будет состоять из чисел от одного до девяти. Это выражение подается на вход функции *rep()* первым аргументом, а вторым передается вектор *runs\$lengths*. Поскольку оба аргумента содержат векторы одинаковой длины, функция *rep()* возьмет первый элемент, произведенный функцией *seq_along()* (в нашем случае это 1), и повторит его количество раз, представленное первым же элементом из второго параметра (у нас это 3). Таким образом, единица будет повторена три раза. Далее двойка будет повторена четыре раза, поскольку вторым элементом в векторе *runs\$lengths*

является именно четверка. Эта логика выполняется для всех элементов в векторах, и в результате мы получим вывод, показанный на рис. 2.21.



Рис. 2.21 ❖ Вывод переменной *group id*

Затем выполним следующий код с целью создания набора данных для визуализации:

```
dfShadeInfo <-
  cbind(dfPI, group_id) %>%
  transmute(group_id, Year, Party = GetPoliticalLevel) %>%
  group_by(group_id, Party) %>%
  summarize(start = min(Year), end = max(Year)+0.99) %>%
  ungroup()
```

Этот фрагмент кода начинается с присоединения вектора *group_id* в качестве нового столбца к датафрейму *dfPI* при помощи функции *cbind()*, где буква *c* означает *column* (столбец). Затем мы использовали знакомый уже вам оператор конвейера *%>%* для передачи результирующего датафрейма функции *transmute()* в качестве первого аргумента.

Примечание. Функция `transmute()` очень похожа на функцию `mutate()` из пакета `dplyr` в том отношении, что она служит для создания новых столбцов. Отличие состоит лишь в том, что при использовании `transmute()` итоговый датафрейм будет состоять только из столбцов, определенных в этой функции.

В результате вызова функции *transmute()* получится датафрейм, состоящий из трех столбцов: *group_id*, *Year* и *Party*. На рис. 2.22 показаны первые девять строк датафрейма для демонстрации работы функции *transmute()*.

group_id	Year	Party
1	1947	Republican
1	1948	Republican
1	1949	Republican
2	1950	Democrat
2	1951	Democrat
2	1952	Democrat
2	1953	Democrat
3	1954	Republican
3	1955	Republican

Рис. 2.22 ❖ Результат использования функции *transmute()*

Заметьте, что теперь у нас появился уникальный идентификатор каждого периода нахождения партии у власти. Таким образом, мы можем использовать традиционные функции агрегации для получения начала и конца каждого периода. Снова применим оператор конвейера для передачи датафрейма, полученного в результате вызова функции *transmute()*, в качестве первого аргумента на вход функции *group_by()*. Функция *group_by()*, как понятно из ее названия, сгруппирует переданный ей датафрейм по столбцам *group_id* и *Party*. Результат функции *group_by()* отправится в качестве первого параметра в функцию *summarize()*. Функция *summarize()* позволяет применять функции агрегирования к набору данных на основании ваших группировок. Таким образом, мы создали столбец *start*, отражающий начало периода нахождения партии у власти (для этого мы взяли минимальный номер года в группировке), и *end*, характеризующий конец периода нахождения партии у власти. При этом к значениям столбца *end* мы добавим 0.99, чтобы минимизировать разрывы между уходящими и приходящими режимами.

Шаг 12. Создайте наборы данных, необходимые для отрисовки графика

Следующий код мы используем для создания набора данных, на основании которого будет построена линейная диаграмма:

```
dfLineChartInfo <-
  dfPI %>%
    transmute(Year, GetGDPStat)
```

Наша цель – отобразить на линейной диаграмме выборочную информацию по ВВП за указанные годы. Для этого нам нужно оставить в наборе данных два столбца: *Year* и *GetGDPStat*. Создадим требуемый набор данных путем извлечения нужных нам столбцов из датафрейма *dfPI* при помощи функции *transmute()*.

Шаг 13. Создайте символьный вектор для хранения цветовой схемы затенения

Цвет Республиканской партии – красный, а Демократической – синий. Вы можете использовать эту цветовую схему в визуализации при помощи именованного символьного вектора *partyColors*. Позже мы используем этот вектор для затенения красным цветом периодов нахождения у власти Республиканской партии и синим – Демократической.

Шаг 14. Постройте диаграмму при помощи функции `ggplot()`

Предыдущие шаги были посвящены подготовке данных для анализа. Вы увидели, как легко можно средствами языка R привести к нужному для визуализации в Power BI виду исходную информацию.

На данном шаге мы воспользуемся уже знакомой вам функцией `ggplot()` для построения диаграммы:

```
p <- ggplot()
```

Вы могли заметить, что в данном случае мы вызвали функцию `ggplot()` без аргументов. Мы так сделали, потому что на нашей диаграмме будет два слоя – каждый со своим источником данных и своим набором эстетик, так что все эти характеристики лучше будет установить уже на уровне добавления слоев.



Шаг 15. Добавьте слой для создания затенения

Это можно сделать при помощи следующего кода:

```
geom_rect(
  data = dfShadeInfo,
  aes(xmin = start, xmax = end, fill = Party,
      ymin = -Inf, ymax = Inf, alpha = 0.4, color = NA)
)
```

Источником данных для данного слоя является датафрейм `dfShadeInfo`. На рис. 2.23 показано, как он выглядит при выборе фильтра *Overall*.



group_id	Party	start	end
1	Republican	1947	1949.99
2	Democrat	1950	1953.99
3	Republican	1954	1955.99
4	Democrat	1956	1981.99
5	Republican	1982	1987.99
6	Democrat	1988	1995.99
7	Republican	1996	2007.99
8	Democrat	2008	2015.99
9	Republican	2016	2018.99

Рис. 2.23 ❖ Датафрейм `dfShadeInfo`

На рис. 2.23 показаны девять периодов смены режима, и этот датафрейм мы используем для отрисовки фонового затенения с использованием цветов политических партий, находящихся у власти. Начало и конец периода нахождения партии у власти определяется установкой соответствия между столбцами *start* и *end* и аргументами *xmin* и *xmax* функции *aes()* соответственно. Верхняя граница области затенения определяется путем присвоения аргументу *ymax* значения *Inf*, а нижняя – посредством присвоения аргументу *ymin* значения *-Inf*. Использование значений бесконечности для определений верхней и нижней границ областей затенения гарантирует полную заливку фона диаграммы по вертикали. При этом затенение выполняется для всех периодов, что приводит к закрашиванию всего фонового пространства диаграммы, что видно по рис. 2.24.

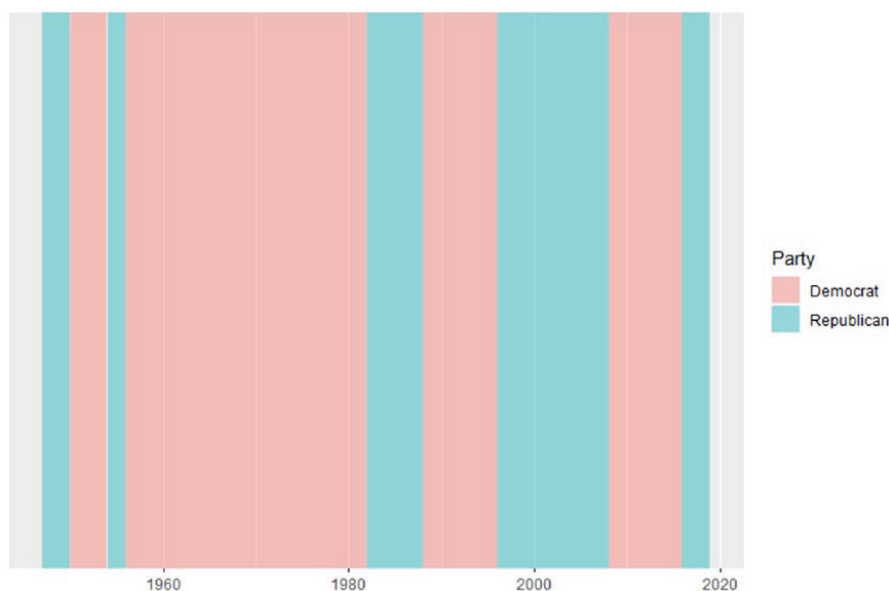


Рис. 2.24 ❖ Фоновое затенение, полученное при помощи геометрии *geom_rect()*

Заметьте, что заливка соответствует периодам нахождения партий у власти. Цветовая схема заливки будет настроена позже.

Шаг 16. Добавьте линейную диаграмму на основании выбора пользователя

Следующий код используется для создания слоя визуализации с линейной диаграммой:

```
geom_line(data = dfLineChartInfo, aes(x = Year, y = GetGDPStat))
```

Линейная диаграмма создается при помощи геометрии `geom_line()`. При этом в качестве исходных данных мы используем датафрейм `dfLineChartInfo`, а эстетикам `x` и `y` поставим в соответствие столбцы `Year` и `GetGDPStat`.

Шаг 17. Раскрасьте фоновую заливку в соответствии с предопределенной цветовой схемой партий

Для установки цветовой схемы фоновой заливки используем следующий код:

```
scale_fill_manual(values=partyColors)
```

Функцию `scale_fill_manual()` можно использовать в `ggplot2` для переопределения цветовой схемы, применяемой по умолчанию. Ранее мы определили именованный символьный вектор `partyColors` с цветами, соответствующими политическим партиям. Библиотека `ggplot2` использует этот вектор для определения цветовой схемы затенения, если передать его в качестве параметра `values` в функцию `scale_fill_manual()`.

Шаг 18. Отформатируйте ось `y` в соответствии с выбором пользователя

Далее нам необходимо отформатировать ось `y` в соответствии с выбором пользователя следующим образом:

```
scale_y_continuous(labels=
  ifelse(gdpStatName == "ActualGDP",
    comma_format(),
    percent_format())
) +
```

Если пользователь хочет видеть ВВП в абсолютных значениях, ось `y` будет отформатирована при помощи функции `comma_format()`, поскольку нам нужен числовой формат. В противном случае для форматирования будет использована функция `percent_format()`, которая поможет отобразить данные о ВВП в виде процентов. Функции `comma_format()` и `percent_format()` располагаются в пакете `scales`.

Шаг 19. Добавьте метки на оси `x` и `y`

Следующий фрагмент кода позволит нам добавить динамическую метку на ось `y` и статическую – на ось `x`:

```
ylab(yAxisName) +
xlab("Year")
```

Шаг 20. Снабдите диаграмму динамическим заголовком и подзаголовком

Функция `ggtitle()` поможет вам динамически создать заголовок и подзаголовок для диаграммы следующим образом:

```
ggtitle(chartTitle, subtitle = chartSubtitle)
```

Шаг 21. Измените внешний вид диаграммы в стиле журнала The Economist

Поскольку мы визуализируем экономические данные, давайте применим тему `theme_economist()` из пакета `ggthemes`. В этой теме используется форматирование, приближенное к журналу *The Economist*. Применить тему `theme_economist()` к визуализации проще простого:

```
theme_economist()
```

Шаг 22. Перенесите код в Power BI

Полный скрипт для создания нашей визуализации приведен ниже. Этот скрипт полностью функционален и может быть перенесен в Power BI. Убедитесь, что вы вставили его после закомментированной секции в редакторе скриптов:

```
library(tidyverse)
library(ggthemes)
library(scales)

currentColumns <- sort(colnames(dataset))
requiredColumns <- c("GetGDPStat", "GetGDPStatName", "GetPoliticalLevel",
"GetPoliticalLevelName", "Year")
columnTest <- isTRUE(all.equal(currentColumns, requiredColumns))

politicalLevelName <- unique(dataset$GetPoliticalLevelName)
gdpStatName <- unique(dataset$GetGDPStatName)

if(length(politicalLevelName) == 1 & length(gdpStatName) == 1 & columnTest)
{
  dfPI <- dataset

  #Переменные для динамической составляющей диаграммы
  politicalLevelName <- unique(dfPI$GetPoliticalLevelName)
  gdpStatName <- unique(dfPI$GetGDPStatName)
  yAxisName <- paste(gdpStatName, ifelse(gdpStatName == "Actual GDP", "(in
Billions)", ""), sep = " ")
  chartTitle <- paste(gdpStatName, "Analysis", sep = " ")
  chartSubtitle <- paste(politicalLevelName, "view", sep = " ")
}
```

```

dfPI$GetPoliticalLevel <- as.character(dfPI$GetPoliticalLevel)
runs <- rle(dfPI$GetPoliticalLevel)
group_id <- rep(seq_along(runs$lengths), runs$lengths)

#Создаем набор данных для областей затенения
dfShadeInfo <-
  cbind(dfPI, group_id) %>%
  transmute(group_id, Year, Party = GetPoliticalLevel) %>%
  group_by(group_id, Party) %>%
  summarize(start = min(Year), end = max(Year)+0.99) %>%
  ungroup()

#Создаем набор данных для слоя линейной диаграммы
dfLineChartInfo <-
  dfPI %>%
  transmute(Year, GetGDPStat)

#Определяем цветовую схему заливки
partyColors = c("Republican"="red", "Democrat"="blue", "Tie" = "white")

#Создаем визуализацию
p <- ggplot() +
  geom_rect(
    data = dfShadeInfo,
    aes(xmin = start, xmax = end, fill = Party,
        ymin = -Inf, ymax = Inf, alpha = 0.4, color = NA)
  ) +
  geom_line(data = dfLineChartInfo, aes(x = Year, y = GetGDPStat)) +
  scale_fill_manual(values=partyColors) +
  scale_y_continuous(
    labels=ifelse(gdpStatName == "Actual GDP", comma_format(),percent_format())
  ) +
  ylab(yAxisName) +
  xlab("Year") +
  ggtitle(chartTitle, subtitle = chartSubtitle) +
  theme_economist()

p
} else {
  plot.new()
  title("The data supplied did not meet the requirements of the chart.")
}

```

КАРТА

Пространственная аналитика (spatial analytics) в последние годы получила огромное распространение. Одной из форм этой области знаний, особенно популярной в среде бизнеса, является *географическая пространственная аналитика* (geographic-based spatial analytics). Power BI располагает встроенными средствами построения визуализаций в виде географических карт,

а в данной главе мы посмотрим, как можно отображать пользовательские карты в Power BI при помощи языка R. Мы с вами построим так называемую *тепловую карту* (heat map) штата, на которой округа будут выкрашены в определенные цвета в соответствии с квинтилем численности населения, в который попадает округ. Чем темнее цвет, тем больше численность населения в округе. На рис. 2.25 показан пример визуализации применительно к штату Индиана (Indiana).

Indiana's County Population Analysis
(the darker shades the higher the population)

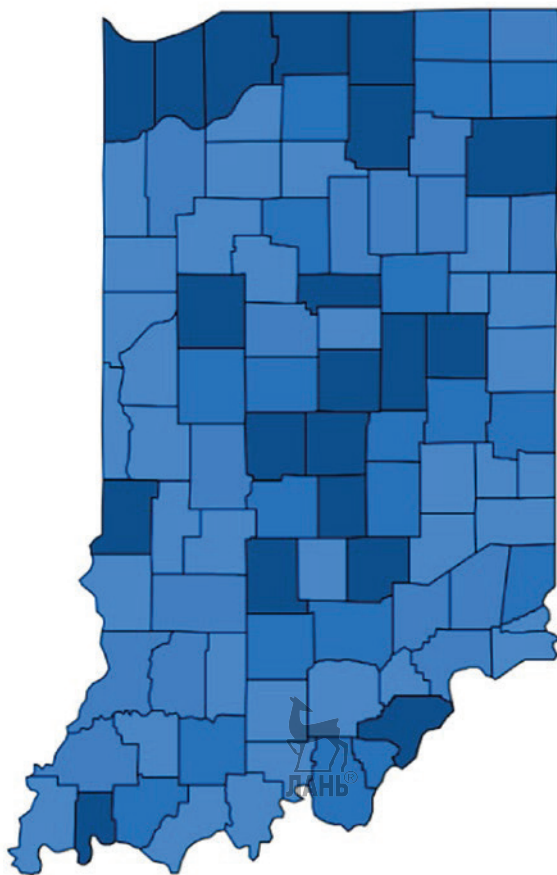


Рис. 2.25 ❖ Визуализация плотности населения по округам в штате Индиана при помощи пакета *ggplot2*

Визуализация была построена на основе набора данных, содержащего координаты границ округов в континентальной части Соединенных Штатов. При выборе конкретного штата библиотека *ggplot2* использует географические координаты для отображения карты. При этом округа выкрашены

в разные оттенки синего – чем темнее оттенок, тем выше численность населения округа. Для переключения между штатами можно воспользоваться интерактивными инструментами из состава Power BI. Если переключить вид с Индианы на Кентукки (Kentucky), визуализация будет выглядеть так, как показано на рис. 2.26.

Kentucky's County Population Analysis
(the darker shades the higher the population)

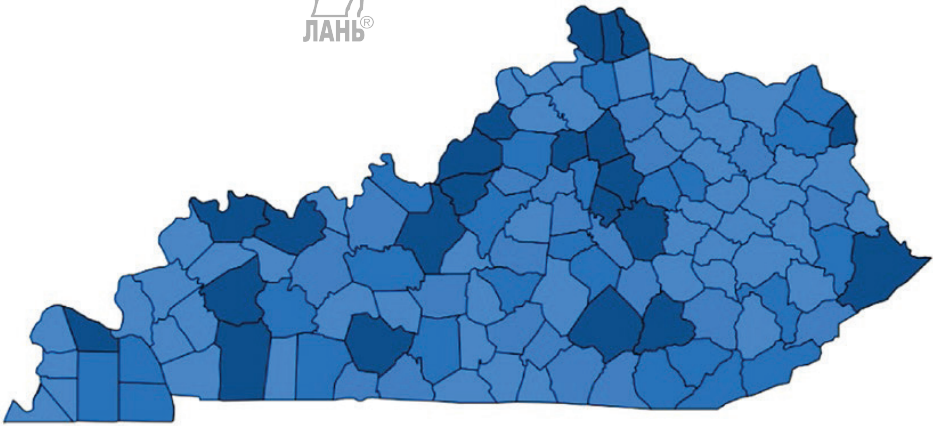


Рис. 2.26 ❖ Визуализация плотности населения по округам в штате Кентукки при помощи пакета *ggplot2*

Одним из заметных преимуществ такой визуализации является ограничение карты конкретным выбранным штатом. Это позволяет не занимать львиную долю пространства на экране информацией, которая пользователю в данный момент не нужна. Кроме того, поскольку при построении данной визуализации мы используем пакет *ggplot2*, то можем воспользоваться всеми его преимуществами, включая добавление динамических заголовков и подзаголовков. Давайте пройдем по шагам и вместе построим эту привлекательную визуализацию.

Шаг 1. Загрузите исходные данные

1. Загрузите набор данных с перечислением всех штатов, включая их полные и сокращенные названия. В качестве источника данных мы воспользовались следующей страницей из Википедии: https://en.wikipedia.org/wiki/List_of_U.S._state_abbreviations. Информацию можно найти на листе *Abbreviations* в рабочей книге *DataWrangling.xlsx*.
2. Постройте набор данных с необходимой для визуализации информацией об округах. Источником данных для этой информации послужил сайт http://www.nrcs.usda.gov/wps/portal/nrcs/detail/national/home/?cid=nrcs143_013697, а сами данные можно обнаружить на листе *County FIPS Codes* в файле *DataWrangling.xlsx*.

3. Загрузите информацию о численности населения по округам с сайта Census. Готовые данные можно найти на листе *PopulationInfo* в рабочей книге *DataWrangling.xlsx* вместе со скриптом на языке Python для получения этого набора.
4. Создайте набор данных, содержащий недостающие FIPS-коды (коды Федерального стандарта обработки информации) округов. Какие-то из них могут быть пропущены из-за проблем с картой, другие могут отсутствовать в исходном наборе данных *County FIPS Codes*. Этот набор данных мы будем использовать для исправления указанного недостатка. Сведения о недостающих кодах располагаются на листе *TheMissingInfo* в файле *DataWrangling.xlsx*.
5. Получите из пакета *ggplot2* информацию о границах округов в континентальной части штатов США. В рабочей книге *DataWrangling.xlsx* эти данные собраны на листе *Map Info*, а ниже представлен код для получения этой информации:

```
library(tidyverse)
us_counties <- map_data("county", ".")
path <- "<место, где вы хотите сохранить набор данных>"
write_csv(us_counties, path)
```

6. Используйте *Power Query* для преобразования полученных наборов данных в вид, пригодный для визуализации. Итоговый набор данных можно найти на рабочем листе *ChartData* в файле *DataWrangling.xlsx*. Запрос, с помощью которого были собраны эти данные, также называется *ChartData*, и вы при желании можете познакомиться со всеми его шагами. Именно этот итоговый набор данных мы будем использовать для построения визуализации в Power BI.
7. Сохраните данные на листе *ChartData* в рабочей книге. Также эта информация находится в CSV-формате в файле *chartdata.csv*.

Как и в случае с любой визуализацией, да и с любым проектом, связанным с анализом данных, большая часть времени обычно уходит на сбор, очистку и преобразование исходных данных. В этом примере я выполнил всю работу за вас, чтобы вы могли сконцентрироваться на построении визуализации. Но вы всегда можете пристально рассмотреть все шаги по обработке данных в репозитории к этой главе. Тем из вас, кто знаком с R и/или Python, будет особенно полезно сделать это, чтобы познакомиться с инструментом *Power Query*.

Шаг 2. Загрузите данные в Power BI

Вам необходимо загрузить данные из файла *chartdata.csv* в Power BI и немного поработать с ними, прежде чем импортировать в модель данных. Все выполненные преобразования вы можете рассмотреть в файле с расширением *pbi*.

Шаг 3. Создайте срез в отчете на основании выбранного в фильтре штата

Создайте срез на основании столбца *State*, предварительно перенеся его в рабочую область и поменяв тип визуализации на элемент среза. Как вы уже знаете, элемент среза помечен на панели визуализаций иконкой воронки.

Шаг 4. Настройте визуальный элемент R в Power BI

Перейдите на *панель визуализаций* (Visualization pane) и перенесите визуальный элемент R в рабочую область. Измените размер перенесенного элемента под свои требования. Добавьте поля *index*, *lat*, *long*, *County*, *Total Population* и *State* из таблицы *chartdata* на панель **Поля** (Fields), расположенную под панелью визуализаций. Если эта панель не отображается, щелкните по элементу визуализации R, и она должна появиться.

Шаг 5. Экпортируйте данные в R Studio для дальнейшей разработки

Нажмите на диагональную стрелку в правой части встроенного в Power BI редактора скриптов R. Это приведет к экспорту данных и базового кода на языке R в *R Studio*. Обратитесь к четвертому шагу инструкции по созданию диаграммы с аннотацией за дополнительной информацией.

Шаг 6. Загрузите необходимые пакеты

Для этого примера вам потребуется загрузить следующие пакеты:

```
library(tidyverse)
library(ggthemes)
```

Пакет *dplyr* из коллекции *tidyverse* понадобится для выполнения необходимых преобразований данных. Пакет *stringr* из той же коллекции будет применен для форматирования заголовков, а библиотека *ggplot2* – для построения графической визуализации. Из библиотеки *ggthemes* мы возьмем тему для визуализации *theme_map()*.

Шаг 7. Создайте переменные для проверки данных

Ниже представлен код для проверки данных:

```
currentColumns <- sort(colnames(dataset))
requiredColumns <- c("County", "Index", "lat", "long", "State", "Total Population")
columnTest <- isTRUE(all.equal(currentColumns, requiredColumns))
state <- str_to_title(unique(dataset$State))
```

В переменной *currentColumns* хранится список имен столбцов набора данных, переданного в R из Power BI, в виде символьного вектора. При этом элементы отсортированы в алфавитном порядке для облегчения сравнения. Переменная *requiredColumns* служит для хранения списка желаемых столбцов также в виде упорядоченного по алфавиту вектора. Результат выполнения сравнения этих двух переменных мы будем хранить в переменной *columnTest*. В ней мы использовали функцию *all.equal()* для проверки на равенство векторов *currentColumns* и *requiredColumns*. В случае их равенства будет возвращено значение TRUE, иначе – информация о том, в чем состоят различия между содержимым этих переменных. Нам нужно получить булево значение, поэтому мы оборачиваем выражение в функцию *isTRUE()*, которая гарантированно вернет TRUE или FALSE в зависимости от результата выполнения операции сравнения. В заключительной строке кода мы получаем уникальные значения из столбца *State* датафрейма *dataset* и используем функцию *str_to_title()* из пакета *stringr*, чтобы привести названия штатов к приемлемому формату.

Шаг 8. Выполните проверку данных

Используем следующий шаблон для проверки данных:

```
if (length(state) == 1 & columnTest) {
  <execute code to build visual>
} else {
  plot.new()
  title("The data supplied did not meet the requirements of the chart.")
}
```

Если в нашем наборе данных представлен только один штат и проверка идентичности столбцов прошла успешно, создаем наполнение нашего элемента визуализации. В противном случае создаем пустую диаграмму с сообщением о том, что исходные данные не отвечают требованиям визуализации.

Шаг 9. Создайте переменные для заголовков диаграммы

Следующий код понадобится вам для создания заголовков диаграммы:

```
chartTitle <- paste0(state, "'s County Population Analysis")
subTitle <- "(the darker shades the higher the population)"
```

Основной заголовок получается путем выполнения конкатенации названия выбранного штата и строки «'s County Population Analysis» (Анализ численности населения округов). Результат присваиваем переменной *chartTitle*.

Для подзаголовка используем строку «the darker shades the higher the population» (чем темнее цвет, тем выше численность населения), присвоенную переменной *subTitle*.

Шаг 10. Добавьте к набору данных столбец с квинтилем



Если помните, целью нашей визуализации является группирование округов в штатах США по квинтилям на основе численности населения. Используем функцию `ntile()` из пакета `dplyr` для группировки округов по квинтилям следующим образом:

```
chartdata <-
  dataset %>%
  mutate(quintile = factor(ntile(`Total Population`, 5)))
```

В результате выполнения этого кода мы получим новый датафрейм с именем `chartdata`, для чего сначала при помощи оператора конвейера передадим имеющийся датафрейм `dataset` первым аргументом в функцию `mutate()`. В этой функции создается новый столбец в датафрейме с именем `quintile` посредством вызова функции `ntile()`. Сама эта функция обернута в функцию `factor()` с целью преобразования квинтилей в тип данных фактор. Это действие имеет схожий эффект с выбором варианта **Не суммировать** (Don't Summarize) в Power BI. Преобразование квинтилей в фактор переводит их из разряда числовых данных в категориальные.

Шаг 11. Создайте символьный вектор для хранения цветовой схемы затенения

Мы будем использовать разные оттенки синего цвета для заливки округов внутри выбранного штата. Для этого воспользуемся именованным символьным вектором, созданным при помощи следующего кода:

```
quintileColors <-
  c(
    "1" = "dodgerblue",
    "2" = "dodgerblue1",
    "3" = "dodgerblue2",
    "4" = "dodgerblue3",
    "5" = "dodgerblue4"
  )
```

Цвет `dodgerblue` из первого квинтиля характеризует наиболее светлый оттенок синего в данном списке, тогда как `doderblue4` из пятого квинтиля – наиболее темный. Здесь мы используем названия для определения цветов вместо шестнадцатеричных значений. Откройте файл по указанной ссылке, чтобы узнать, какие цвета доступны вам при использовании библиотеки `ggplot2`: www.stat.columbia.edu/~tzheng/files/Rcolor.pdf. Там вы найдете перечисление всех цветов с примерами, которые можете использовать в R.



Шаг 12. Постройте диаграмму при помощи функции ggplot()

Для построения диаграммы воспользуемся следующим вызовом функции `ggplot()`:

```
ggplot(chartdata, aes(long, lat, group = County, fill = quintile))
```

В качестве источника данных мы использовали датафрейм `chartdata`. Настройку эстетик выполнили при помощи функции `aes()`, ассоциировав с координатой *x* столбец `long`, с координатой *y* – `lat`, с эстетикой `group` – столбец `County`, а с `fill` – `quintile`. Здесь мы впервые столкнулись с эстетикой `group`, поскольку ранее работали исключительно с индивидуальными геометриями (*individual geom*), тогда как `geom_polygon()` относится к так называемым коллективным геометриям (*collective geom*).

Примечание. Индивидуальные геометрии зависят лишь от одной строки данных, а коллективные – от множества. В данном примере мы применим геометрию `geom_polygon()`, относящуюся к разряду коллективных, и будем использовать данные из нескольких строк для определения границ многоугольника.

Геометрия `geom_polygon()` позволит нам строить многоугольники с использованием широты и долготы в наборе данных для групп, каждая из которых представляет свой округ. Свойство `fill` используется, чтобы сообщить библиотеке `ggplot2` информацию о том, на основании чего вы собираетесь выполнять цветовую заливку округов. В данном случае основанием для выбора цвета будет квинтиль по численности населения, в который входит округ.

Примечание. Мы можем использовать *прямоугольную (декартову) систему координат* (Cartesian coordinate system) – в данном случае систему координат, базирующуюся на координатах *x* и *y*, – для представления пространственных данных, поскольку имеем дело с относительно небольшими площадями земной поверхности. Несмотря на то что Земля имеет форму шара, когда речь идет о небольших географических участках, вы вполне можете проводить анализ с использованием тех же техник, которые применяются на плоскости, и получать при этом весьма приемлемые результаты.

Шаг 13. Добавьте слой с картой

Следующий код может быть использован для добавления на визуализацию слоя с географической картой:

```
geom_polygon(show.legend = FALSE, color = "black")
```

Мы заранее установили эстетики, необходимые для отрисовки географической карты, при определении функции `ggplot()`. Все эти эстетики по умолчанию будут перенесены в функцию `geom_polygon()`, поскольку характеристики, определенные в функции `ggplot()`, автоматически распространяются на все вложенные слои. Кроме того, нам нет необходимости выводить рядом с картой легенду, в связи с чем мы присвоили аргументу `show.legend` значение `FALSE`. В довершение мы передали аргументу `color` текстовое значение «black», чтобы границы многоугольников были черными.

Примечание. Заметьте, что, когда вы хотите присвоить атрибуту на диаграмме определенный цвет, делать это следует без использования функции `aes()`. Подробно причины этого мы обсуждали в главе 1.

Шаг 14. Отформатируйте оси x и y

Следующий фрагмент кода поможет вам выполнить форматирование осей визуализации:

```
scale_x_continuous(name = NULL, labels = NULL, breaks = NULL) +
scale_y_continuous(name = NULL, labels = NULL, breaks = NULL)
```

Как видите, мы убираем названия осей, метки и линии сетки, присвоив соответствующим атрибутам значение `NULL`, поскольку мы выводим географическую карту, на которой эти элементы графиков просто не нужны.

Шаг 15. Раскрасьте округа на основании квинтилей

При помощи следующей строки кода можно выполнить цветовую заливку округов:

```
scale_fill_manual(values = quintileColors)
```

В пакете предусмотрены очень подходящие значения по умолчанию, что и делает эту библиотеку такой популярной. Но еще приятнее то, что эти значения по умолчанию с легкостью можно переопределить. На рис. 2.27 показано, как будет выглядеть цветовая заливка округов в пакете `ggplot2` с использованием значений по умолчанию.

На рис. 2.28 видно, как будет выглядеть заливка, если на вход функции `scale_fill_manual()` передать нашу пользовательскую цветовую схему, собранную ранее.

В таком виде пользователю гораздо легче визуально определить, какие округа во Флориде являются наиболее и наименее населенными, ориентируясь на оттенки синего цвета.

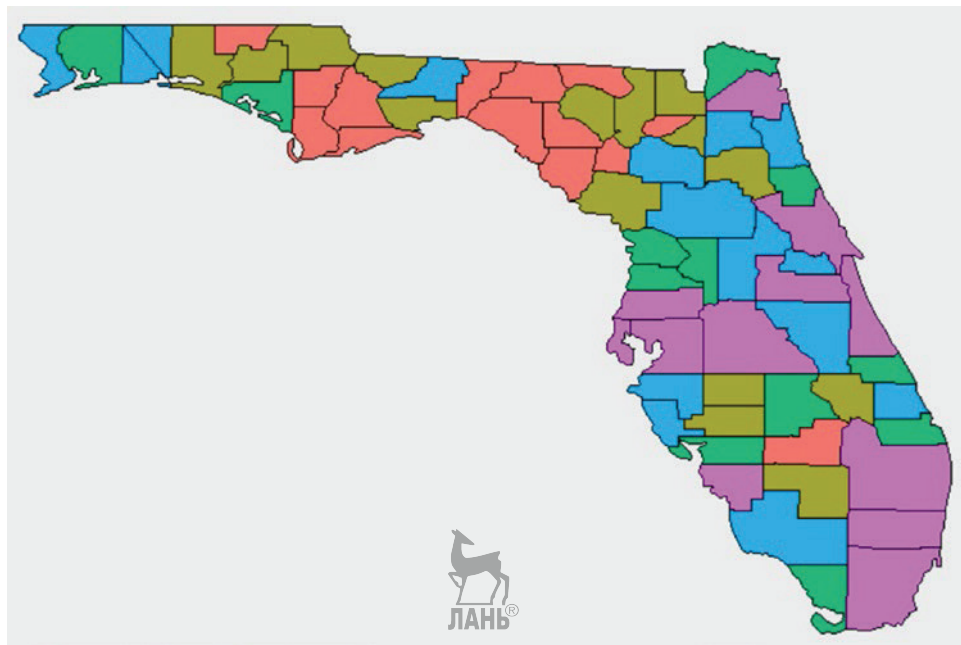


Рис. 2.27 ❖ Визуализация географической карты штата Флорида с цветовой заливкой по умолчанию

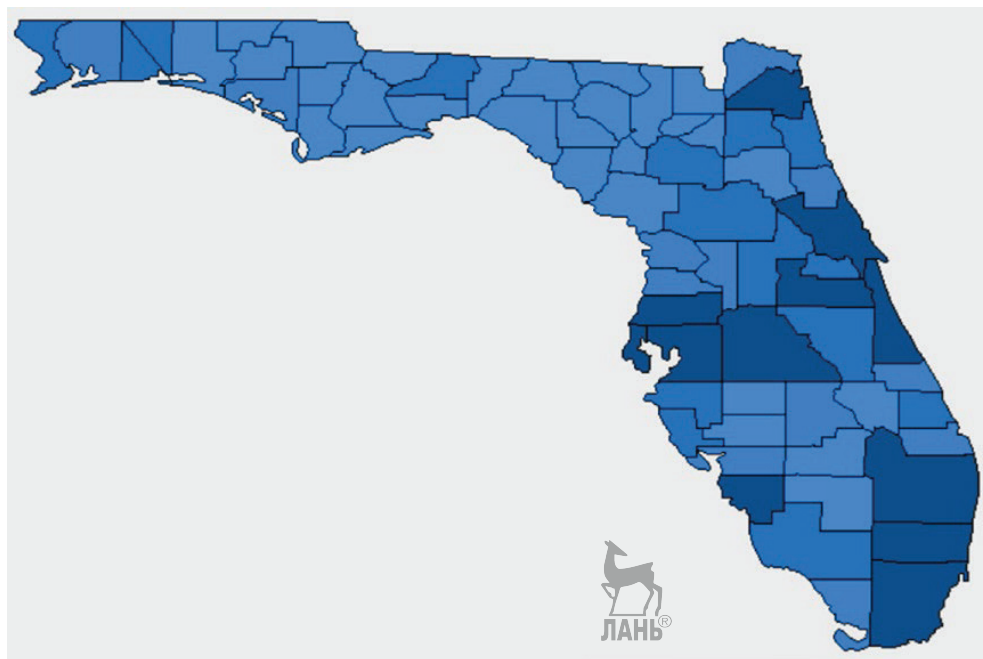


Рис. 2.28 ❖ Визуализация географической карты штата Флорида с пользовательской цветовой заливкой

Шаг 16. Улучшите отображение карты выбранного штата

Следующая строка кода поможет вам выполнить быструю аппроксимацию пропорций на карте при помощи *равноугольной цилиндрической проекции Меркатора* (mercator map projection):

```
coord_quickmap()
```

Функция *geom_polygon()* прекрасно справляется с выводом географической карты, но эта строка кода поможет сделать изображение более точным.

Шаг 17. Снабдите диаграмму динамическим заголовком и подзаголовком

Здесь мы воспользуемся функцией *ggtitle()* для снабжения диаграммы динамическим заголовком и подзаголовком:

```
ggtitle(chartTitle, subtitle = subTitle)
```

Шаг 18. Примените тему theme_map()

Воспользуемся темой *theme_map()* из пакета *ggthemes* для повышения привлекательности географической карты.

Шаг 19. Перенесите код в Power BI

Полный код визуального элемента на языке R, который можно перенести во встроенный редактор скриптов Power BI, представлен ниже. В результате визуализация будет реагировать на выбор штата в добавленном срезе:

```
library(tidyverse)
library(ggthemes)

currentColumns <- sort(colnames(dataset))
requiredColumns <- c("County", "Index", "lat", "long", "State", "Total Population")
columnTest <- isTRUE(all.equal(currentColumns, requiredColumns))
state <- str_to_title(unique(dataset$State))

if (length(state) == 1 & columnTest) {
  chartTitle <- paste0(state, "'s County Population Analysis")
  subTitle <- "(the darker shades the higher the population)"

  chartdata <-
    dataset %>%
    mutate(quintile = factor(ntile(`Total Population`, 5)))
```

```

quintileColors <-
  c(
    "1" = "dodgerblue",
    "2" = "dodgerblue1",
    "3" = "dodgerblue2",
    "4" = "dodgerblue3",
    "5" = "dodgerblue4"
  )

ggplot(chartdata, aes(long, lat, group = County, fill = quintile)) +
  geom_polygon(show.legend = FALSE, color = "black") +
  scale_x_continuous(name=NULL, labels=NULL, breaks=NULL) +
  scale_y_continuous(name=NULL, labels=NULL, breaks=NULL) +
  scale_fill_manual(values = quintileColors) +
  coord_quickmap() +
  ggtitle(chartTitle, subtitle = subTitle) +
  theme_map()
} else {
  plot.new()
  title("The data supplied did not meet the requirements of the chart.")
}

```

ДИАГРАММА КВАДРАНТОВ

Если вы работаете в области анализа данных с продуктами от *Microsoft*, вам должен быть прекрасно известен «магический квадрант» от *Gartner*, изображенный на рис. 2.29.

Microsoft уверенно лидирует в области производства продуктов для бизнес-аналитики, и исследовательской компании *Gartner* это прекрасно известно. Более того, в *Microsoft* прекрасно осведомлены о своем привилегированном положении и используют в том числе и эту диаграмму, чтобы упрочить свое лидерство в этой области.

В качестве основы при построении своей знаменитой диаграммы в *Gartner* воспользовались так называемой *квадратной диаграммой*, или *диаграммой квадрантов* (quad chart), прекрасно подходящей для сравнения сущностей на основании двух показателей или метрик. В данном случае производится сравнение поставщиков продуктов в области бизнес-аналитики по двум экспертным шкалам: *полноте видения* (completeness of vision) и способности реализации (ability to execute). При этом в компании дополнили диаграмму незначительными деталями, которые значительно улучшили ее зрительное восприятие. Среди этих деталей можно выделить заголовки квадрантов, затенение второго и третьего секторов, использование нетрадиционных меток для осей *x* и *y* и добавление подписи к диаграмме.

Все эти нюансы не так просто реализовать при помощи стандартных средств Power BI, в то же время R и его пакет *ggplot2* прекрасно справляются с этой задачей.



Рис. 2.29 ❖ «Магический квадрант» от Gartner 2020 года

В этом разделе мы воспользуемся набором статистических данных по игрокам баскетбольной команды *LA Lakers* в сезонах НБА 2008–2009 годов. В результате мы построим диаграмму квадрантов, показанную на рис. 2.30, в которой игроки команды сравниваются по количеству набранных очков и сделанных подборов.

Как и раньше, мы воспользуемся полным спектром визуальных возможностей от Power BI, включая срезы, помогающие лучше понять зашифрованные в данных информационные послы. В частности, к этой диаграмме мы применим фильтры по типу игры (дома или на выезде) и четверти матча. На рис. 2.31 показано сравнение игроков в матчах, проведенных вдали от дома.

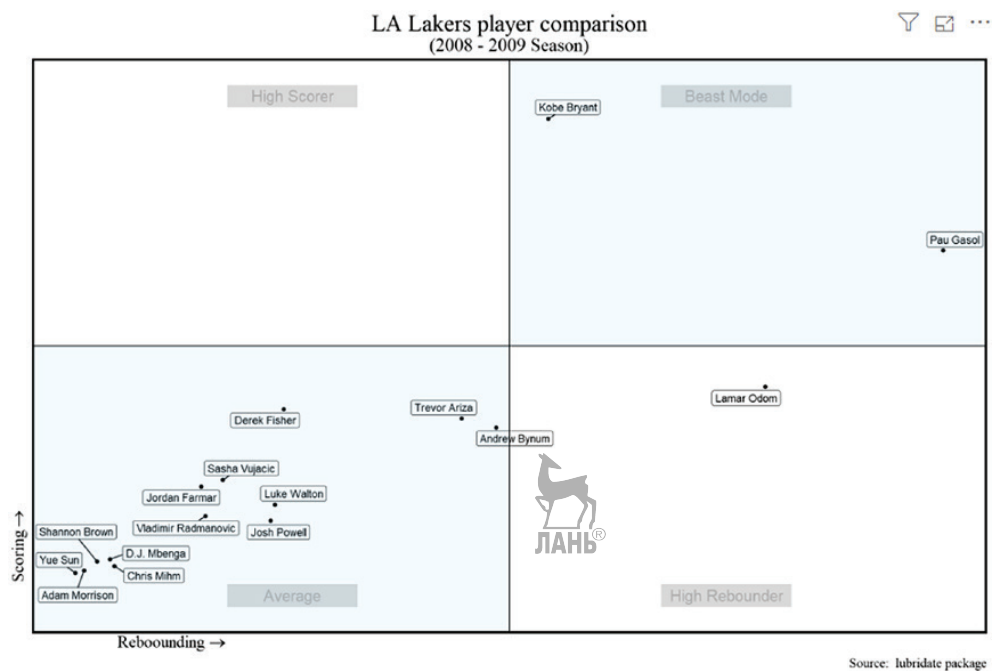


Рис. 2.30 ❖ Диаграмма квадрантов для сравнения игроков команды LA Lakers

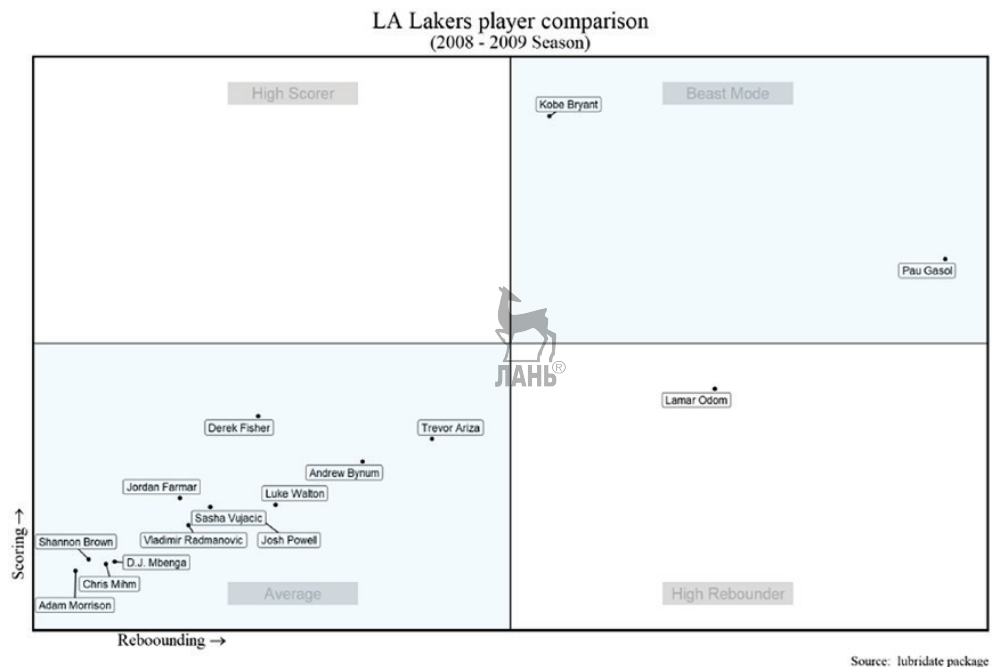


Рис. 2.31 ❖ Диаграмма квадрантов для сравнения игроков команды LA Lakers в выездных играх

Как и всегда, пройдемся по шагам и создадим эту прекрасную диаграмму вместе.



Шаг 1. Загрузите исходные данные

Набор данных для визуализации мы извлекли из пакета *lubridate*. Фрагмент кода для получения нужной нам информации приведен ниже:

```
library(tidyverse)
library(lubridate)

chartData <-
  lakers %>%
  filter(
    (player != "" & team == "LAL" &
     result == "made" &
     etype %in% c("shot", "free throw"))
    |
    (player != "" &
     team == "LAL" &
     etype == "rebound")
  ) %>%
  mutate(
    rebound = ifelse(etype == "rebound", 1, 0)
  ) %>%
  group_by(game_type, period, player) %>%
  summarize(
    `Total Points` = sum(points),
    Rebounds = sum(rebound)
  ) %>%
  rename(
    `Game Type` = game_type,
    Period = period,
    Player = player
  )

path <- "<путь для сохранения файла>"
write_csv(chartdata, path)
```

Итоговый набор данных был сохранен в виде файла CSV, который будет легко импортировать в Power BI. Не волнуйтесь, если вы не полностью поняли представленный выше фрагмент кода, мы подробнее поговорим об используемой в нем технике далее в этой книге. На данный момент вам достаточно знать, что если вам необходимо воссоздать набор данных для этой визуализации, довольно воспользоваться приведенным скриптом на языке R.

Шаг 2. Загрузите данные в Power BI

Загрузите данные из файла CSV, полученного на предыдущем шаге, в Power BI при помощи инструмента **Получить данные** (GetData). Типы данных в загруженной таблице должны быть такими, как показано в табл. 2.4.

Таблица 2.4. Столбцы таблицы с типами данных

Имя столбца	Тип данных
Game type	Text
Period	Whole Number
Player	Text
Total Points	Whole Number
Rebounds	Whole Number

Данные уже импортированы в файл с расширением *pbi*x, который вы можете загрузить из репозитория этой главы. Вы также можете внимательно рассмотреть шаги в Power Query, которые были применены для преобразования данных.

Шаг 3. Создайте срезы в отчете по типу игры и четверти матча

Наш отчет будет содержать два среза. В первом пользователь сможет выбрать тип игры (дома или на выезде), представленный полем *Game Type*, а во втором – номер четверти матча (поле *Period*). Перенесите эти поля в рабочую область и создайте на их основании срезы, воспользовавшись соответствующей иконкой на панели визуализаций.

Шаг 4. Настройте визуальный элемент R в Power BI

Перейдите на *панель визуализаций* (Visualization pane) и перенесите визуальный элемент R в рабочую область. Измените размер перенесенного элемента под свои требования. Добавьте поля *Player*, *Rebounds* и *Total Points* из таблицы *chartData* на панель **Поля** (Fields), расположенную под панелью визуализаций. Если панель с полями не отображается, выделите визуальный элемент R, и она должна появиться.

Шаг 5. Экспортируйте данные в R Studio для дальнейшей разработки

Нажмите на диагональную стрелку в правой части встроенного в Power BI редактора скриптов R. Это приведет к экспорту данных и базового кода на

языке R в *R Studio*. Обратитесь к четвертому шагу инструкции по созданию диаграммы с аннотацией за дополнительной информацией.

Шаг 6. Загрузите необходимые пакеты

Для этого примера вам потребуется загрузить следующие пакеты:

```
library(tidyverse)
library(ggrepel)
library(ggthemes)
library(scales)
```

Пакет *dplyr* из коллекции *tidyverse* понадобится для выполнения необходимых преобразований данных. Пакет *ggplot2* из той же коллекции мы используем для построения графической визуализации. Библиотекой *ggrepel* воспользуемся для снабжения точек данных метками, тему *theme_tufte()* возьмем из пакета *ggthemes*, а функцию *rescale()* из пакета *scales* применим с целью выполнения масштабирования.

Шаг 7. Создайте переменные для проверки данных

Ниже представлен код для проверки данных:

```
currentColumns <- sort(colnames(dataset))
requiredColumns <- c("Player", "Rebounds", "Total Points")
columnTest <- isTRUE(all.equal(currentColumns, requiredColumns))
noPlayerDups <-
length(unique(dataset$Player)) == length(dataset$Player)
```

В переменной *currentColumns* мы будем хранить список имен столбцов набора данных, переданного в R из Power BI, в виде символьного вектора. При этом элементы будут отсортированы в алфавитном порядке для облегчения сравнения. Переменная *requiredColumns* служит для хранения списка желаемых столбцов также в виде упорядоченного по алфавиту вектора. Результат выполнения сравнения этих двух переменных мы будем хранить в переменной *columnTest*. В ней, как и раньше, мы использовали функцию *all.equal()* для проверки на равенство векторов *currentColumns* и *requiredColumns*. В случае их равенства будет возвращено значение TRUE, иначе – информация о том, в чем состоят различия между содержимым этих переменных. Нам нужно получить булево значение, поэтому мы оборачиваем выражение в функцию *isTRUE()*, которая гарантированно вернет TRUE или FALSE в зависимости от результата выполнения операции сравнения. В заключительной строке кода мы объявляем переменную *noPlayerDups*, в которой будет храниться результат проверки на наличие дубликатов игроков в наборе данных. Для выполнения этой проверки мы сравним количество уникальных элементов

в столбце *Player* с длиной этого столбца. Если они равны, значит, наш список игроков не содержит дубликатов.

Шаг 8. Выполните проверку данных

Используем следующий шаблон для проверки данных:

```
if (noPlayerDups & columnTest) {
  <code to produce visual>
} else {
  plot.new()
  title("The data supplied did not meet the requirements of the chart.")
}
```

Если в нашем наборе данных представлены только уникальные игроки и проверка идентичности столбцов прошла успешно, создаем наполнение нашего элемента визуализации. В противном случае создаем пустую диаграмму с сообщением о том, что исходные данные не отвечают требованиям визуализации.



Шаг 9. Создайте заголовки диаграммы

В данном примере мы используем статические переменные для инициализации заголовков и подписи к диаграмме при помощи следующего кода:

```
chart.title <- "LA Lakers player comparison"
chart.subtitle <- "(2008 - 2009 Season)"
chart_source <- "Source: lubridate package"
```



Шаг 10. Добавьте дополнительные столбцы в набор данных

Напомним, что мы строим диаграмму с квадрантами для сравнения игроков баскетбольной команды *LA Lakers* по набранным очкам и сделанным подборами. Однако по абсолютным величинам эти два показателя несравнимы, поскольку диапазоны набранных игроками очков значительно превышают диапазоны количества сделанных подборов. Для визуализации нам необходимо привести эти показатели к единому масштабу, что мы и сделаем при помощи следующего фрагмента кода:

```
graph_data <-
  dataset %>%
  mutate(
    Scaled.Rebounds = round(rescale(Rebounds, to = c(-10, 10)), 1)
    , Scaled.TotalPoints = round(rescale(`Total Points`, to = c(-10, 10)), 1)
  )
```

Функция *mutate()* используется здесь для добавления двух столбцов к нашему датафрейму: *Scaled.Rebounds* и *Scaled.TotalPoints*. При создании этих столбцов мы применили функцию *rescale()* из пакета *scales* для масштабирования значений от -10 до $+10$. Эти новые столбцы мы используем на визуализации для представления набранных очков и сделанных подборов игроками.

Шаг 11. Постройте диаграмму при помощи функции *ggplot()*

Теперь при помощи функции *ggplot()* построим диаграмму:

```
p <- ggplot(graph_data,
  aes(x = Scaled.Rebounds, y = Scaled.TotalPoints)
)
```

Набор данных, который мы будем использовать в нашей визуализации, хранится в переменной *graph_data*, а при вызове функции *ggplot()* мы установим лишь две эстетики: *x* и *y*.

Шаг 12. Используйте геометрию *geom_point()* для создания диаграммы рассеяния

Далее нам необходимо разместить на диаграмме точки данных, представляющие игроков. Позиции точек при этом будут определяться на основании масштабированных версий показателей набранных игроками очков и сделанных подборов. Добавим геометрию *geom_point()* при помощи следующего фрагмента кода:

```
p <- ggplot(
  graph_data,
  aes(x = Scaled.Rebounds, y = Scaled.TotalPoints)
) +
  geom_point()
```

Геометрия *geom_point()* наследует эстетики у функции *ggplot()*, так что свойства *x* и *y* не нуждаются в переопределении.

Шаг 13. Добавьте метки игроков для всех квадрантов

Теперь нам нужно снабдить точки данных в визуализации метками, представляющими имя и фамилию игрока. Это можно сделать при помощи геометрии *geom_label_repel()* следующим образом:

```
ggplot(graph_data,
  aes(x = Scaled.Rebounds,
    y = Scaled.TotalPoints)
) +
  geom_point() +
  geom_label_repel(aes(
    label = Player),
    size = 4, show.legend = FALSE
  )
```



Геометрия *geom_label_repel()* тоже наследует эстетики у функции *ggplot()*, так что в этом случае свойства *x* и *y* также не нуждаются в переопределении.

Единственная эстетика, которую нам придется определить, – это *label*, которой мы поставим в соответствие столбец *Player*. Значения из этого столбца будут использоваться в качестве меток для точек данных, в результате чего на визуализацию будут выведены имена и фамилии игроков. Аргументу *size* присвоим значение 4 за пределами функции *aes()*, чтобы все метки точек данных имели одинаковый размер шрифта. Кроме того, в пакете *ggplot2* по умолчанию создаются легенды для всех шкал, добавленных помимо шкал *x* и *y*. Но для *label* нам легенда не нужна, и мы избавились от нее, присвоив свойству *show.legend* значение *FALSE*.

Шаг 14. Добавьте горизонтальные и вертикальные линии, проходящие через центр

Наша визуализация требует прорисовки отдельных линий для осей *x* и *y*. Последние две строки в представленном ниже фрагменте кода добавляют на диаграмму горизонтальную и вертикальную линии, составляющие базовое перекрестие:

```
p <- ggplot(
  graph_data,
  aes(x = Scaled.Rebounds,
    y = Scaled.TotalPoints)
) +
  geom_point() +
  geom_label_repel(
    aes(label = Player),
    size = 4,
    show.legend = FALSE
  ) +
  geom_hline(yintercept = 0) +
  geom_vline(xintercept = 0)
```



Геометрии *geom_hline()* и *geom_vline()* используются для отрисовки линий на диаграммах. Установка параметра *yintercept* в геометрии *geom_hline()* в ноль позволяет нарисовать горизонтальную линию, проходящую через нулевую точку на оси *y*, а обнуление параметра *xintercept* в геометрии *geom_vline()*

приводит к отрисовке вертикальной линии, проходящей через нулевую координату на оси x .

Шаг 15. Добавьте на диаграмму заголовки квадрантов

На рис. 2.32 мы видим, как выглядит наша диаграмма после выполнения всех описанных выше шагов.

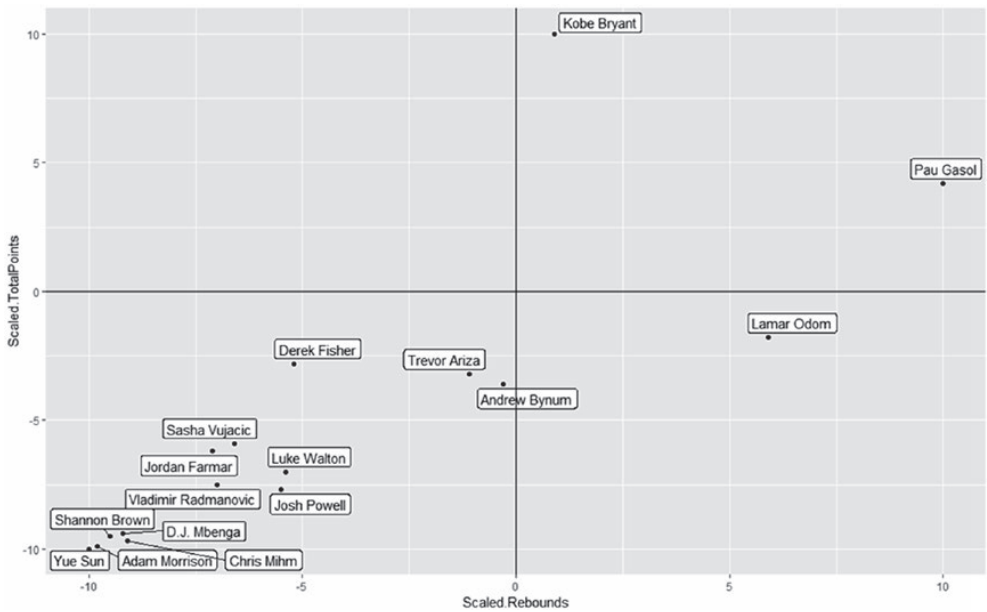


Рис. 2.32 ❖ Заготовка для диаграммы квадрантов

Наша визуализация постепенно обретает все более знакомые очертания. На очередном шаге мы добавим заголовки для каждого из четырех квадрантов при помощи следующего фрагмента кода:

```
annotate("text", x = -5, y = 11, label = "High Scorer", alpha = 0.2, size = 6) +
annotate("rect", xmin = -3.5, xmax = -6.5, ymin = 10.5, ymax = 11.5, alpha = .2) +
annotate("text", x = 5, y = 11, label = "Beast Mode", alpha = 0.2, size = 6) +
annotate("rect", xmin = 3.5, xmax = 6.5, ymin = 10.5, ymax = 11.5, alpha = .2) +
annotate("text", x = -5, y = -11, label = "Average", alpha = 0.2, size = 6) +
annotate("rect", xmin = -3.5, xmax = -6.5, ymin = -11.5, ymax = -10.5, alpha = .2) +
annotate("text", x = 5, y = -11, label = "High Rebounder", alpha = 0.2, size = 6) +
annotate("rect", xmin = 3.5, xmax = 6.5, ymin = -11.5, ymax = -10.5, alpha = .2) +
```

Здесь мы размещаем в рабочей области текстовые метки с обрамлениями, соответствующие четырем квадрантам. Верхний левый квадрант именуется «High Scorer» (бомбардир), поскольку игроки, попадающие в эту группу,

приносят своей команде много очков, но при этом совершают мало подборов, то есть не борются за мяч под кольцами. Верхний правый квадрант мы назвали «Beast Mode» (режим монстра) – в этот сектор попадают игроки, набирающие много очков и делающие много подборов. Левый нижний квадрант озаглавлен «Average» (середняки), поскольку игроки в этой группе не выделяются из общей массы ни по заброшенным мячам, ни по подборам. И заключительный сегмент (правый нижний) назван «High Rebounder» (мастер подборов). Игроки из этой части диаграммы – настоящие рабочие лошадки, они не так много очков приносят команде, но постоянно и успешно борются за мяч, тем самым позволяя своим партнерам по команде начинать все новые атаки.

И текстовые метки, и обрамления создаются при помощи одной и той же функции *annotate()*. Первым аргументом функции мы указываем тип геометрии для отрисовки – в данном случае мы использовали варианты «*text*» и «*rect*» для вывода текстовой надписи и границ обрамляющего прямоугольника соответственно. Сначала посмотрим, как работает функция *annotate()* при выводе текста, а затем узнаем, как при помощи нее можно рисовать линии на диаграмме.

Как мы уже сказали, для вывода текстовой метки необходимо первым аргументом в функцию *annotate()* передать строковый параметр «*text*». Аргументы *x* и *y* определяют координаты расположения текста. В данном случае нам нужно разместить метки квадрантов по центру соответствующих сегментов, что мы и делаем, задавая соответствующие значения аргумента *x*. Аргумент *y* отвечает за вертикальное положение текста, и мы присваиваем ему значения 11 и –11, чтобы текст не перекрывал основную массу точек данных. Аргумент *alpha* в случае со всеми текстовыми метками мы установили в значение 0.2, чтобы имена спортсменов в случае перекрытия визуально находились поверх названий квадрантов.

Примечание. Аргумент *alpha* используется для установки прозрачности геометрий на диаграмме. Чем ближе значение к нулю, тем более прозрачным окажется элемент, а чем ближе к единице, тем его очертания будут более отчетливыми.

После создания текстовых меток квадрантов необходимо добавить им прямоугольную обводку. Это можно сделать, передав в качестве первого аргумента в функцию *annotate()* строковое значение «*rect*». Координаты углов прямоугольника определяются аргументами *xmin*, *xmax*, *ymin* и *ymax*. Ниже представлено описание каждого из них:

- аргумент *xmin* определяет координату *x* для левого верхнего и левого нижнего углов прямоугольника;
- аргумент *xmax* определяет координату *x* для правого верхнего и правого нижнего углов прямоугольника;
- аргумент *ymin* определяет координату *y* для левого нижнего и правого нижнего углов прямоугольника;
- аргумент *ymax* определяет координату *y* для левого верхнего и правого верхнего углов прямоугольника.

После определения координат углов прямоугольника функция *annotate()* соединяет полученные точки линиями. В результате мы получаем прямоугольное обрамление меток квадрантов.

Шаг 16. Добавьте метки на оси x и y

Следующий фрагмент кода мы используем для снабжения метками осей *x* и *y*:

```
xlab(bquote("Rebounding" ~ symbol('\256'))) +
ylab(bquote("Scoring" ~ symbol('\256'))) +
```

Здесь я использовал трюк, вызвав функцию *bquote()*, которая обычно применяется для создания математических выражений в языке R. В данном случае мы воспользуемся ей для вставки символа стрелки на метки осей. Помогут нам в этом функции *xlab()* и *ylab()*. Функция *xlab()* позволит переименовать ось *x*, а *ylab()* – ось *y*. Для вставки самой стрелки используем выражение *symbol('\256')*.

Шаг 17. Снабдите диаграмму динамическими заголовками и подписями

Заголовок, подзаголовок и подпись на диаграмму можно добавить при помощи функции *labs()* следующим образом:

```
labs(
  title = chart.title,
  subtitle = chart.subtitle,
  caption = chart_source
)
```

Шаг 18. Примените тему *theme_tufte*

Давайте применим к нашей диаграмме тему *theme_tufte()* из пакета *ggthemes*, которая базируется на принципах, заложенных признанным экспертом в области визуализации данных Эдвардом Тафти. Для этого добавьте к коду следующую строку:

```
theme_tufte() +
```

Шаг 19. Выполните финальную очистку

Выполнение предшествующих шагов приведет к построению визуализации, показанной на рис. 2.33.

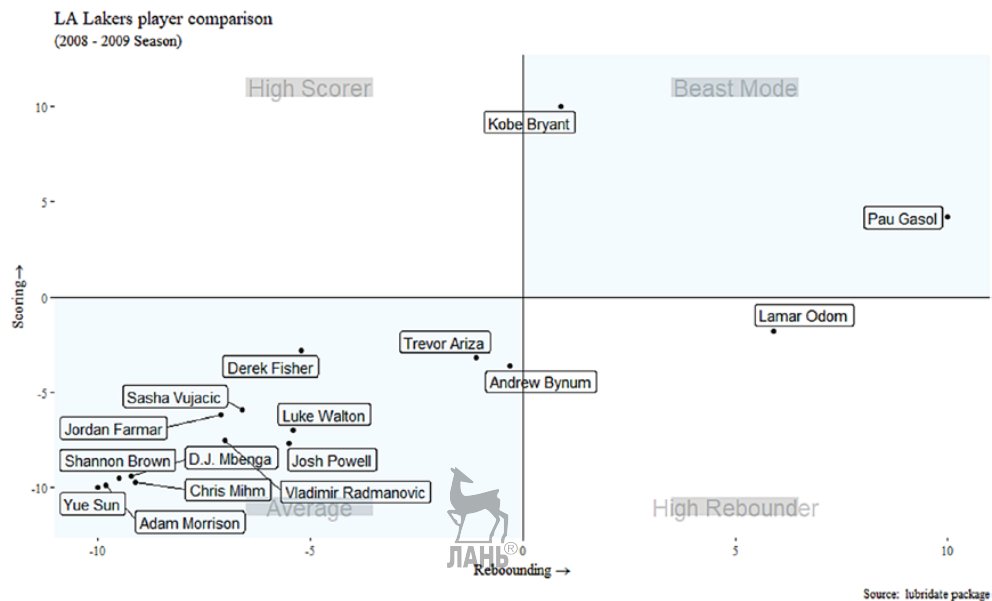


Рис. 2.33 ❖ Предварительная версия диаграммы квадрантов

Нам бы хотелось изменить формат заголовков и меток на осях, чтобы диаграмма выглядела так, как на рис. 2.34.

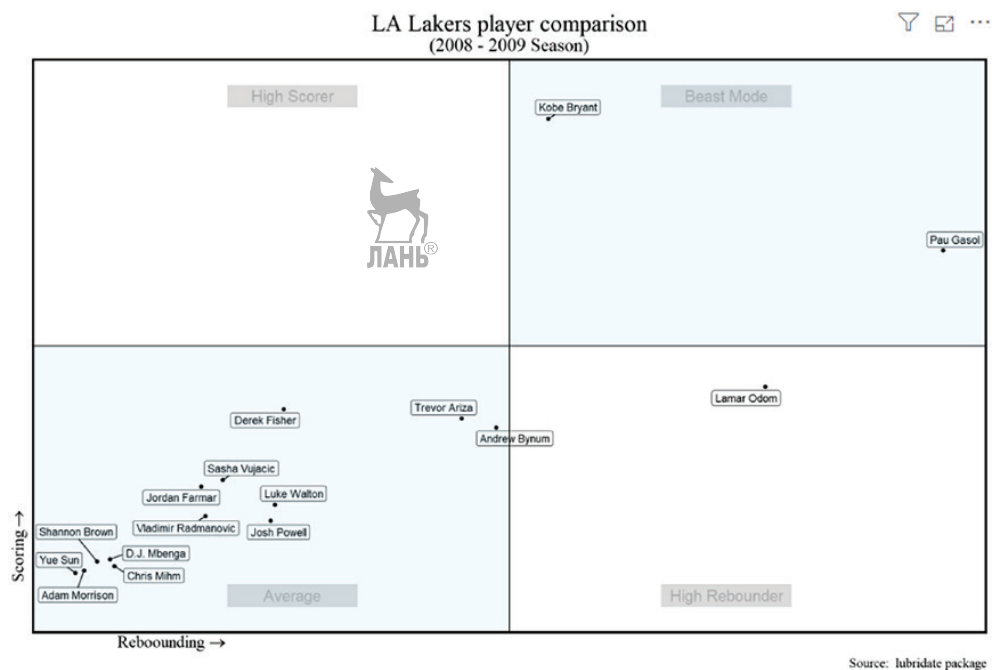


Рис. 2.34 ❖ Финальная версия диаграммы квадрантов

Сделаем это при помощи следующего фрагмента кода:

```
theme(
  plot.title = element_text(hjust = 0.5, size = 25)
  , plot.subtitle = element_text(hjust = 0.5, size = 20)
  , panel.border = element_rect(
    colour = "black",
    size = 2,
    fill = NA)
  , axis.title.x = element_text(hjust = 0.1, size = 18)
  , axis.title.y = element_text(hjust = 0.1, size = 18)
) +
scale_x_continuous(labels = NULL, breaks = NULL) +
scale_y_continuous(labels = NULL, breaks = NULL)
```

Давайте пройдемся по представленному выше коду детально. Как видите, большинство изменений делается внутри функции *theme()*. Ниже приведен перечень свойств, которые подверглись изменениям, вместе с подробным описанием каждой модификации:

- *plot.title*: это свойство отвечает за заголовок диаграммы, а для изменения формата данного элемента мы воспользовались функцией *element_text()*. С помощью аргумента *hjust*, принимающего значение от 0 до 1, можно изменить горизонтальное положение текста следующим образом: 0 располагает текст по левому краю, 0.5 центрирует его, а 1 относит его направо. Мы использовали центральное значение для расположения текста по центру. Изменение свойства *size* позволило увеличить размер шрифта до 25 единиц;
- *plot.subtitle*: это свойство отвечает за подзаголовок диаграммы, и форматирование его также выполняется при помощи функции *element_text()*. В целом здесь все свойства и принципы те же, что и при работе с заголовком диаграммы. Отметим лишь, что размер шрифта для подзаголовка мы сделали чуть меньше, а именно 20 единиц;
- *panel.border*: этот компонент характеризует границы элемента визуализации, для модификации которых используется функция *element_rect()*. Аргумент *colour* ожидаемо отвечает за цвет границ, а *size* – за их толщину;
- *axis.title.x*: это свойство позволяет контролировать внешний вид метки оси *x*, для чего используется функция *element_text()*. Мы хотим, чтобы метка располагалась ближе к левому краю диаграммы, поэтому присвоим аргументу *hjust* значение 0.1. Посредством аргумента *size* установим размер шрифта равным 18;
- *axis.title.y*: это свойство позволяет менять внешний вид метки оси *y* подобно тому, как мы меняли метку оси *x*. Таким образом, чем ближе будет значение аргумента *hjust* к нулю, тем ниже будет располагаться метка вертикальной оси. Мы хотим, чтобы метка располагалась ближе к нижнему краю диаграммы, так что присвоим аргументу *hjust* значение 0.1. Размер шрифта метки сделаем также равным 18.

Наконец, функции *scale_x_continuous()* и *scale_y_continuous()* используются для переформатирования осей. Метки и линии сетки нам здесь не нужны,

так что мы присвоили соответствующим аргументам этих функций значение NULL. Финальный вид диаграммы мы уже показывали ранее на рис. 2.34.

Шаг 20. Перенесите код в Power BI



Полный скрипт для нашего элемента визуализации приведен ниже:

```
library(tidyverse)
library(ggrepel)
library(ggthemes)
library(scales)

currentColumns <- sort(colnames(dataset))
requiredColumns <- c("Player", "Rebounds", "Total Points")
columnTest <- isTRUE(all.equal(currentColumns, requiredColumns))
noPlayerDups <- length(unique(dataset$Player)) == length(dataset$Player)

if (noPlayerDups & columnTest) {
  chart.title <- "LA Lakers player comparison"
  chart.subtitle <- "(2008 - 2009 Season)"
  chart_source <- "Source: lubridate package"

  graph_data <-
    dataset %>%
    mutate(
      Scaled.Rebounds = round(rescale(Rebounds, to = c(-10, 10)), 1)
      , Scaled.TotalPoints = round(rescale(`Total Points`, to = c(-10, 10)), 1)
    )

  p <- ggplot(graph_data, aes(x = Scaled.Rebounds, y = Scaled.TotalPoints)) +
    geom_point() +
    geom_label_repel(aes(label = Player), size = 4, show.legend = FALSE) +
    geom_hline(yintercept = 0) +
    geom_vline(xintercept = 0) +

    # quad 1
    annotate("text", x = -5, y = 11, label = "High Scorer", alpha = 0.2, size = 6) +
    annotate("rect", xmin = -3.5, xmax = -6.5, ymin = 10.5, ymax = 11.5, alpha = .2) +

    # quad 2
    annotate("text", x = 5, y = 11, label = "Beast Mode", alpha = 0.2, size = 6) +
    annotate("rect", xmin = 3.5, xmax = 6.5, ymin = 10.5, ymax = 11.5, alpha = .2) +

    # quad 3
    annotate("text", x = -5, y = -11, label = "Average", alpha = 0.2, size = 6) +
    annotate("rect", xmin = -3.5, xmax = -6.5, ymin = -11.5, ymax = -10.5, alpha = .2) +

    # quad 4
    annotate("text", x = 5, y = -11, label = "High Rebounder", alpha = 0.2, size = 6) +
    annotate("rect", xmin = 3.5, xmax = 6.5, ymin = -11.5, ymax = -10.5, alpha = .2) +

    # Shade lower left quadrant
    annotate("rect", xmin = -Inf, xmax = 0.0, ymin = -Inf, ymax = 0, alpha = 0.1,
```

```

fill = "lightskyblue") +

# Shade upper right quadrant
annotate("rect", xmin = 0.0, xmax = Inf, ymin = 0.0, ymax = Inf, alpha = 0.1,
fill = "lightskyblue") +

# Titles
xlab(bquote("Rebounding" ~ symbol('\256'))) +
ylab(bquote("Scoring" ~ symbol('\256'))) +

labs(title = chart.title, subtitle = chartsubtitle, caption = chart_source) +

# Prettying things up
theme_tufte() +
theme(
  plot.title = element_text(hjust = 0.5, size = 25)
  , plot.subtitle = element_text(hjust = 0.5, size = 20)
  , panel.border = element_rect(colour = "black", size = 2, fill = NA)
  , axis.title.x = element_text(hjust = 0.1, size = 18)
  , axis.title.y = element_text(hjust = 0.1, size = 18)
) +
scale_x_continuous(labels = NULL, breaks = NULL) +
scale_y_continuous(labels = NULL, breaks = NULL)
p
} else {
  plot.new()
  title("The data supplied did not meet the requirements of the chart.")
}

```

Этот скрипт полностью функционален и может быть перенесен в Power BI. Элемент визуализации будет полностью интерактивен и зависеть от выбора пользователя.

ДОБАВЛЕНИЕ ЛИНИИ РЕГРЕССИИ

Одним из наиболее популярных улучшений, которые аналитик данных может внести в диаграмму рассеяния, является добавление *линий тренда* (trend lines). Добавить линии тренда на диаграмму рассеяния в R с использованием пакета *ggplot2* проще простого. К тому же в этом случае, в отличие от других инструментов, в вашем распоряжении будет наиболее полный спектр пользовательских настроек. К примеру, вы сможете:

- использовать для линий тренда *линейные модели* (linear models), *обобщенные линейные модели* (generalized linear models), *обобщенные аддитивные модели* (generalized additive models) и некоторые другие;
- сами определять, какой должна быть *доверительная вероятность* (confidence level);
- использовать для линий тренда свои собственные формулы;
- применять множество других пользовательских настроек, перечислить которые просто невозможно.

В данном разделе мы будем работать с простейшим примером, основанном на наборе данных, который поставляется вместе с R. Имя этого набора данных – *women*, и он показывает зависимость среднего веса от роста женщин в США. Мы представим эти данные в виде диаграммы рассеяния, после чего добавим линию тренда на основе *линейной регрессионной модели* (linear regression model). Желаемый результат показан на рис. 2.35.

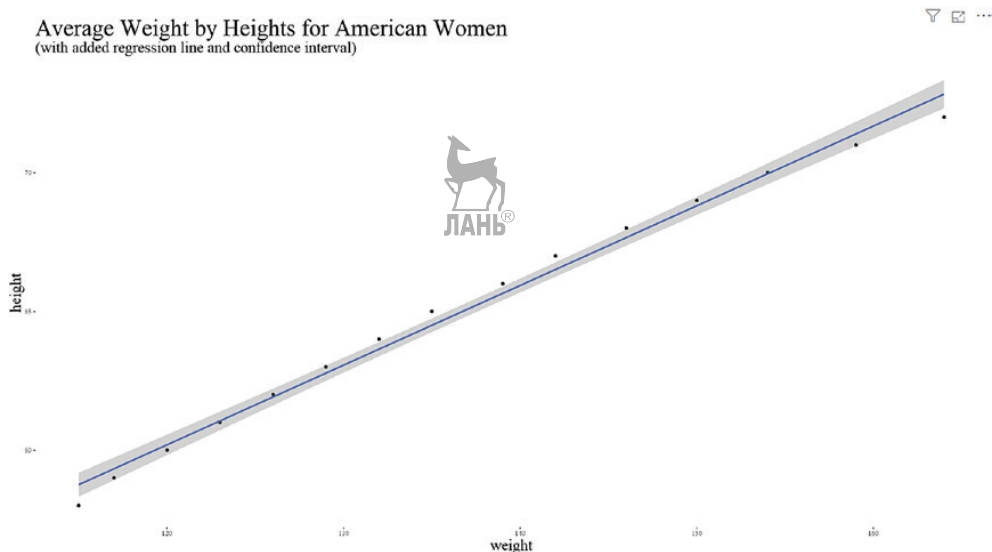


Рис. 2.35 ❖ Диаграмма рассеяния с добавленной линией регрессии

Давайте по сложившейся традиции пройдемся по шагам.

Шаг 1. Загрузите исходные данные

Набор данных, который мы будем использовать в нашем примере, поставляется вместе с языком R. При помощи приведенного ниже кода можно создать файл CSV, который будет являться источником данных для рассматриваемого в этом разделе примера:

```
library(tidyverse)

data(women)
setwd("<путь к желаемой папке>")
write_csv(women, "women.csv")
```



В результате запуска этого скрипта будет создан файл *women.csv*, который мы на следующем шаге используем в качестве источника данных для Power BI.

Шаг 2. Загрузите данные в Power BI

Загрузите данные из файла CSV, полученного на предыдущем шаге, в Power BI при помощи инструмента **Получить данные** (GetData), указав путь к файлу *women.csv*. По окончании загрузки убедитесь, что поля *height* и *weight* имеют числовой тип данных.

Шаг 3. Настройте визуальный элемент R в Power BI

Перейдите на *панель визуализаций* (Visualization pane) и перенесите визуальный элемент R в рабочую область. Измените размер перенесенного элемента под свои требования. Переместите поля *height* и *weight* из таблицы *women* на панель **Поля** (Fields), расположенную под панелью визуализаций. Если панель с полями не отображается, выделите визуальный элемент R, и она должна появиться.



Шаг 4. Экспортируйте данные в R Studio для дальнейшей разработки

Нажмите на диагональную стрелку в правой части встроенного в Power BI редактора скриптов R. Это приведет к экспорту данных и базового кода на языке R в *R Studio*. Обратитесь к четвертому шагу инструкции по созданию диаграммы с аннотацией за дополнительной информацией.



Шаг 5. Загрузите необходимые пакеты

Для этого примера вам потребуется загрузить пакеты *tidyverse* и *ggthemes*. Пакет *ggplot2* из коллекции *tidyverse* мы используем для построения графической визуализации, библиотека *dplyr* нам пригодится для конфигурирования данных на диаграмме, а в качестве темы для визуализации мы применим знакомую вам тему *theme_tufte()* из пакета *ggthemes*.

Шаг 6. Создайте переменные для проверки данных

Ниже представлен код для проверки данных:

```
currentColumns <- sort(colnames(dataset))
requiredColumns <- c("height", "weight")
columnTest <- isTRUE(all.equal(currentColumns, requiredColumns))
```

Нам необходимо убедиться в том, что набор данных, переданный в R из Power BI, имеет правильную структуру, и в противном случае иметь возможность эту структуру изменить. В этом примере мы удостоверимся, что в переданном наборе содержатся лишь нужные нам столбцы.

Для этого мы сначала сохраняем в переменной *currentColumns* список имен столбцов датафрейма *dataset*, переданного в R из Power BI, в виде символьного вектора. При этом элементы будут отсортированы в алфавитном порядке для облегчения задачи сравнения. Переменная *requiredColumns* служит для хранения списка желаемых столбцов также в виде упорядоченного по алфавиту символьного вектора. Результат выполнения сравнения этих двух переменных мы будем хранить в переменной *columnTest*. Здесь мы использовали функцию *all.equal()* для проверки на равенство векторов *currentColumns* и *requiredColumns*. В случае их равенства будет возвращено значение TRUE, иначе – информация о том, в чем состоят различия между содержимым этих переменных. Нам нужно получить булево значение, поэтому мы оборачиваем выражение в функцию *isTRUE()*, которая гарантированно вернет TRUE или FALSE в зависимости от результата выполнения операции сравнения.



Шаг 7. Выполните проверку данных

Используем следующий шаблон для проверки данных:

```
if (columnTest) {
  <code to produce visual>
} else {
  plot.new()
  title("The data supplied did not meet the requirements of the chart.")
}
```



Если проверка идентичности столбцов прошла успешно, создаем наполнение нашего элемента визуализации. В противном случае создаем пустую диаграмму с сообщением о том, что исходные данные не отвечают требованиям визуализации.

Шаг 8. Постройте диаграмму при помощи функции ggplot()

Теперь при помощи функции *ggplot()* построим диаграмму:

```
chartdata <- dataset
ggplot(chartdata, aes(x = weight, y = height)) +
```

Для начала мы создали копию датафрейма *dataset*, сохранив ее в переменной *chartdata*. При вызове функции *ggplot()* мы, помимо передачи источника данных, установим при помощи функции *aes()* две эстетики: *x* и *y*.

Шаг 9. Используйте геометрию `geom_point()` для создания диаграммы рассеяния

Далее нам необходимо разместить на диаграмме точки данных, представляющие исследуемых женщин. При добавлении геометрии `geom_point()` аргументы `x` и `y` будут унаследованы от функции `ggplot()`, так что явно их указывать нет никакой необходимости. Таким образом, требуемый код для добавления нужной нам геометрии упрощается до вида:

```
geom_point() +
```

Результат добавления геометрии `geom_point()` показан на рис. 2.36.

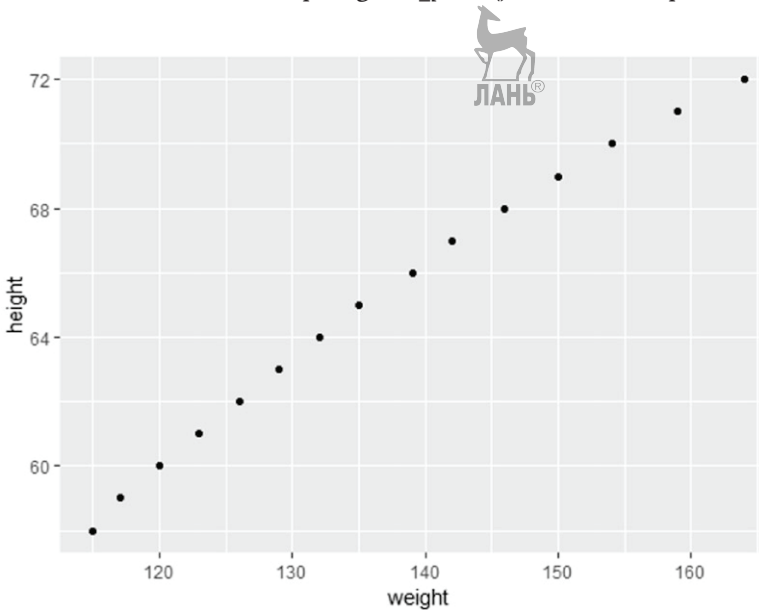


Рис. 2.36 ❖ Диаграмма рассеяния, демонстрирующая зависимость среднего веса американок от их роста

Шаг 10. Добавьте на визуализацию слой с линией регрессии

На этом шаге мы добавим слой с линией регрессии при помощи геометрии `geom_smooth()`. Скрипт, который нам понадобится для этого, очень простой:

```
geom_smooth(method='lm') +
```

В результате будет добавлена линия регрессии на график на основании данных на диаграмме рассеяния. Прелесть этой функции состоит в ее настраиваемости, ведь она позволяет вам:

- задавать при необходимости свою собственную формулу для расчета линейной регрессии;
- добавлять *доверительные интервалы* (confidence intervals) к линиям регрессии;
- выбирать уровень доверительной вероятности по своему усмотрению;
- использовать различные методы сглаживания, например *обобщенную линейную регрессию* (generalized linear regression – glm), *локальную регрессию* (local regression – loess), *робастные линейные модели* (robust linear models – rlm) и *обобщенные аддитивные модели* (generalized additive models – gam).

На рис. 2.37 показан внешний вид диаграммы после добавления линии регрессии.

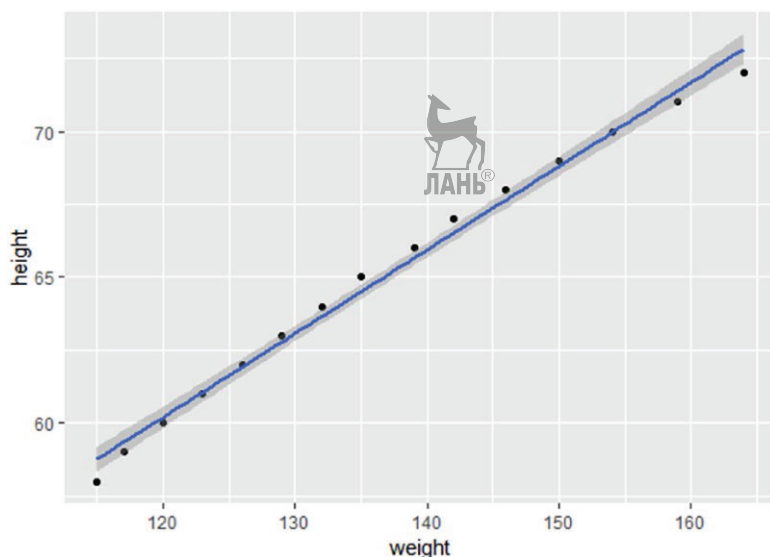


Рис. 2.37 ❖ Диаграмма рассеяния с линией регрессии

Шаг 11. Снабдите диаграмму заголовком и подзаголовком

Воспользуемся функцией `ggtitle()` для добавления на диаграмму заголовка и подзаголовка.

```
ggtitle(
  "Average Weight by Heights for American Women",
  subtitle = "(with added regression line and confidence interval)"
) +
```

Обратите внимание, что функция `ggtitle()` представляет собой альтернативу функции `labs()` с функциональностью, необходимой нам в данном примере.

Шаг 12. Примените тему



Мы хотим, чтобы наша диаграмма отвечала принципам, сформулированным признанным экспертом в области визуализации данных Эдвардом Тафти. Для этого достаточно применить к нашей визуализации тему *theme_tufte()* следующим образом:

```
theme_tufte() +
```

Шаг 13. Выполните финальную очистку

Функцию *theme()* можно использовать для изменения размера шрифта в заголовке, подзаголовке и метках осей. Для модификации заголовка применяется компонент *plot.title*, внешний вид подзаголовка можно изменить при помощи аргумента *plot.subtitle*, а метки осей меняются с помощью изменения параметра *axis.title*. Все три компонента в своих целях используют функцию *element_text()*:

```
theme(
  axis.title = element_text(size = 20),
  plot.title = element_text(size = 30),
  plot.subtitle = element_text(size = 20)
)
```

Шаг 14. Перенесите код в Power BI

Полный скрипт для нашего элемента визуализации приведен ниже. Скопируйте его и перенесите во встроенный редактор скриптов R в Power BI:

```
library(tidyverse)
library(ggthemes)

currentColumns <- sort(colnames(dataset))
requiredColumns <- c("height", "weight")
columnTest <- isTRUE(all.equal(currentColumns, requiredColumns))

if (columnTest) {
  chartdata <- dataset

  ggplot(chartdata, aes(x = weight, y = height)) +
    geom_point() +
    geom_smooth(method='lm') +
    ggtitle(
      "Average Weight by Heights for American Women",
      subtitle = "(with added regression line and confidence interval)"
    ) +
    theme_tufte() +
    theme(
      axis.title = element_text(size = 20),
```

```
        plot.title = element_text(size = 30),  
        plot.subtitle = element_text(size = 20)  
      )  
    } else {  
      plot.new()  
      title("The data supplied did not meet the requirements of the chart.")  
    }  
  }
```

В данной главе мы рассмотрели несколько сценариев, демонстрирующих, насколько выразительными вы можете быть в средствах построения визуализаций в Power BI при помощи пакета *ggplot2*. Все эти сценарии можно легко адаптировать под собственные нужды или использовать их в качестве отправной точки при создании собственных диаграмм. Библиотека *ggplot2* совместно со вспомогательными пакетами представляет собой полноценный фреймворк для создания универсальных и очень сложных визуализаций в Power BI без необходимости писать огромные скрипты. Пакет *ggplot2* – лучший друг Power BI!



Часть II

.....

**ЗАГРУЗКА
ИНФОРМАЦИИ
В МОДЕЛЬ
ДАННЫХ POWER
BI ПРИ ПОМОЩИ R
И PYTHON**



Глава 3



Чтение файлов CSV

Power BI – это превосходный инструмент для визуализации данных, который по праву считается одним из лидеров в этом сегменте. Но перед тем как начать строить потрясающие воображение визуализации данных, для чего и предназначен Power BI, необходимо эти данные каким-то образом загрузить в модель.

На помощь Power BI в этих случаях приходит Power Query – инструмент, позволяющий легко справиться с большинством ситуаций при загрузке информации из различных источников в модель данных. Интуитивно понятный пользовательский интерфейс Power Query значительно облегчает задачу импорта данных в Power BI, но иногда возникают ситуации, когда воспользоваться богатым спектром средств этого инструмента не представляется возможным.

К счастью, Power BI поддерживает языки R и Python, с помощью которых можно легко и просто решить любой нестандартный сценарий по загрузке данных. В данной главе вы научитесь использовать скрипты на языках R и Python для динамической выборки файлов при объединении из папки и фильтрации строк на основании текстовых шаблонов с применением техники сравнения строк. Давайте начнем с динамического объединения файлов.

Динамическое объединение файлов

Объединение файлов из папки перед загрузкой в Power BI – довольно распространенная задача. При этом выполнить динамическую выборку файлов с использованием сложной логики стандартными средствами Power Query бывает очень непросто. И здесь вам на помощь придут скрипты R или Python.

Объединение файлов при помощи Power Query делается довольно легко, при условии что выбираются все файлы в папке с одинаковой структурой данных. Также вы можете использовать простейшую логику для объединения файлов в один набор данных. Что касается применения более сложных правил для выбора файлов, здесь стандартных средств Power Query может не хватить. В этом случае вы всегда можете позвать на помощь скрипты, написанные на языке R или Python.

Пример сценария

Вот вам сценарий для решения. Представьте, что сейчас 5 февраля 2014 года, а вы – финансовый аналитик крупной компании, торгующей всякой всячиной. У вас есть папка, в которой хранятся файлы с продажами по месяцам за период с 2010-12-01 по 2014-01-01. При этом каждый месячный файл занимает довольно много места, и из-за ограничений Power BI у вас нет возможности загрузить все имеющиеся у вас исторические данные. Таким образом, вы решаете осуществлять загрузку информации за скользящий период, исчисляющийся последними 24 месяцами.

Сопутствующая техническая информация:

- все отчеты хранятся в виде файлов CSV;
- принятый шаблон именования файлов: YYYY-MM-DD.csv;
- все файлы имеют одинаковую структуру.

Выработать последовательность действий в Power Query для динамического отбора файлов, входящих в скользящий период на основании последних 24 месяцев, будет весьма непросто. Для этого необходимо очень хорошо знать этот инструмент и уметь программировать на встроенном в Power Query языке M. При этом результирующий код окажется намного более сложным и менее понятным по сравнению со скриптами на R или Python, выполняющими аналогичные действия. Давайте сначала посмотрим, как выполнить описанную задачу при помощи скрипта R, а затем посредством Python. Ознакомившись с этими фрагментами кода, вы поймете, как просто можно манипулировать данными при помощи языков программирования R и Python.

Выбор файлов за скользящий период из 24 месяцев при помощи R

Разработчики на языке R занимаются манипулированием данными уже очень давно – буквально с начала 1990-х. Под этим действием мы понимаем извлечение сырых необработанных данных и подготовку их для дальнейшего анализа. А объединение информации из разных файлов в один набор данных – и вовсе одна из наиболее распространенных задач в области обработки данных. Сообщество R за долгие годы разработало множество полезных пакетов, способных облегчить выполнение подобных задач. И в этой главе мы рассмотрим способы их использования. Давайте пройдем по шагам и решим этот сценарий на языке R вместе.

Шаг 1. Импортируйте необходимые пакеты для скрипта

В данном примере мы будем использовать четыре пакета из коллекции *tidyverse*. Это пакеты *readr*, *lubridate*, *purrr* и *stringr*. Библиотека *readr* используется для загрузки неструктурированных файлов (flat files) в R, пакет *lubridate* облегчает работу с датами, в пакете *purrr* реализованы техники функционального программирования в R, а библиотека *stringr* помогает при работе с текстом.

Пакеты *readr* и *purrr* включены в загрузку библиотеки *tidyverse*, так что вам нет необходимости импортировать их отдельно. В то же время вы должны явно загрузить пакеты *lubridate* и *stringr*, поскольку они не входят в ядро коллекции *tidyverse*. Код для импорта нужных пакетов приведен ниже:

```
library(tidyverse)
library(lubridate)
library(stringr)
```

Вам может показаться, что это все слишком сложно, раз вам необходимо загружать целых четыре библиотеки для работы простого скрипта, чего в Power Query делать не нужно. Все, что нужно для работы в Power Query, есть там по умолчанию. Но это не совсем так. Почитайте эту книгу, и вы поймете, как полезно иметь доступ ко всему многообразию пакетов в R. Используя эти пакеты, вы сможете решать даже очень сложные задачи с написанием небольших фрагментов кода.

Шаг 2. Установите рабочую директорию на папку, содержащую наборы данных о продажах

Рабочая директория (*working directory*) в R относится к корневому каталогу, в котором расположены ваши скрипты и данные. Использование рабочей директории желательно, поскольку позволяет вам хранить все необходимые для работы файлы в одном месте. Кроме того, в этом случае вы сможете использовать *адресацию по относительному пути к файлам* (*relative file path referencing*). Это бывает крайне удобно, поскольку исключает необходимость указания полных путей к файлам. Для рассматриваемого в данной главе примера необходимо установить в качестве рабочей директории путь к папке *R_Code*, вложенной в раздел *Chapter03* в репозитории. Указание на папку с именем *R_Code* содержится во фрагментах кода в данной главе. Таким образом, если вы сохранили сопроводительный код в папке *Documents*, то рабочую директорию необходимо установить при помощи фрагмента кода, приведенного ниже:

```
setwd("C:/Users/<"username">/Documents/AdvancedAnalyticsPowerBI/Chapter03/R_Code")
```

Функция *setwd()* отвечает за установку рабочей директории в соответствии с переданным ей в качестве параметра путем.

Если вы хотите присвоить рабочей директории новый путь относительно текущей рабочей директории, вам необходимо в начале передаваемого пути поставить точку (.). Давайте рассмотрим это на примере. Во фрагменте кода, показанном ниже, предполагается, что текущая рабочая директория указывает на папку *Documents*, а вы хотите поменять ее на папку *R_Code* с использованием прямых слешей:

```
setwd("../AdvancedAnalyticsPowerBI/R_Code ")
```

В этом фрагменте кода мы при помощи точки указываем R начать с папки *Documents*, являющейся вашей рабочей директорией. Далее мы перенаправляем путь рабочей директории в новое место путем указания папок через прямые слешы (/). Вы можете использовать и обратные слешы (\), но в этом случае вам придется их экранировать, указывая дважды, поскольку этот символ в языке R является специальным.

Примечание. Когда вам необходимо использовать символ, входящий в категорию специальных, вы должны дать R или Python понять, что вам нужен сам этот символ, путем его экранирования (escaping) с помощью символа обратного слеша (\). Часто используемые специальные символы, нуждающиеся в экранировании, – это точка (.) и звездочка (*).

Шаг 3. Считайте имена файлов в символьный вектор

Если вам нужно выбрать файлы, входящие в скользящий период за последние 24 месяца, без анализа их имен вам не обойтись. Давайте воспользуемся функцией *list.files()*. Данная функция позволяет считать имена всех файлов, находящихся в директории, переданной ей в качестве параметра. Пример использования этой функции приведен ниже:

```
monthly_reports <- list.files("./Data/SalesData/")
```

Примечание. Символьный вектор (character vector) представляет собой вектор из элементов *символьного типа* (character). Этот тип данных в R напоминает тип *string* в DAX и *text* – в языке M.

В предыдущем коде R собирает имена всех файлов из указанной директории в символьный вектор и присваивает его переменной *monthly_reports*.

Примечание. В языке программирования R сочетание символов <- используется в качестве оператора присваивания значений переменным.

Ниже показан вывод содержимого переменной *monthly_reports*:

```
> monthly_reports
[1] "2010-12-01.csv" "2011-01-01.csv" "2011-02-01.csv"
[4] "2011-03-01.csv" "2011-04-01.csv" "2011-05-01.csv"
[7] "2011-06-01.csv" "2011-07-01.csv" "2011-08-01.csv"
[10] "2011-09-01.csv" "2011-10-01.csv" "2011-11-01.csv"
[13] "2011-12-01.csv" "2012-01-01.csv" "2012-02-01.csv"
[16] "2012-03-01.csv" "2012-04-01.csv" "2012-05-01.csv"
[19] "2012-06-01.csv" "2012-07-01.csv" "2012-08-01.csv"
[22] "2012-09-01.csv" "2012-10-01.csv" "2012-11-01.csv"
[25] "2012-12-01.csv" "2013-01-01.csv" "2013-02-01.csv"
[28] "2013-03-01.csv" "2013-04-01.csv" "2013-05-01.csv"
[31] "2013-06-01.csv" "2013-07-01.csv" "2013-08-01.csv"
[34] "2013-09-01.csv" "2013-10-01.csv" "2013-11-01.csv"
[37] "2013-12-01.csv" "2014-01-01.csv"
```




Шаг 4. Создайте вектор дат

Теперь нам необходимо получить вектор дат, представляющих имена полученных файлов. Чтобы иметь возможность оставить только файлы, входящие в скользящий период из 24 месяцев, нужно для начала перевести строки с именами файлов в правильные даты. Это можно сделать в два приема. Сначала нужно избавиться от расширения «.csv», чтобы остались строки, которые можно преобразовать в даты. Для этого воспользуемся функцией *str_replace()* из пакета *stringr*, а ее результат присвоим переменной *date_format*, представляющей символьный вектор. Сделать это можно следующим образом:

```
date_format <- stringr::str_replace(monthly_reports, ".csv", "")
```

Функция *str_replace()* принимает на вход три параметра: строку, которую необходимо изменить, шаблон в строке, который необходимо заменить, и строку подстановки. Ниже показан вывод переменной *date_format* после вызова функции:




```
> date_format
[1] "2010-12-01" "2011-01-01" "2011-02-01" "2011-03-01"
[5] "2011-04-01" "2011-05-01" "2011-06-01" "2011-07-01"
[9] "2011-08-01" "2011-09-01" "2011-10-01" "2011-11-01"
[13] "2011-12-01" "2012-01-01" "2012-02-01" "2012-03-01"
[17] "2012-04-01" "2012-05-01" "2012-06-01" "2012-07-01"
[21] "2012-08-01" "2012-09-01" "2012-10-01" "2012-11-01"
[25] "2012-12-01" "2013-01-01" "2013-02-01" "2013-03-01"
[29] "2013-04-01" "2013-05-01" "2013-06-01" "2013-07-01"
[33] "2013-08-01" "2013-09-01" "2013-10-01" "2013-11-01"
[37] "2013-12-01" "2014-01-01"
```

Примечание. Указывать название пакета *stringr::* перед именем функции *str_replace()* вовсе не обязательно. Это сделано для большей ясности того, из какой именно библиотеки берется функция.

Что нужно сделать дальше? Как мы уже говорили, преобразовать текст в даты. Для этого можно воспользоваться функцией *ymd()* из пакета *lubridate* следующим образом:

```
date_format <- lubridate::ymd(date_format)
```

Функция *ymd()* – очень ловкая и умелая. Она позволяет быстро преобразовать строки в даты и не задает лишних вопросов. Кроме того, эту функцию гораздо легче использовать, чем применять аналогичные средства в других языках, таких как язык *M* в *Power Query*. Мы преобразуем строки в даты и присвоим результат той же переменной *date_format*. Ниже показан вывод содержимого этой переменной после выполнения преобразования:



```
> date_format
[1] "2010-12-01" "2011-01-01" "2011-02-01" "2011-03-01"
[5] "2011-04-01" "2011-05-01" "2011-06-01" "2011-07-01"
```

```
[9] "2011-08-01" "2011-09-01" "2011-10-01" "2011-11-01"
[13] "2011-12-01" "2012-01-01" "2012-02-01" "2012-03-01"
[17] "2012-04-01" "2012-05-01" "2012-06-01" "2012-07-01"
[21] "2012-08-01" "2012-09-01" "2012-10-01" "2012-11-01"
[25] "2012-12-01" "2013-01-01" "2013-02-01" "2013-03-01"
[29] "2013-04-01" "2013-05-01" "2013-06-01" "2013-07-01"
[33] "2013-08-01" "2013-09-01" "2013-10-01" "2013-11-01"
[37] "2013-12-01" "2014-01-01"
```

Как видите, результаты поразительно похожи на предыдущий вывод, но на самом деле они не идентичны. Доказать это можно при помощи функции `str()`. Данная функция используется в языке R для вывода структуры объекта. Если передать переменную `date_format` функции `str()` в качестве параметра, по выводу можно легко понять, что мы имеем дело с вектором объектов типа `date`:

```
> str(date_format)
Date[1:38], format: "2010-12-01" "2011-01-01" "2011-02-01"
```

Примечание. Желательно подобные проверки выполнять в консоли R (R console), а не в редакторе скриптов в R Studio, чтобы по случайности тестовый код не добавился к вашему итоговому скрипту.

Шаг 5. Создайте датафрейм, состоящий из двух векторов

Теперь нам необходимо собрать все наши данные в датафрейм, поскольку это наиболее подходящий контейнер для осуществления аналитики. В R *датафрейм* (data frame) представляет собой двумерный табличный объект, состоящий из переменных (столбцов), которые могут быть разных типов. В какой-то степени датафреймы в R похожи на таблицы в *Microsoft Excel* и *SQL Server*, но предназначены при этом исключительно для аналитики. Код, необходимый для преобразования переменных `monthly_reports` и `date_format` в датафрейм, показан ниже:

```
df <- tibble::data_frame(monthly_reports, date_format)
```

Здесь мы использовали функцию `data_frame()` из пакета *tibble* вместо функции `data.frame()`, входящей в R, поскольку последняя обладает нежелательными побочными эффектами. Функция `data_frame()` возвращает объект с типом данных *tibble*, который можно рассматривать как более современную версию датафрейма. Лично мне с такими объектами работать куда удобнее.

Одно из преимуществ типа данных *tibble* состоит в том, как информация об объекте выводится на экран. При выводе объекта *tibble* вы увидите только первые десять строк из таблицы и лишь те столбцы, которые поместятся на экран. При распечатке традиционного датафрейма выводится гораздо больше информации, и это может стать проблемой при работе с большими объемами данных. Кроме того, в случае с объектом типа *tibble* мы сразу видим, данные каких типов хранятся в каждом столбце, что бывает весьма полезно.

Ниже показано, какая информация будет выведена на экран о датафрейме, хранящемся в переменной *df*:

```
> df
# A tibble: 38 x 2
monthly_reports date_format
<chr>           <date>
1 2010-12-01.csv 2010-12-01
2 2011-01-01.csv 2011-01-01
3 2011-02-01.csv 2011-02-01
4 2011-03-01.csv 2011-03-01
5 2011-04-01.csv 2011-04-01
6 2011-05-01.csv 2011-05-01
7 2011-06-01.csv 2011-06-01
8 2011-07-01.csv 2011-07-01
9 2011-08-01.csv 2011-08-01
10 2011-09-01.csv 2011-09-01
# ... with 28 more rows
```



Шаг 6. Получите верхнюю и нижнюю границы желаемого диапазона дат

Помните, что нам нужен скользящий период за последние 24 месяца? Таким образом, нам необходимо определить верхнюю и нижнюю границы диапазона дат для этого периода. Это можно сделать при помощи следующего кода:

```
max_month <- max(df$date_format)
min_month <- max_month %m+% lubridate::months(-23)
```



Сначала мы использовали функцию *max()* применительно к столбцу *date_format* датафрейма *df* для извлечения последнего месяца и присвоили результат переменной *max_month*. После этого вычли 23 месяца из полученной даты для определения нижней границы скользящего периода. С этой целью мы воспользовались оператором *%m+%* из пакета *lubridate*. Этот оператор служит специально для добавления и вычитания месяцев. В данном случае мы добавили к нашей дате -23 месяца, сохранив результат в переменной *min_month*.

Шаг 7. Ограничьте датафрейм только нужными нам месяцами

На этом шаге мы применим фильтр, который вернет вектор, содержащий только нужные нам файлы для объединения. Для этого нам достаточно ограничить выборку снизу полученным ранее минимальным месяцем, входящим в скользящий период. Это можно сделать следующим образом:

```
reports_to_read <- df$monthly_reports[df$date_format >= min_month]
```

В языке R можно таким образом фильтровать столбец с данными. В нашем случае мы поместили в квадратные скобки фильтрующее выражение `df$date_format >= min_month`, возвращающее булев вектор. При вычислении этого выражения каждый элемент из столбца `date_format` сравнивается со значением переменной `min_month`. В случае, если рассматриваемый элемент больше или равен минимальному месяцу, возвращается значение `True`, в противном случае – `False`. В результате в датафрейме сохранятся только те строки, для которых указанное выше булево выражение вернет значение `True`.

При этом нам нужен только столбец с именами файлов, который мы получим, используя выражение `df$monthly_reports`. Если бы мы хотели получить все столбцы из датафрейма, можно было бы сделать это так, как показано ниже:

```
reports_to_read <- df[df$date_format >= min_month]
```

Такой метод ограничения данных в датафреймах с использованием базовой функциональности языка R прекрасно работает в простых ситуациях. Далее в этой книге вы познакомитесь с более продвинутыми способами фильтрации данных. В переменной `reports_to_read` у нас будут собраны имена файлов CSV, начиная с 2012 года.

Шаг 8. Создайте датафрейм на основании объединенных файлов

Итак, в переменной `reports_to_read` мы собрали перечень файлов для объединения в единый датафрейм. Теперь пришло время выполнить само это объединение. В R можно собрать вместе нужные файлы всего одной строчкой кода с использованием мощной функции `map_df()` из пакета `purrr` следующим образом:

```
df_output <- purrr::map_df(reports_to_read, read_csv)
```

Здесь мы использовали функцию `map_df()` для преобразования каждого элемента из переменной `reports_to_read` посредством вызова функции `read_csv()` из пакета `readr`. Постфикс `_df` в имени функции указывает на то, что в качестве результата функция вернет датафрейм, содержащий данные из всех файлов. Ниже показано, как выглядит наш результирующий датафрейм:

```
> df_output
# A tibble: 54,771 x 9
  Category OrderDate      DueDate
<chr>      <dtm>          <dtm>
1 Bikes    2012-01-01 05:00:00 2013-01-12 05:00:00
2 Accesso~ 2012-01-01 05:00:00 2013-01-12 05:00:00
3 Accesso~ 2012-01-01 05:00:00 2013-01-12 05:00:00
4 Bikes    2012-01-01 05:00:00 2013-01-12 05:00:00
5 Bikes    2012-01-01 05:00:00 2013-01-12 05:00:00
```

```

6 Accesso~ 2012-01-01 05:00:00 2013-01-12 05:00:00
7 Accesso~ 2012-01-01 05:00:00 2013-01-12 05:00:00
8 Bikes    2012-01-01 05:00:00 2013-01-12 05:00:00
9 Accesso~ 2012-01-01 05:00:00 2013-01-12 05:00:00
10 Accesso~ 2012-01-01 05:00:00 2013-01-12 05:00:00
# ... with 54,761 more rows, and 6 more variables:
#   ShipDate <dtm>, OrderQuantity <int>, UnitPrice <dbl>,
#   SalesAmount <dbl>, TaxAmt <dbl>, `Order Month` <chr>

```

Заметьте, как удобно отражена информация. Пакет *tibble* выводит на экран только то, что помещается, и отдельно показывает, какие столбцы (переменные) не были отображены.

Шаг 9. Соберите написанный код и перенесите в редактор скриптов в Power BI

Пришло время скомпоновать весь написанный на предыдущих шагах код в единый скрипт и перенести его во встроенный редактор скриптов R в Power BI. Установите рабочую директорию по своему усмотрению:



```

library(tidyverse)
library(lubridate)
library(stringr)

setwd("./Data/SalesData")
monthly_reports <- list.files(".")
date_format <- stringr::str_replace(monthly_reports, ".csv", "")
date_format <- lubridate::ymd(date_format)
df <- data_frame(monthly_reports, date_format)

max_month <- max(df$date_format)
min_month <- max_month %m+% months(-23)

reports_to_read <- df$monthly_reports[df$date_format >= min_month]
df_output <- purrr::map_df(reports_to_read, read_csv)

```

Скопируйте скрипт на языке R в буфер обмена и откройте файл с расширением *pbix* для этого примера. На вкладке **Главная** (Home) откройте выпадающую кнопку **Получить данные** (GetData), выберите подменю **Другие** (More) и перейдите на вкладку **Другое** (Other). В правой части окна найдите и выберите пункт **R-скрипт** (R script), как показано на рис. 3.1.

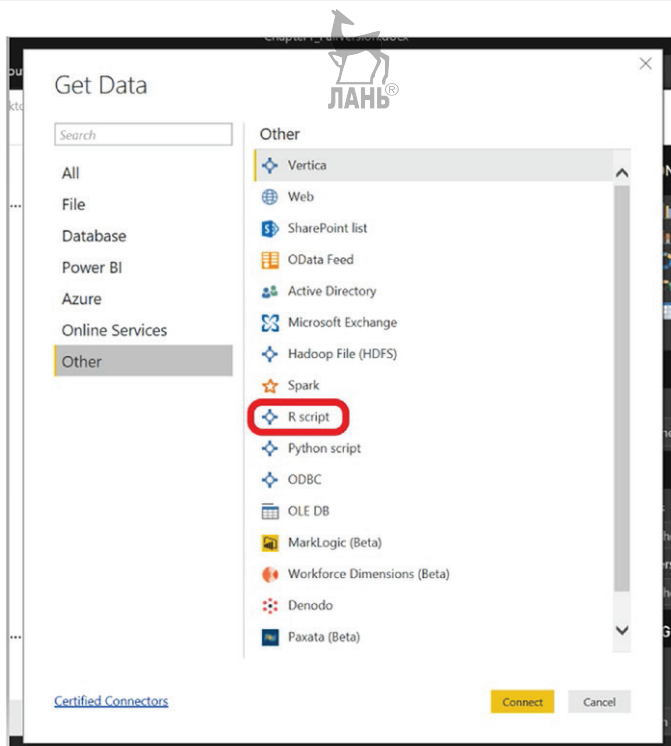


Рис. 3.1 ❖ Открытие редактора скриптов R

После этого в Power BI откроется окно встроенного редактора скриптов R, показанное на рис. 3.2.



Рис. 3.2 ❖ Редактор скриптов R в Power BI

Вставьте скопированный скрипт в это окно и нажмите на кнопку **ОК**. Теперь вы можете обращаться с загруженными посредством скрипта R данными точно так же, как и с любыми другими. Например, вы можете осуществить дополнительное преобразование данных при помощи инструмента Power Query или загрузить их непосредственно в модель данных Power BI.

Выбор файлов за скользящий период из 24 месяцев при помощи Python

Как и язык R, Python появился на заре 90-х. Он также обладает большим количеством инструментов и библиотек, которые способны значительно облегчить задачу объединения файлов за скользящий временной период. Давайте перечислим шаги, которые необходимо проделать, чтобы решить этот сценарий при помощи языка Python.

Шаг 1. Создайте скрипт на Python и загрузите необходимые библиотеки

Первое, что вам необходимо сделать, – это загрузить библиотеки и функции, которые понадобятся при написании скрипта. Для этого используем следующий фрагмент кода:

```
import os
import pandas as pd
from dateutil.relativedelta import relativedelta
```

Здесь мы полностью загружаем две библиотеки (*os* и *pandas*), а также одну функцию (*relativedelta*). Пакет *os* используется для взаимодействия с операционной системой, библиотека *pandas* позволяет облегчить работу с данными, а функция *relativedelta()* из пакета *dateutil* поможет удобно манипулировать датами.

Шаг 2. Установите рабочую директорию на папку Python_Code

Аналогом функции языка R *setwd()* в Python является функция *chdir()* из библиотеки *os*. Чтобы воспользоваться этой функцией, достаточно передать ей в качестве аргумента путь по тем же правилам, что и в R. Следующая строка кода поможет вам направить рабочую директорию на папку *Python_Code* при условии, что вы скачали сопроводительные файлы в папку *Documents*:

```
os.chdir("C:/Users/<"username">/Documents/AdvancedAnalyticsPowerBI/Python_Code")
```

Теперь в своем скрипте вы можете использовать относительные пути к файлам с началом в папке *Python_Code*.

Шаг 3. Загрузите перечень имен файлов в список

Теперь прочитаем имена файлов из папки *SalesData* в список. Если вам необходимо получить данные за скользящий период из 24 месяцев, вам придется анализировать имена соответствующих файлов на предмет их вхождения в желаемый интервал дат. Для начала воспользуемся функцией *listdir()* из библиотеки *os* – она позволит загрузить имена всех файлов из директории *SalesData* в переменную списочного типа. Это можно сделать, выполнив следующую строку кода:

```
monthly_reports = os.listdir("./Data/SalesData/")
```

В результате в переменной *monthly_reports* будут собраны все без исключения имена файлов из папки *SalesData*. Давайте взглянем на вывод содержимого переменной *monthly_reports* на экран:

```
>>> monthly_reports
['2010-12-01.csv', '2011-01-01.csv', '2011-02-01.csv',
'2011-03-01.csv', '2011-04-01.csv', '2011-05-01.csv',
'2011-06-01.csv', '2011-07-01.csv', '2011-08-01.csv',
'2011-09-01.csv', '2011-10-01.csv', '2011-11-01.csv',
'2011-12-01.csv', '2012-01-01.csv', '2012-02-01.csv',
'2012-03-01.csv', '2012-04-01.csv', '2012-05-01.csv',
'2012-06-01.csv', '2012-07-01.csv', '2012-08-01.csv',
'2012-09-01.csv', '2012-10-01.csv', '2012-11-01.csv',
'2012-12-01.csv', '2013-01-01.csv', '2013-02-01.csv',
'2013-03-01.csv', '2013-04-01.csv', '2013-05-01.csv',
'2013-06-01.csv', '2013-07-01.csv', '2013-08-01.csv',
'2013-09-01.csv', '2013-10-01.csv', '2013-11-01.csv',
'2013-12-01.csv', '2014-01-01.csv']
```

Шаг 4. Создайте датафрейм *pandas* с информацией о файлах для объединения

Датафреймы *pandas* очень похожи на датафреймы языка R. Для данной задачи мы воспользуемся именно датафреймами библиотеки *pandas*. Проще всего можно создать датафрейм *pandas* путем преобразования из словаря. В Python *словарь* (dictionary) представляет собой структуру данных, содержащую одну или более пар *ключ* (key)/*значение* (value).

Давайте для начала создадим словарь с единственной парой ключ/значение на основании переменной *monthly_reports* и сохраним его в переменной с именем *d*, как показано ниже:

```
d = {'monthly_reports': monthly_reports}
```

Теперь преобразуем полученный словарь в датафрейм следующим образом:

```
df = pd.DataFrame(d)
```

Мы взяли созданный нами словарь и на его основании получили датафрейм, состоящий из единственного столбца. При этом ключ (строка «*monthly_reports*») из словаря *d* стал названием колонки, а значение (наш список имен файлов) преобразовалось в строки датафрейма.

Шаг 5. Создайте новый столбец с датой в датафрейме

Нам понадобится создать новую колонку в датафрейме с датой. Сделать это можно при помощи следующего фрагмента кода:

```
df["date_format"] = pd.to_datetime(
    df.monthly_reports.str.replace(".csv", "")
)
```

Здесь мы сначала отсекаем расширение *.csv* у имен файлов в столбце *monthly_reports*, оставляя тем самым только текстовое представление даты, а затем преобразуем полученные значения в тип даты посредством функции *to_datetime()* из библиотеки *pandas*. Результат присваиваем выражению *df["date_format"]*, что приводит к созданию нового столбца с данными.

Шаг 6. Определите границы нужного нам диапазона дат

Теперь нам нужно определить верхнюю и нижнюю границы требуемого диапазона дат. Мы по-прежнему хотим извлекать файлы за скользящий период из последних 24 месяцев. Получим нижнюю границу диапазона следующим образом:

```
min_month = df.date_format.max() - relativedelta(months=23)
```

Сначала мы определим последний доступный месяц, вызвав метод *max()* у столбца *date_format*.

Примечание. Каждый столбец в датафрейме *pandas* представляет собой объект типа *Series*. У объекта *Series*, в свою очередь, есть набор методов, которыми можно пользоваться в зависимости от типа данных содержимого. Одним из методов, доступным для объекта *Series* с типом данных *datetime*, является метод *max()*.

Теперь, когда мы вычислили последний месяц в списке файлов, можно отсчитать 24 месяца назад. Это можно сделать, вызвав функцию *relativedelta()* из библиотеки *dateutil* и передав ей в качестве аргумента *months* значение 23. В результате этого будет вычтено 23 месяца из последней даты, обнаруженной в столбце *date_format*.

Шаг 7. Ограничьте датафрейм нужным диапазоном

Теперь, когда вы знаете, из какого диапазона дат вам нужны значения, вы можете ограничить датафрейм *df* только требуемыми датами. Это можно сделать следующим образом:

```
reports_to_read = df[df["date_format"]>=min_month]
```

При помощи фрагмента кода в квадратных скобках создается перечень логических значений на основании того, является ли дата из столбца *date_format* большей или равной значению вычисленной ранее переменной *min_month*. Python оставляет в датафрейме *df* только те строки, в которых логическое выражение в квадратных скобках вернуло значение *True*, и присваивает результирующий набор данных переменной *reports_to_read*. Давайте посмотрим на вывод этой переменной:

```
>>> reports_to_read
monthly_reports date_format
14  2012-02-01.csv  2012-02-01
15  2012-03-01.csv  2012-03-01
16  2012-04-01.csv  2012-04-01
17  2012-05-01.csv  2012-05-01
18  2012-06-01.csv  2012-06-01
19  2012-07-01.csv  2012-07-01
20  2012-08-01.csv  2012-08-01
21  2012-09-01.csv  2012-09-01
22  2012-10-01.csv  2012-10-01
23  2012-11-01.csv  2012-11-01
24  2012-12-01.csv  2012-12-01
25  2013-01-01.csv  2013-01-01
26  2013-02-01.csv  2013-02-01
27  2013-03-01.csv  2013-03-01
28  2013-04-01.csv  2013-04-01
29  2013-05-01.csv  2013-05-01
30  2013-06-01.csv  2013-06-01
31  2013-07-01.csv  2013-07-01
32  2013-08-01.csv  2013-08-01
33  2013-09-01.csv  2013-09-01
34  2013-10-01.csv  2013-10-01
35  2013-11-01.csv  2013-11-01
36  2013-12-01.csv  2013-12-01
37  2014-01-01.csv  2014-01-01
```

Шаг 8. Объедините файлы в единый датафрейм

На данном шаге нам необходимо собрать содержимое всех нужных нам файлов за скользящий 24-месячный период в один датафрейм. Это можно очень просто сделать при помощи следующего кода:

```
df_output = pd.concat(
    map(
        pd.read_csv,
        ["/Data/SalesData/"+reports_to_read["monthly_reports"]])
```

Здесь мы использовали функцию *map()* примерно так же, как делали это в случае с функцией *map_df()* при извлечении данных с помощью языка R. Эта функция для каждого элемента из столбца *monthly_reports* вызывает переданную функцию *read_csv()* из библиотеки *pandas*. В результате мы получаем список датафреймов. В заключение вызываем функцию *concat()* из пакета *pandas* для объединения данных в единый датафрейм.

Шаг 9. Перенесите скрипт в Power BI

Полный скрипт на языке Python представлен ниже:

```
import os
import pandas as pd
from dateutil.relativedelta import relativedelta

os.chdir("<path to where the Python_Code folder is located>")
monthly_reports = os.listdir("./Data/SalesData")

d = {'monthly_reports': monthly_reports}
df = pd.DataFrame(d)
df["date_format"] = pd.to_datetime(
    df.monthly_reports.str.replace(".csv", ""))

min_month = df.date_format.max() - relativedelta(months=23)
reports_to_read = df[df["date_format"]>=min_month]

df_output = pd.concat(
    map(
        pd.read_csv,
        "./Data/SalesData/"+reports_to_read["monthly_reports"])))
```

Скопируйте скрипт на языке Python в буфер обмена и откройте файл с расширением *pbix* для этого примера. На вкладке **Главная** (Home) откройте выпадающую кнопку **Получить данные** (GetData), выберите подменю **Другие** (More) и перейдите на вкладку **Другое** (Other). В правой части окна найдите и выберите пункт **Скрипт Python** (Python script), как показано на рис. 3.3.

После этого в Power BI откроется окно встроенного редактора скриптов Python, показанное на рис. 3.4.

Вставьте скопированный скрипт в это окно и нажмите на кнопку **ОК**. Теперь вы можете обращаться с загруженными посредством скрипта Python данными точно так же, как и с любыми другими. К примеру, вы можете осуществить дополнительное преобразование данных при помощи инструмента Power Query или загрузить информацию непосредственно в модель данных Power BI.

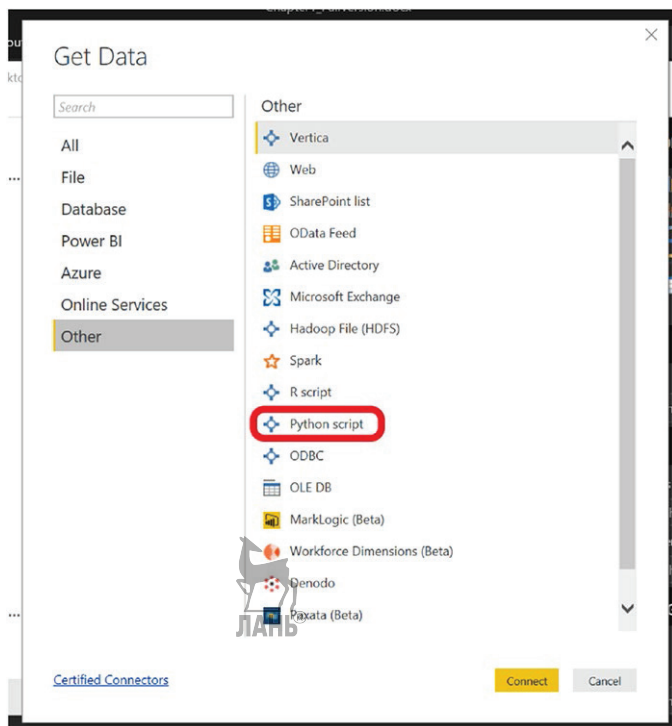


Рис. 3.3 ❖ Открытие редактора скриптов Python

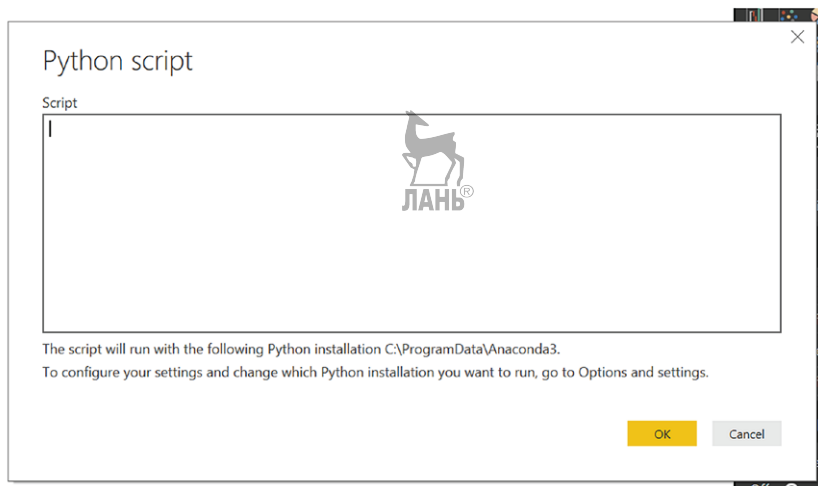


Рис. 3.4 ❖ Редактор скриптов Python в Power BI

В этом разделе мы научились динамически определять, какие файлы входят в скользящий период за последние 24 месяца, при помощи информации об именах файлов, при этом сделали мы это как на языке R, так и на Python.

Получившиеся фрагменты кода оказались намного более простыми и понятными по сравнению с кодом на языке M, который пришлось бы написать для выполнения этой задачи в Power Query.

Вам может никогда не понадобится делать нечто подобное, но вы всегда можете модифицировать представленный в этих примерах код под собственные нужды. Откройтесь тому новому, что могут принести в привычный мир Power BI языки программирования R и Python. Вы уже увидели, как легко можно взаимодействовать с файлами в операционной системе при помощи этих гибких языков. Далее в этой книге вы узнаете и о других полезных возможностях, доступных языкам R и Python. К примеру, в следующем разделе мы посмотрим, как можно легко и просто применять регулярные выражения в языках R и Python для выполнения фильтрации строк в исходном файле CSV.

ФИЛЬТРАЦИЯ СТРОК НА ОСНОВЕ РЕГУЛЯРНЫХ ВЫРАЖЕНИЙ

В большинстве ситуаций вы можете довольно легко исключить строки, нежелательные для импорта в модель данных Power BI. Но бывают случаи, когда логика отбора не позволяет воспользоваться встроенными средствами этого мощного инструмента. Один из таких случаев связан с фильтрацией записей в наборе данных на основании *строковых шаблонов* (string pattern), которые могут быть реализованы только при помощи *регулярных выражений* (regular expression). Power Query не поддерживает использование регулярных выражений напрямую, тогда как в языках R и Python вы можете свободно пользоваться всеми их преимуществами.

Представьте, что вы отвечаете за аналитику при запуске политической кампании. У вас есть файл со списком избирателей, который вам необходимо проанализировать для организации почтовой рассылки. В сферу ваших интересов входят только избиратели с заполненным по всем правилам электронным адресом. Например, строку с электронным адресом *john.doe@email* мы должны будем исключить из загрузки в модель данных Power BI, а строку с адресом *janedoe@email.com* – оставить. В результате выполненной очистки мы должны получить набор данных, состоящий из строк, в которых электронный адрес заполнен правильно, и именно его мы впоследствии импортируем в модель данных Power BI.

Использование регулярных выражений в R

На момент написания данной книги сценарий, который мы рассмотрим в этом разделе, нельзя было реализовать средствами Power Query, поскольку в этом инструменте нативно не поддерживаются регулярные выражения.

Что касается языка R, то в нем регулярные выражения используются давно и очень активно. А код, который вам придется написать для решения этой задачи, будет очень несложным и интуитивно понятным. По традиции пройдемся по шагам, необходимым для решения поставленной задачи.

Шаг 1. Загрузите необходимые для работы пакеты

Первое, что вам необходимо сделать, – это, как и раньше, импортировать необходимые для решения поставленной задачи пакеты. В данном случае нам хватит всего двух библиотек: *tidyverse* и *stringr*. Импортируем их при помощи следующего кода:

```
library(tidyverse)
library(stringr)
```

Примечание. Заметьте, что пакет *stringr* является частью коллекции *tidyverse*, но не входит в ее ядро, которое импортируется при загрузке библиотеки *tidyverse*.

Шаг 2. Загрузите в R файл с потенциальными избирателями

Для этого вам нужно выполнить следующие строки кода:

```
setwd(«путь к папке R_Code»)
goodemails_raw <- read_csv("./Data/EmailAddresses.csv")
```

Здесь вам должно быть все понятно. Сначала мы устанавливаем путь к папке *R_Code* в качестве рабочей директории при помощи знакомой вам функции *setwd()*. После этого используем относительный путь для загрузки исходного файла *EmailAddresses.csv* для последующего анализа.

Шаг 3. Определите регулярное выражение

Используйте приведенный ниже код для определения регулярного выражения:

```
email_pattern <- "^[a-zA-Z][\\w\\_]{4,20})\\@([a-zA-Z0-9.-]+)\\.([a-zA-Z]{2,3})$"
```

Именно это регулярное выражение мы будем использовать как мерилlo правильности заполнения электронного адреса. Пусть вас не пугает кажущаяся сложность этой строки. Даже если вы не умеете писать «регулярки», вы в большинстве случаев можете найти их в интернете под собственные нужды. Помните цитату неизвестного автора: «Хорошие программисты пишут хороший код. Отличные программисты воруют отличный код». Но я бы не назвал поиск типовых регулярных выражений в интернете воровством, ведь очень многие программисты-альтруисты сами с удовольствием выкладывают свои наработки на всеобщее обозрение.

Шаг 4. Исключите неправильные адреса из набора данных

Для этого достаточно выполнить следующий код:

```
goodemails <-
  goodemails_raw %>%
  filter(str_detect(Email, email_pattern) == TRUE)
```

Здесь мы воспользовались функцией `filter()` из пакета `dplyr` для выполнения фильтрации датафрейма `goodemails_raw`. Сама фильтрация производится в функции `str_detect()`, принадлежащей библиотеке `stringr`. Эта функция позволяет выполнить проверку на соответствие строки переданному строковому шаблону. Значение `True` возвращается в случае полного соответствия строки шаблону, в противном случае возвращается значение `False`. Функция `filter()` оставит в наборе данных только те строки, для которых функция `str_detect()` вернет значение `True`.

Шаг 5. Объедините написанный код в один скрипт и перенесите в редактор скриптов в Power BI

Полный скрипт фильтрации данных приведен ниже:

```
library(tidyverse)
library(stringr)

setwd("<полный путь к папке с сопроводительным кодом к главе>")

goodemails_raw <- read_csv("./Data/EmailAddresses.csv")

email_pattern <- "^[a-zA-Z][\\w\\_]{4,20})\\@([a-zA-Z0-9.-]+)\\.([a-zA-Z]{2,3})$"

goodemails <-
  goodemails_raw %>%
  filter(str_detect(Email, email_pattern) == TRUE)
```

Скопируйте скрипт на языке R в буфер обмена и откройте файл с расширением `pbix` для этого примера. На вкладке **Главная** (Home) откройте выпадающую кнопку **Получить данные** (GetData), выберите подменю **Другие** (More) и перейдите на вкладку **Другое** (Other). В правой части окна найдите и выберите пункт **R-скрипт** (R script).

Использование регулярных выражений в Python

Теперь пришла очередь проявить себя при работе с регулярными выражениями языку Python. Как и в случае с R, скрипт на Python будет простым и понятным.

Шаг 1. Загрузите необходимые для работы библиотеки

Первое, что вам необходимо сделать, – это импортировать необходимые библиотеки. Здесь нам хватит всего двух пакетов: *os* и *pandas*. Импортируем их при помощи следующего кода:

```
import os
import pandas as pd
```

Шаг 2. Загрузите в Python файл с избирателями и присвойте его содержимое датафрейму

Для этого вам нужно выполнить следующие строки кода:

```
os.chdir("<путь к папке Python_Code в директории Chapter 3>")
goodemails_raw = pd.read_csv("<путь к EmailAddresses.csv>")
```

Здесь также никаких секретов для вас быть уже не должно. Сначала мы устанавливаем путь к папке *Python_Code* в качестве рабочей директории при помощи функции *os.chdir()*. После этого загружаем файл *EmailAddresses.csv* и присваиваем его содержимое переменной *goodemails_raw*.

Шаг 3. Определите регулярное выражение

Используйте приведенный ниже код для определения регулярного выражения:

```
email_pattern = "^[a-zA-Z][\\w\\_]{4,20})\\@([a-zA-Z0-9-\\.]+)\\.([a-zA-Z]{2,3})$"
```

Это же регулярное выражение мы использовали для проверки правильности ввода электронного адреса и в скрипте R.

Шаг 4. Исключите неправильные адреса из набора данных

Для этого достаточно выполнить следующий код:

```
goodemails = goodemails_raw[
    goodemails_raw["Email"].str.match(email_pattern)]
```

Сначала посредством выражения в квадратных скобках создается последовательность логических значений, которая впоследствии используется для выполнения фильтрации путем оставления только тех строк, в которых булево значение равно *True*.

Шаг 5. Объедините написанный код в один скрипт и перенесите в редактор скриптов Python в Power BI

Полный скрипт фильтрации данных приведен ниже (пути к файлам вам необходимо прописать самостоятельно):

```
import os
import pandas as pd

os.chdir("<путь к папке Python_Code в директории Chapter 3>")
goodemails_raw = pd.read_csv("<путь к EmailAddresses.csv>")

email_pattern = "^[a-zA-Z][\\w\\_]{4,20}\\@([a-zA-Z0-9.-]+)\\.([a-zA-Z]{2,3})$"

goodemails = goodemails_raw[
    goodemails_raw["Email"].str.match(email_pattern)]
```

Скопируйте скрипт на языке Python в буфер обмена и откройте файл с расширением *pbix* для этого примера. На вкладке **Главная** (Home) откройте выпадающую кнопку **Получить данные** (GetData), выберите подменю **Другие** (More) и перейдите на вкладку **Другое** (Other). В правой части окна найдите и выберите пункт **Скрипт Python** (Python script).

Как видите, регулярные выражения очень удобно использовать, когда необходимо произвести сравнение текстовой информации с подготовленными заранее строковыми шаблонами. Это очень полезный инструмент в руках любого аналитика данных. К сожалению, поддержка регулярных выражений пока не реализована в Power Query, зато она успешно реализована в языках Python и R. Здесь мы привели лишь один пример использования регулярных выражений для проверки правильности ввода информации. Далее в этой книге мы посмотрим более сложные сценарии с применением «регулярок». Сейчас же обратимся к теме загрузки информации из рабочих книг Microsoft Excel в модель данных Power BI.



Глава 4

.....

Чтение данных из Microsoft Excel

В Power BI есть встроенные средства доступа к информации, хранящейся в Microsoft Excel, для загрузки ее в модель данных. Эти средства легки в освоении и интуитивно понятны для более или менее простых сценариев, но когда ситуация усложняется, бывает трудно справиться с ней стандартными способами загрузки данных. И в этих случаях вам на помощь могут прийти языки R и Python.

Давайте рассмотрим сценарий со следующими условиями:

- на дворе 2 февраля 2014 года, и вы финансовый директор компании Contoso International;
- вы получаете от отдела ИТ информацию о ежегодных продажах в виде файлов MS Excel, сохраняя их в папке *ExcelFiles*;
- для каждого года создается своя рабочая книга, а данные по месяцам находятся на отдельных листах;
- в папке *ExcelFiles* располагается пять файлов. При этом рабочая книга за 2010 год содержит только данные за декабрь, книги с 2011 по 2013 год полные, а в файле за 2014 год есть информация лишь за январь;
- в начале каждого месяца в книгу добавляется новый лист с информацией о продажах за предыдущий месяц;
- в начале февраля создается новая рабочая книга за текущий год, которая обновляется по примеру книг за предыдущие годы.

Для проведения полного финансового анализа вам необходимо объединить информацию из всех имеющихся файлов. Вы попробовали использовать для этого Power Query, но, не обладая глубокими познаниями в этом инструменте, решили призвать на помощь старичков R и Python. Ваш друг Тайрон неплохо разбирается в этих языках и согласился вам помочь. Он предоставил вам подробную пошаговую инструкцию для решения задачи при помощи обоих языков программирования. Сначала давайте рассмотрим инструкцию для языка R.

ЧТЕНИЕ ФАЙЛОВ EXCEL ПРИ ПОМОЩИ R

В данном примере мы спроектируем рабочий процесс, выполняющий следующие действия.

1. Проход по всем файлам в папке *ExcelFiles*.
2. Объединение данных из всех листов рабочей книги в один датафрейм.
3. Добавление датафрейма, созданного на втором шаге, к основному датафрейму, в котором по окончании работы будет собрана вся информация из всех рабочих книг.

На рис. 4.1 наш рабочий процесс представлен графически.



Рис. 4.1 ❖ Рабочий процесс по объединению данных с листов Excel в единый датафрейм при помощи R

Теперь давайте пройдем по шагам, которые помогут превратить разработанный план действий в рабочий код на языке R.

Шаг 1. Импортируйте пакеты *tidyverse* и *readxl*

Для реализации описанного сценария нам понадобятся пакеты *tidyverse* и *readxl*. С пакетом *tidyverse* вы уже хорошо знакомы. В свою очередь, библиотека *readxl* очень похожа на *readr* с тем лишь исключением, что она предназначена для чтения данных из файлов *Microsoft Excel*. Код, который вам понадобится для импорта необходимых пакетов, приведен ниже:

```
library(tidyverse)
library(readxl)
```

Шаг 2. Создайте оболочку функции `combine_sheets`

На этом шаге мы пропишем остов функции для чтения данных из листов рабочей книги Excel и сбора их в единый датафрейм. Путь к рабочей книге Excel, которую требуется обработать, будет передаваться в функцию в качестве аргумента. Ниже показано, как будет выглядеть структура функции:

```
combine_sheets <- function(excel_file_path) {
  <код на языке R>
  return(<объект R>)
}
```

Шаг 3. Получите имена листов для объединения из указанной рабочей книги

Теперь начнем наполнять созданную ранее функцию смыслом. Нам необходимо определить список листов в рабочей книге для последующего объединения данных. Сделаем это при помощи следующего фрагмента кода:

```
df <- excel_file_path %>%
  excel_sheets()
```

Код начинается с обращения к пути, указывающему на рабочую книгу Excel, из которой необходимо извлечь данные. Этот путь хранится в переменной `excel_file_path`, которая передается в качестве первого параметра функции `excel_sheets()` из пакета `readxl` с использованием знакомого вам оператора конвейера (`%>%`). Функция `excel_sheets()` возвращает символьный вектор, содержащий имена листов из рабочей книги, расположенной по пути, обозначенному переменной `excel_file_path`. Если вам необходима дополнительная информация о работе оператора конвейера, обратитесь к главе 2 данной книги.

Шаг 4. Преобразуйте символьный вектор, полученный на предыдущем шаге, в именованный символьный вектор

На данном шаге нам необходимо трансформировать символьный вектор, полученный на шаге 3, в *именованный символьный вектор* (named character vector). Свойство `name` именованного вектора будет использовано для идентификации рабочего листа, из которого берутся строки для результирующего датафрейма. Именованный вектор создается при помощи следующего кода:

```
df <- excel_file_path %>%
  excel_sheets() %>%
  set_names() %>%
```



Здесь мы берем символьный вектор, созданный с помощью функции `excel_sheets()`, и передаем его в функцию `set_names()` из пакета `purrr`. Эта функция устанавливает имена для всех элементов исходного символьного вектора. Чтобы лучше понять принципы работы функции, можно рассмотреть содержимое датафрейма, находящегося в переменной `df`, до и после вызова функции `set_names()`. Так выглядел датафрейм до выполнения именования:

```
> df
[1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
```

А так он стал выглядеть после вызова функции `set_names()`:

```
> df
  Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
"Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
```

Обратите внимание, что каждому элементу в символьном векторе было присвоено свое имя, совпадающее со значением элемента. На следующем шаге мы воспользуемся этим свойством нового вектора.

Примечание. Вы не обязаны именовать элементы вектора в соответствии с их значениями. Вместо этого вы можете передать функции символьный вектор той же длины с именами, которые вы хотите использовать для элементов, или функцию на случай, если имена у элементов уже есть и вы хотите модифицировать их согласно какому-то правилу.

Шаг 5. Используйте функцию `map_dfr()` для объединения информации с листов в один датафрейм

На этом шаге мы добавим в наш код лишь одну строку, но она будет выполнять массу работы. Вот как будет выглядеть наш итоговый фрагмент кода:

```
df <- excel_file_path %>%
  excel_sheets() %>%
  set_names() %>%
  map_dfr(.f = ~read_excel(path = excel_file_path, sheet = ..1)
    , .id = "sheet"
  )
```

Давайте посмотрим, что происходит в этой последней строке кода:

- именованный символьный вектор, созданный на предыдущем шаге, передается в функцию `map_dfr()` в качестве первого аргумента;
- аргумент `.f` служит для указания действия, применяемого к переданному в функцию `map_dfr()` именованному символьному вектору. В данном случае мы указали функцию `read_excel()`. Описание того, как именно должна использоваться функция `read_excel()`, содержится в формуле (formula), начинающейся со знака `~` (тильда), после которого следует

название функции и ее аргументы. Аргументами функции `read_excel()` являются `path` и `sheet`. В качестве первого мы передаем путь к рабочей книге, содержащей листы для объединения, а в качестве второго – значение из именованного символьного вектора. Здесь мы передали выражение `..1` – такой подход позволяет обращаться к аргументу по номеру его позиции. Именованный символьный вектор, возвращенный функцией `set_names()`, был передан в функцию `map_dfr()` в качестве первого аргумента, поэтому мы можем обращаться к нему по индексу `..1`. Если бы он располагался на *n*-й позиции, мы бы использовали нотацию `..n`;

- функция `map_dfr()` объединяет датафреймы, которые были созданы для каждого листа в рабочей книге, в единый датафрейм. Можно добавить к результирующему датафрейму столбец для определения того, из какого листа была перенесена строка. Мы сделали это при помощи аргумента `.id`. Если вы используете эту опцию, будет создан столбец, заполненный именами из именованного символьного вектора. Если в качестве источника используется неименованный символьный вектор, будет задействован порядковый номер имени листа в векторе. Значение, которое вы передадите этому аргументу, будет использовано в качестве имени столбца.

Шаг 6. Верните датафрейм из функции

Мы хотим, чтобы наша функция возвращала созданный датафрейм. Для этого используем инструкцию `return`, которой на вход передадим наш датафрейм. Полное определение функции будет выглядеть следующим образом:

```
combine_sheets <- function(excel_file_path) {
  df <- excel_file_path %>%
    excel_sheets() %>%
    set_names() %>%
    map_df(.f = ~read_excel(path = excel_file_path, sheet = ..1)
      ,.id = "sheet"
    )
  return(df)
}
```

Шаг 7. Направьте рабочую директорию на папку с файлами Excel

Итак, сложная часть задачи позади – функция создана. Теперь нам нужно написать вспомогательный код для использования нашей функции `combine_sheets()`, которая поможет объединить данные из рабочих книг, находящихся в папке `ExcelFiles`. Сначала установим рабочую директорию, чтобы Power BI

знал, какую папку использовать. Как всегда, это можно сделать следующим образом:

```
setwd("<полный путь к папке ExcelFiles>")
```

Шаг 8. Сохраните в переменной `excel_file_paths` список файлов для обработки

Мы используем функцию `list.files()` для получения списка имен файлов в текущей папке и сохраним его в переменной `excel_file_paths` следующим образом:

```
excel_file_paths <- list.files(".")
```

Примечание. Точка, переданная в качестве аргумента функции `list.files()`, говорит ей о том, что необходимо вернуть все файлы из текущей рабочей директории. Можно указать путь ко вложенной папке при помощи шаблона «./<вложенная папка>». Функция `list.files()` также может принимать символьный вектор, содержащий один или больше полных путей, которые необходимо обработать.

Шаг 9. Используйте функцию `map_dfr()` для применения функции `combine_sheets()` ко всем выбранным файлам

Вот мы и добрались до шага, на котором соберем данные из всех нужных нам файлов в один датафрейм. Воспользуемся для этого следующим фрагментом кода:

```
combine_workbooks <- map_dfr(excel_file_paths, combine_sheets)
```

Функция `map_dfr()` применит написанную нами ранее функцию `combine_sheets()` к именам файлов Excel, содержащимся в переменной `excel_file_paths`. Результатом этой операции будет набор датафреймов, собранных функцией `map_dfr()` в единый датафрейм, который будет присвоен переменной `combine_workbooks`.

Шаг 10. Скопируйте скрипт и вставьте в редактор скриптов R в Power BI через инструмент Получить данные (GetData)

Итоговый скрипт у нас получился такой:

```
library(tidyverse)
library(readxl)
```

```

combine_sheets <- function(excel_file_path){
  df <- excel_file_path %>%
    excel_sheets() %>%
    set_names() %>%
    map_dfr(.f = ~read_excel(path = excel_file_path, sheet = ..1)
      ,.id = "sheet"
    )
  return(df)
}

setwd("<полный путь к папке ExcelFiles>")
excel_file_paths <- list.files(".")
combine_workbooks <- map_dfr(excel_file_paths, combine_sheets)

```

После запуска скрипта в редакторе R в Power BI результирующий набор данных предстанет перед вами как любой другой, и вы сможете с ним работать как обычно.

ЧТЕНИЕ ФАЙЛОВ EXCEL ПРИ ПОМОЩИ PYTHON

Теперь выполним те же действия при помощи языка Python. Сценарий и последовательность действий будут очень похожими, но с некоторыми нюансами, характерными для языка Python. На рис. 4.2 схематически показан весь процесс.

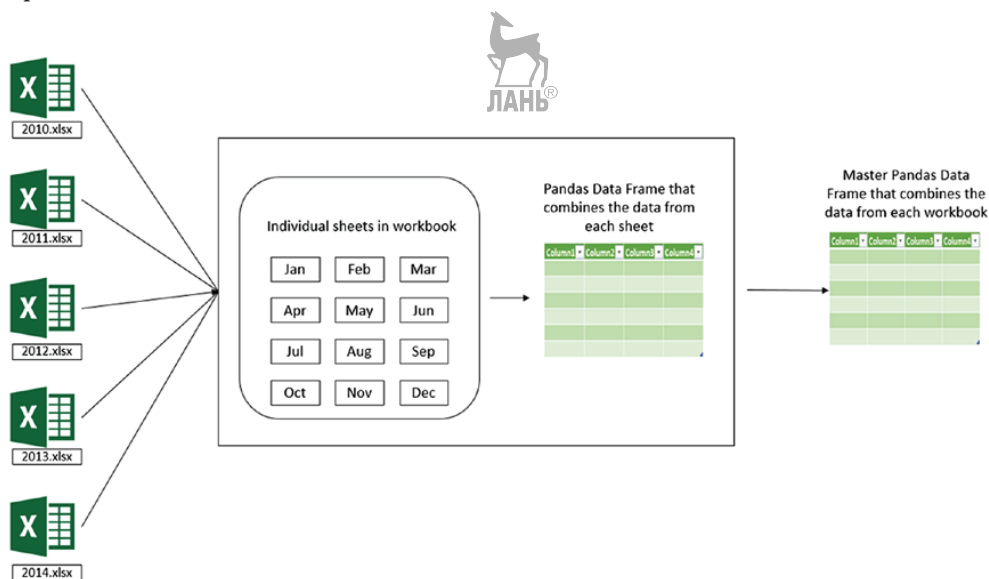


Рис. 4.2 ❖ Рабочий процесс по объединению данных с листов Excel в единый датафрейм при помощи Python

Шаг 1. Импортируйте библиотеки *os* и *pandas*

Для решения поставленной задачи в Python нам потребуются пакеты *os* и *pandas*. Пакет *os* служит для взаимодействия с файловой системой, а *pandas* – для чтения данных из Excel в Python.

Шаг 2. Создайте оболочку функции `combine_sheets()`

Вам понадобится написать функцию для считывания информации из файлов Excel и объединения ее в один датафрейм. На вход функция должна принимать путь к рабочей книге Excel, содержащей листы, данные с которых необходимо объединить. Заготовка для функции будет выглядеть следующим образом:

```
def combine_sheets(excel_file_path):  
    <код Python>  
    return <объект Python>
```

Шаг 3. Создайте объект Excel на основании пути, переданного в функцию в аргументе `excel_file_path`

Мы определили оболочку функции, теперь перейдем к ее наполнению. Начнем с создания объекта Excel посредством следующего кода:

```
xlsx_file = pd.ExcelFile(excel_file_path)
```

Для создания объекта мы используем класс *ExcelFile* из библиотеки *pandas*. В рамках его инициализации передадим на вход путь к рабочей книге Excel.

Шаг 4. Создайте список имен листов в рабочей книге

Функции необходимо знать, какие рабочие листы из книги объединять. Эту информацию можно получить посредством свойства `sheet_names` экземпляра объекта *xlsx_file*, созданного на предыдущем шаге:

```
ws = xlsx_file.sheet_names
```

В этом коде мы извлекаем список листов для объединения из объекта *xlsx_file* и сохраняем в переменной *ws*.

Шаг 5. Используйте метод `read_excel()` из библиотеки `pandas` для считывания данных в один датафрейм

Код, который мы добавим на этом шаге, будет выглядеть так:

```
df = pd.concat(pd.read_excel(xlsx_file, sheet_name=ws))
```

Здесь выполняется сразу две операции, и будет проще для понимания, если мы разобьем код на части. Фрагмент кода, располагающийся в скобках (`pd.read_excel(xlsx_file, sheet_name=ws)`), предназначен для чтения данных с листов Excel и объединения их в один датафрейм. Первый аргумент *io* служит для передачи файла Excel, из которого требуется считать информацию. Посредством второго аргумента (*sheet_name*) мы говорим методу `read_excel()`, из каких листов необходимо взять данные. Эту информацию мы передаем при помощи переменной *ws*.

Вторая операция, которая здесь выполняется, – это объединение нескольких датафреймов, созданных с помощью метода `read_excel()`, в один датафрейм. И здесь нам на помощь приходит метод `concat()` из библиотеки `pandas`. Результирующий датафрейм присваивается переменной *df*.

Шаг 6. Верните датафрейм *df* из функции `combine_sheets`

На этом шаге мы просто воспользуемся инструкцией `return` для возврата датафрейма из нашей функции:

```
return df
```



Шаг 7. Установите рабочую директорию в папку, в которой находятся файлы Excel

На этом этапе мы завершили работу над функцией `combine_sheets()`. Теперь пришло время заняться сопроводительным кодом для вызова нашей функции с целью объединения данных из нескольких файлов Excel, располагающихся в папке *ExcelFiles*. Сначала направим рабочую директорию на папку, содержащую нужные нам рабочие книги. Сделаем это с помощью привычного кода:

```
os.chdir("<полный путь к папке ExcelFiles>")
```

Шаг 8. Получите список файлов в текущей рабочей директории и присвойте его переменной `excel_file_paths`



Наша функция должна знать, где находятся файлы Excel для обработки. Получить список имен рабочих книг мы можем с помощью функции `listdir()`, как показано в следующем фрагменте кода:

```
excel_file_paths = os.listdir(".")
```

В данном случае мы предполагаем, что нужные вам файлы Excel располагаются в текущей рабочей директории. В этом необходимо убедиться перед запуском скрипта.

Шаг 9. Создайте пустой датафрейм и назовите его `combined_workbooks`

На этом шаге мы создадим в переменной `combined_workbooks` пустой датафрейм, в котором в дальнейшем будем собирать обобщенные данные из всех рабочих книг. Для создания пустого датафрейма `pandas` необходимо выполнить следующую строку кода:

```
combined_workbooks = pd.DataFrame()
```

Датафреймы, созданные при помощи нашей функции `combine_sheets()`, будут впоследствии присоединены к нашему главному датафрейму.

Шаг 10. Создайте заготовку для цикла `for`

Цикл `for` в Python, как и во многих языках программирования, позволяет осуществить итерации по определенной последовательности. В данном случае мы будем проходить по списку `excel_file_paths`, последовательно извлекая путь к очередному файлу рабочей книги. Код для осуществления итераций будет следующий:

```
for excel_file_path in excel_file_paths:
```

Вы будете часто использовать циклы `for` в языке Python, так что давайте остановимся на структуре этой конструкции подробнее. Шаблон цикла `for` в общем виде можно описать так:



```
for <переменная для текущего значения> in <объект последовательности>:
```

Переменная, в которой будет храниться текущее значение последовательности, может быть названа как угодно. Обычно принято называть ее так же, как объект последовательности, но в единственном числе. Объект последовательности, в свою очередь, представляет собой список элементов для итераций. В нашем случае он представлен переменной *excel_file_paths*. Давайте пройдемся пошагово и посмотрим, как это работает.

Если вывести на экран содержимое переменной *excel_file_paths*, мы увидим следующее:

```
['2010.xlsx', '2011.xlsx', '2012.xlsx', '2013.xlsx', '2014.xlsx']
```

Как видите, в переменной *excel_file_paths* находится список из пяти элементов. Цикл *for* проходит по этому списку и на каждом шаге присваивает переменной *excel_file_path* очередное значение: «2010.xlsx», «2011.xlsx», «2012.xlsx» и т. д.

Шаг 11. Объедините данные со всех листов в один датафрейм при помощи функции *combine_sheets()*



На этом шаге мы берем переменную *excel_file_path* и передаем ее функции *combine_sheets()*, как показано ниже:

```
combined_workbook = combine_sheets(excel_file_path)
```

В написанной нами ранее функции *combine_sheets()* происходит объединение данных из всех листов в рабочей книге, путь к которой указан в переданном параметре, в один датафрейм. Этот датафрейм сохраняется в переменной *combined_workbook*.

Шаг 12. Добавьте датафрейм *combined_workbook* к главному датафрейму *combined_workbooks*

Код, необходимый для выполнения этой операции, показан ниже:

```
combined_workbooks = combined_workbooks.append(combined_workbook, ignore_index=True)
```

Здесь мы используем метод *append()* объекта *combined_workbooks* для присоединения к нему датафрейма *combined_workbook*. Аргумент *ignore_index* мы установили в значение *True*, поскольку хотим игнорировать исходные индексы из датафрейма *combined_workbook* и использовать те, что есть в *combined_workbooks*.

Шаг 13. Скопируйте скрипт и вставьте в редактор скриптов Python в Power BI через инструмент Получить данные (GetData)



Итоговый скрипт у нас получился такой:

```
import os
import pandas as pd

def combine_sheets(excel_file_path):
    xlsx_file = pd.ExcelFile(excel_file_path)
    ws = xlsx_file.sheet_names
    df = pd.concat(pd.read_excel(xlsx_file, sheet_name=ws))
    return df

os.chdir("<полный путь к папке SalesData_WorkbookFormat")
excel_file_paths = os.listdir(".")
combined_workbooks = pd.DataFrame()

for excel_file_path in excel_file_paths:
    combined_workbook = combine_sheets(excel_file_path)
    combined_workbooks = combined_workbooks.append(combined_workbook, ignore_index=True)
```

После запуска скрипта в редакторе Python в Power BI результирующий набор данных будет таким же, как любой другой, и вы сможете загрузить его в модель данных.

В этой главе мы продемонстрировали реализацию сценария, который довольно трудно было бы решить стандартными средствами Power Query. Код на языках R и Python, который нам потребовалось написать для этого, оказался гораздо проще, чем скрипт на языке M, который мы вынуждены были бы писать при загрузке данных с помощью Power Query. Кроме того, вы узнали, как при помощи R и Python можно выделять логику в отдельные функции с целью повышения эффективности и простоты восприятия кода. В следующей главе мы посмотрим, как можно в Power BI взаимодействовать с данными, хранящимися в *SQL Server*, посредством языков R и Python.

Глава 5

.....

Чтение данных из SQL Server



Встроенное в Power BI средство загрузки и обработки информации под названием Power Query позволяет импортировать в память данные в самых разных форматах, обрабатывать и преобразовывать их, а также загружать в модель данных. Это поистине потрясающий инструмент, помогающий в большинстве ситуаций, связанных с необходимостью загрузки и трансформации данных.

Но, к сожалению, существуют области, в которых встроенные средства Power BI оказываются бессильны. Power Query создан для чтения данных, а не для их записи. Но это ограничение не свойственно языкам программирования R и Python. В их составе есть все необходимые пакеты и библиотеки для ведения логов. И R, и Python способны довольно легко записывать информацию в разные источники – от простейших плоских файлов до полноценных баз данных. В примерах из этой главы мы будем использовать R и Python для сохранения информации о загрузках в базе данных SQL Server.

ДОБАВЛЕНИЕ БАЗЫ ДАННЫХ ADVENTUREWORKSDW_STARSCHEMA К ВАШЕМУ ЭКЗЕМПЛЯРУ SQL SERVER

Примеры из этой главы требуют, чтобы у вас была в наличии база данных *AdventureWorksDW_StarSchema*, которая является составной частью базы данных *AdventureWorksDW*, разработанной компанией Microsoft. Выполните следующие действия, чтобы восстановить базу из бэкапа в своем экземпляре SQL Server.

1. Откройте репозиторий данной книги и перейдите в папку *Data*, соответствующую пятой главе. Скачайте zip-архив файла *AdventureWorksDW_StarSchema.bak* и разархивируйте его.
2. Скопируйте файл *AdventureWorksDW_StarSchema.bak* в папку *Backup* в SQL Server. В виртуальной машине для анализа данных (Data Science

Virtual Machine – DSVM) эта папка находится в директории C:\Program Files\Microsoft SQL Server\MSSQL15.MSSQLSERVER\MSSQL\Backup.

3. Откройте *SQL Server Management Studio (SSMS)*, щелкните правой кнопкой мыши по папке **Базы данных** (Databases) и выберите пункт **Восстановить базу данных** (Restore Database).
4. Откроется диалоговое окно **Восстановление базы данных** (Restore Database). На вкладке **Общие** (General) в разделе **Источник** (Source) установите переключатель в положение **Устройство** (Device).
5. Нажмите на кнопку с тремя точками, а в открывшемся окне – на кнопку **Добавить** (Add). Укажите путь к файлу *AdventureWorksDW_StarSchema.bak*.
6. Выберите файл *AdventureWorksDW_StarSchema.bak* и нажмите на кнопку **ОК**, чтобы закрыть диалоговое окно **Локальный файл резервной копии** (Locate Backup File).
7. Нажмите на кнопку **ОК** для закрытия окна **Выбор устройств резервного копирования** (Select backup devices).
8. Нажмите кнопку **ОК** в форме **Восстановить базу данных** (Restore Database).



ЧТЕНИЕ ДАННЫХ ИЗ SQL SERVER В POWER BI ПРИ ПОМОЩИ R



Вот вам новый сценарий. Вы занимаетесь аналитикой в компании *Clothing R Us*. У компании есть небольшой отдел ИТ и администратор баз данных, попутно выполняющий функции системного инженера. Отдельного департамента, который бы занимался аналитикой, в компании нет. Вы убедили финансового директора в необходимости вести бизнес-аналитику с целью принятия взвешенных решений на основании опыта компании. Она согласилась организовать небольшую витрину данных с информацией об интернет-продажах. В витрине содержится одна таблица фактов и четыре *измерения первого типа* (Type 1 dimension), а загрузка данных из хранилища выполняется по принципу «kill and fill», подразумевающему удаление старой информации и запись новой. Администратор баз данных наладил процесс импорта и преобразования данных, не предусматривающий хранение логов об объеме загруженной информации в витрину. Но для вас эти сведения критически важны, и вы решаете самостоятельно настроить систему ведения логов, чтобы можно было с уверенностью говорить о том, что загрузка в модель данных Power BI была выполнена корректно.

Примечание. *Медленно меняющиеся измерения первого типа* (Type 1 Slowly Changing Dimension – SCD) подразумевают полную перезапись старых данных новыми, в результате чего история изменений не хранится. При использовании второго типа измерений изменяемые данные также сохраняются в исходной таблице, чтобы при необходимости к ним можно было получить доступ.

Вы знаете, что Power Query прекрасно справляется с задачей чтения данных в Power BI, но запись не входит в его функции. Ваша коллега рассказала вам о том, что вы можете легко и просто организовать запись информации в базу данных посредством языков R или Python. Вы передали ей сведения о том, какие данные вам необходимо записывать, а она в ответ сопроводила вас подробной инструкцией по использованию в этих целях языков R и Python. Сначала пройдем по шагам реализации сценария при помощи языка R.

Шаг 1. Создайте DSN для подключения к базе данных SQL Server

Чтобы подключиться к базе данных, SQL Server необходимо предоставить информацию об аутентификации. Существует масса способов для подключения к SQL Server, здесь мы будем использовать подключение с помощью DSN (Data Source Name). Это относительно простой способ, который легко реализовать в Windows 10. Для этого вам необходимо выполнить следующие действия.

1. Откройте строку поиска в Windows и введите *administrative tools* или *odbc*.
2. Вы увидите список инструментов администрирования. Найдите пункт **Источники данных ODBC (64-разрядная версия)** (ODBC Data Sources (64-bit)) и нажмите на него.
3. Откроется диалоговое окно **Администратор источника данных ODBC (64-разрядная версия)** (ODBC Data Source Administrator (64-bit)). Перейдите на вкладку **Системный DSN** (System DSN). Мы будем использовать источники данных на этой вкладке, чтобы все пользователи компьютера имели доступ к нашим DSN. Если вам достаточно, чтобы доступ был только у текущего пользователя, можно ограничиться работой на вкладке **Пользовательский DSN** (User DSN).
4. Нажмите на кнопку **Добавить** (Add).
5. Вы увидите перечень доступных драйверов. Найдите драйвер *SQL Server* и нажмите на кнопку **Готово** (Finish).
6. Откроется диалоговое окно **Создание источника данных для SQL Server** (Create a New Data Source to SQL Server). Именно здесь производится настройка DSN. В поле **Имя** (Name) необходимо ввести наименование DSN, которое будет использоваться в R. В поле **Описание** (Description) вы можете ввести любое описательное название подключения, а в поле **Сервер** (Server) необходимо указать имя SQL Server, в котором располагается нужная вам база данных. Используйте в качестве имени и описания строку *SQLServer2019*. Если у вас другая версия SQL Server, можете отразить это в названии DSN. На рис. 5.1 показан внешний вид подключения к SQL Server 2017. Далее вам необходимо указать, какой экземпляр SQL Server вы собираетесь использовать. Чтобы узнать имя нужного вам экземпляра, откройте *SQL Server Mana-*

gement Studio и выполните подключение. Наименование, указанное в выпадающем списке **Имя сервера** (Server Name), и будет тем именем, которое необходимо ввести в поле **Сервер** (Server) при создании DSN. После заполнения формы ваше диалоговое окно будет выглядеть так, как показано на рис. 5.1. Нажмите на кнопку **Далее** (Next).

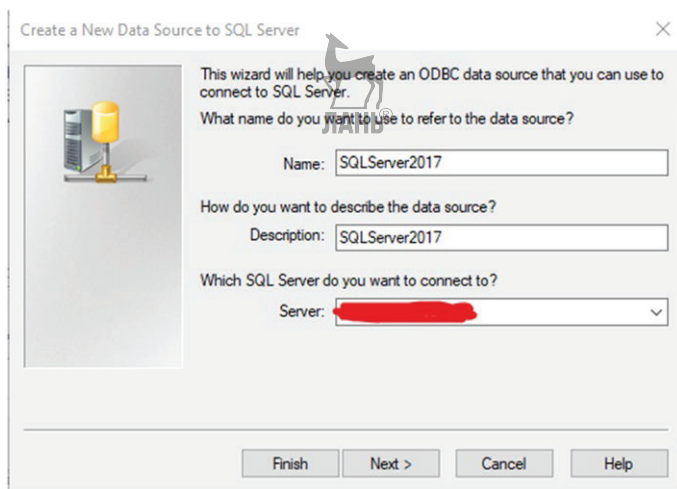


Рис. 5.1 ❖ Создание DSN для SQL Server

7. Вы увидите диалоговое окно, показанное на рис. 5.2. Убедитесь, что у вас стоят такие же настройки, и нажмите на кнопку **Далее** (Next).

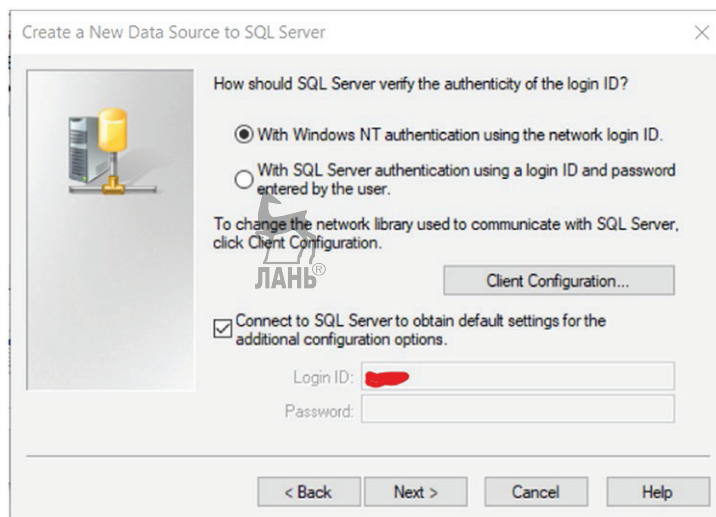


Рис. 5.2 ❖ Окно Создание источника данных для SQL Server

8. Откроется диалоговое окно, показанное на рис. 5.3. Установите флажок **Использовать по умолчанию базу данных** (Change the default database to) и укажите имя базы данных *AdventureWorksDW_StarSchema*. Нажмите на кнопку **Далее** (Next).

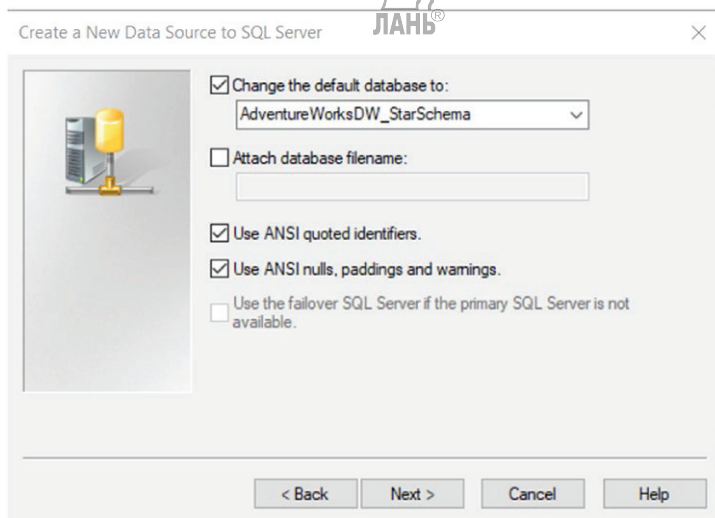


Рис. 5.3 ❖ Указание базы данных по умолчанию

9. После этого откроется окно, изображенное на рис. 5.4. Убедитесь, что вы выбрали такие же настройки, и нажмите на кнопку **Готово** (Finish).

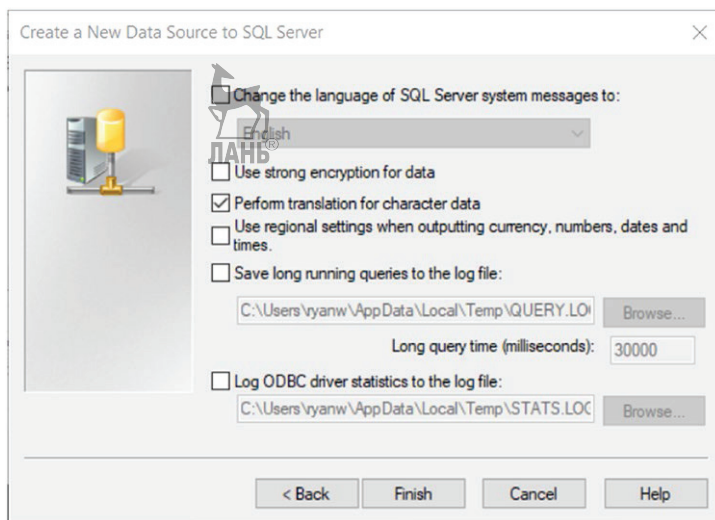


Рис. 5.4 ❖ Финальные настройки DSN

10. В завершение откроется окно **Прогр. установки ODBC для SQL Server**, показанное на рис. 5.5. Проверить созданное подключение можно, нажав на кнопку **Проверить источник данных** (Test Data Source).

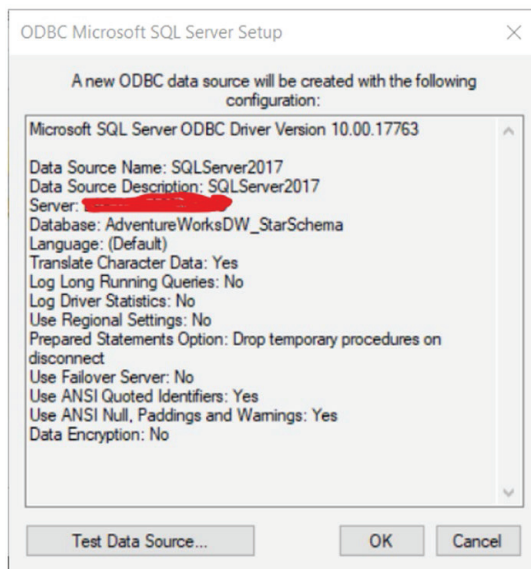


Рис. 5.5 ❖ Сводная информация о созданном DSN

Шаг 2. Создайте таблицу лога в SQL Server

Таблица с именем *dbo.LoadHistoryLog* должна быть в базе данных. Если вам нужно создать ее, вы можете использовать следующий скрипт в *SQL Server Management Studio*:

```
USE AdventureWorksDW_StarSchema
```

```
CREATE TABLE dbo.[LoadHistoryLog] (
    DATETIME DATETIME,
    TABLENAME VARCHAR(25),
    NUM_RECORDS INT
)
```

Чтобы запустить этот скрипт на языке SQL в *SQL Server Management Studio*, нажмите на кнопку **Создать запрос** (New Query) на панели инструментов, после чего будет открыто окно редактора. Вставьте скопированный запрос и нажмите на кнопку **Выполнить** (Execute), чтобы запустить его.

Шаг 3. Начните написание скрипта на R для загрузки таблицы DimDate

Вернитесь в R Studio для создания скрипта на языке R. Пакеты, которые вам понадобятся при решении этого сценария, – это *RODBC* и *lubridate*. Первый мы будем использовать для взаимодействия с SQL Server, а второй – для облегчения работы с датами. Загрузить эти пакеты можно следующим образом:

```
library(RODBC)
library(lubridate)
```

Шаг 4. Создайте переменную для хранения имени загружаемой таблицы

Имя таблицы, которую мы будем загружать в модель данных Power BI, будет использоваться в нескольких фрагментах скрипта. С целью облегчения поддержки скрипта создадим специальную переменную, которую сможем использовать повторно:

```
table_name <- "DimDate"
```

Шаг 5. Создайте переменную для хранения SQL-выражения

Следующий код будет использован для создания выражения на языке SQL, позволяющего загрузить таблицу *DimDate* в Power BI:

```
sql <- paste0(
  "SELECT * FROM dbo.",
  table_name)
```

Здесь мы вызываем функцию *paste0()* для объединения строки запроса «SELECT * FROM dbo.» с переменной *table_name*. Функция *paste0()* работает аналогично функции *paste()*, за исключением того, что в нее не передается разделитель, – предполагается, что его быть не должно.

Шаг 6. Создайте подключение к SQL Server

Подключение к базе данных можно осуществить с помощью следующего фрагмента кода:

```
conn <- odbcConnect(dsn = "SQLServer2017")
```

Как видите, с использованием DSN можно довольно легко подключаться к базе данных SQL Server. Мы воспользовались функцией `odbcConnect()` из пакета *RODBC* для создания подключения, а в качестве единственного аргумента передали имя созданного на первом шаге DSN.

Шаг 7. Извлеките данные из SQL Server и сохраните их в датафрейм

Функция `sqlQuery()` из пакета *RODBC* позволяет отправить запрос базе данных, используя объект подключения `conn`, и вернуть результат в виде датафрейма. Для этого необходимо выполнить следующий код:

```
df_read <- sqlQuery(channel = conn, query = sql)
```

Результаты запроса сохраняются в переменной `df_read`. Эти данные впоследствии будут перенаправлены в Power BI.

Шаг 8. Получите текущее время

Давайте извлечем информацию, которую мы хотим сохранять в виде логов о загрузке в базе данных. Это будет время производства загрузки, имя таблицы, которую загружаем, и количество записей.

Получить текущее время в R можно, воспользовавшись функцией `now()` из пакета *lubridate*. Присвоим его переменной `datestamp`, выполнив следующий код:

```
datestamp <- now()
```

Шаг 9. Получите количество прочитанных записей

В наши планы входит сохранение информации о том, сколько записей было прочитано из таблицы *DimDate*. Получим ее при помощи базовой функции R `nrow()` и сохраним в переменную `num_records`:

```
num_records <- nrow(df_read)
```

Шаг 10. Добавьте в датафрейм запись с информацией для сохранения в лог

Мы извлекли всю необходимую информацию для записи в базу данных. Теперь необходимо привести данные в надлежащий вид для сохранения в таблице, для чего оформим их в виде датафрейма. Создадим *именованный список* (named list) с информацией для загрузки, которую мы заранее распределили по переменным.

Примечание. Именованный список представляет собой обычный список, каждому элементу которого присвоено имя. Сохранение данных в именованный список облегчает создание на их основании датафрейма. В коде, приведенном ниже, показано, как это можно сделать.

Для сохранения информации из переменных в именованный список нужно выполнить следующую инструкцию:

```
list_insert = list("DATESTAMP" = datestamp,
                  "TABLENAME" = table_name,
                  "NUM_RECORDS" = num_records
                )
```

Итак, мы создали именованный список со *скалярами* (scalar), представляющими время чтения, имя таблицы и количество записей.

Примечание. Скаляр представляет собой вектор с единичной длиной.

Далее нам необходимо преобразовать именованный список в датафрейм при помощи следующего кода:

```
df_insert = as_data_frame(list_insert)
```

Здесь мы воспользовались функцией *as_data_frame()* для преобразования созданного ранее именованного списка в датафрейм и сохранения в переменной *df_insert*. Имена из списка будут использованы в качестве заголовков столбцов.

Шаг 11. Сохраните собранную информацию в базе данных

Теперь, когда мы получили в виде датафрейма информацию для записи в базу данных, воспользуемся функцией *sqlSave()* из пакета *RODBC*, чтобы добавить запись в таблицу *LoadHistoryLog*. Это можно сделать следующим образом:

```
sqlSave(
  channel = conn,
  dat = df_insert,
  tablename = "LoadHistoryLog",
  append = TRUE,
  rownames=FALSE
)
```

Аргумент *channel* в функции *sqlSave()* отвечает за подключение к базе данных, которое мы создали ранее. В результате выполнения этой функции добавляется запись из датафрейма *df_insert* в таблицу *LoadHistoryLog*. При этом очень важно установить значение аргумента *append* в TRUE, чтобы функция *sqlSave()* понимала, что запись осуществляется в таблицу, которая уже су-

ществует. Если установить значение FALSE, будет возвращена ошибка, поскольку функция попытается создать в базе данных таблицу, которая уже существует. Также очень важно передать значение FALSE в аргумент *rownames*. Если этого не сделать, будет создано поле с информацией об именах строк в датафрейме, а нам это ни к чему.

Шаг 12. Закройте соединение

В завершение вам необходимо закрыть подключение, которое использовалось для доступа к базе данных. Для этого нужно воспользоваться функцией *odbcClose()* из пакета *RODBC* следующим образом:

```
odbcClose(conn)
```



Шаг 13. Скопируйте написанный скрипт в Power BI

Ниже представлен полный скрипт, который мы создали на предыдущих шагах:

```
library(tidyverse)
library(RODBC)
library(lubridate)

table_name <- "DimDate"
sql <- paste0(
  "SELECT * FROM dbo.",
  table_name)

conn <- odbcConnect(dsn = "SQLServer2017")
df_read <- sqlQuery(channel = conn, query = sql)

timestamp <- now()
num_records <- nrow(df_read)

list_insert <- list(
  "DATESTAMP" = timestamp,
  "TABLENAME" = table_name,
  "NUM_RECORDS" = num_records)

df_insert <- as_data_frame(list_insert)

sqlSave(
  channel = conn,
  dat = df_insert,
  tablename = "LoadHistoryLog",
  append = TRUE,
  rownames=FALSE)

odbcClose(conn)
```



Как и в предыдущем примере, вам необходимо скопировать этот скрипт в Power BI посредством инструмента **Получить данные** (GetData). В нашем коде создаются два датафрейма – *df_read* и *df_insert*, – и вам нужно сообщить Power BI, какой из них вы хотите загрузить в модель данных. Нам нужно загрузить датафрейм *df_read*, так что выберите его и нажмите на кнопку **Изменить** (Edit). Теперь переименуйте запрос из *df_read* в *DimDate* и нажмите на кнопку **Закрыть и применить** (Close & Apply), чтобы загрузить информацию в модель данных Power BI.

Шаг 14. Создайте скрипт для загрузки таблицы DimProduct на базе ReadLog_DimDate.R

Теперь вам необходимо воспроизвести эти шаги еще для четырех таблиц. Это можно легко сделать, скопировав скрипт *ReadLog_DimDate.R* несколько раз и произведя нужные изменения. В данном случае необходимо изменить переменную *table_name*, присвоив ей значение «DimProduct». Затем скопируйте и вставьте скрипт в Power BI, как показано на предыдущем шаге, с тем лишь отличием, что запрос должен называться *DimProduct*. После произведения модификаций код будет выглядеть так, как показано ниже:

```
library(tidyverse)
library(RODBC)
library(lubridate)

table_name <- "DimProduct"
sql <- paste0(
  "SELECT * FROM dbo.",
  table_name)

conn <- odbcConnect(dsn = "SQLServer2017")
df_read <- sqlQuery(channel = conn, query = sql)

datestamp <- now()
num_records <- nrow(df_read)

list_insert <- list(
  "DATESTAMP" = datestamp,
  "TABLENAME" = table_name,
  "NUM_RECORDS" = num_records)

df_insert <- as_data_frame(list_insert)

sqlSave(
  channel = conn,
  dat = df_insert,
  tablename = "LoadHistoryLog",
  append = TRUE,
  rownames=FALSE)

odbcClose(conn)
```



Шаг 15. Создайте скрипт для загрузки таблицы DimPromotion

Создайте еще одну копию скрипта *ReadLog_DimDate.R* и измените значение переменной *table_name* с «DimDate» на «DimPromotion». Затем снова скопируйте его в Power BI так, как было показано на шаге 13, не забыв переименовать запрос *df_read* в *DimPromotion*. Код должен получиться таким:

```
library(tidyverse)
library(RODBC)
library(lubridate)

table_name <- "DimPromotion"
sql <- paste0(
  "SELECT * FROM dbo.",
  table_name)

conn <- odbcConnect(dsn = "SQLServer2017")
df_read <- sqlQuery(channel = conn, query = sql)

datestamp <- now()
num_records <- nrow(df_read)

list_insert <- list(
  "DATESTAMP" = datestamp,
  "TABLERNAME" = table_name,
  "NUM_RECORDS" = num_records)

df_insert <- as_data_frame(list_insert)

sqlSave(
  channel = conn,
  dat = df_insert,
  tablename = "LoadHistoryLog",
  append = TRUE,
  rownames=FALSE)

odbcClose(conn)
```

Шаг 16. Создайте скрипт для загрузки таблицы DimSalesTerritory на основе ReadLog_DimDate.R

Нам понадобится еще две копии скрипта *ReadLog_DimDate.R* для загрузки оставшихся таблиц. На этом шаге скопируйте код и поменяйте название таблицы в соответствующей переменной с «DimDate» на «DimSalesTerritory». Затем снова отправьте скрипт в Power BI, попутно поменяв имя запроса с *df_read* на *DimSalesTerritory*. После произведенных изменений код должен выглядеть так:

```

library(tidyverse)
library(RODBC)
library(lubridate)

table_name <- "DimSalesTerritory"
sql <- paste0(
  "SELECT * FROM dbo.",
  table_name)

conn <- odbcConnect(dsn = "SQLServer2017")
df_read <- sqlQuery(channel = conn, query = sql)

timestamp <- now()
num_records <- nrow(df_read)

list_insert <- list(
  "DATESTAMP" = timestamp,
  "TABLENAME" = table_name,
  "NUM_RECORDS" = num_records)

df_insert <- as_data_frame(list_insert)

sqlSave(
  channel = conn,
  dat = df_insert,
  tablename = "LoadHistoryLog",
  append = TRUE,
  rownames=FALSE)

odbcClose(conn)

```

Шаг 17. Создайте скрипт для загрузки таблицы FactInternetSales на основе ReadLog_DimDate.R

В последний раз скопируйте скрипт *ReadLog_DimDate.R*, поменяв значение переменной *table_name* с «DimDate» на «FactInternetSales». Скопируйте созданный код в Power BI. На этот раз запрос будет называться *FactInternetSales*:

```

library(tidyverse)
library(RODBC)
library(lubridate)

table_name <- "FactInternetSales"
sql <- paste0(
  "SELECT * FROM dbo.",
  table_name)

conn <- odbcConnect(dsn = "SQLServer2017")
df_read <- sqlQuery(channel = conn, query = sql)

timestamp <- now()
num_records <- nrow(df_read)

```





```
list_insert <- list(
  "DATESTAMP" = datestamp,
  "TABLENAME" = table_name,
  "NUM_RECORDS" = num_records)

df_insert <- as_data_frame(list_insert)

sqlSave(
  channel = conn,
  dat = df_insert,
  tablename = "LoadHistoryLog",
  append = TRUE,
  rownames=FALSE)

odbcClose(conn)
```

В результате выполненных действий не только информация из выбранных таблиц будет загружена в модель данных Power BI, но также в базе данных сохранятся сведения о количестве загруженных записей.

ЧТЕНИЕ ДАННЫХ ИЗ SQL SERVER В POWER BI ПРИ ПОМОЩИ PYTHON

В данном разделе мы произведем те же операции, но с использованием языка Python. Логика останется прежней, но для выполнения задачи вам потребуется чуть меньше шагов. Итак, начнем.

Шаг 1. Создайте DSN для SQL Server

Если вы выполнили пример загрузки данных из предыдущего раздела с использованием языка R, значит, у вас уже есть DSN, и вы сразу можете переходить к следующему шагу. Если нет, обратитесь к этому шагу и пройдите данную процедуру. Механизм создания DSN не зависит от R и Python, так что здесь будет все то же самое.

Шаг 2. Создайте таблицу для ведения логов в SQL Server



Таблица для ведения логов с именем *dbo.LoadHistoryLog* должна быть предварительно создана в базе данных. Выполните приведенную ниже инструкцию SQL в *SQL Server Management Studio (SSMS)*, если вам необходимо создать таблицу *dbo.LoadHistoryLog*:

```
USE AdventureWorksDW_StarSchema
```

```
CREATE TABLE dbo.[LoadHistoryLog] (
```

```

DATESTAMP DATETIME,
TABLENAME VARCHAR(25),
NUM_RECORDS INT
)

```

Чтобы запустить этот скрипт в *SQL Server Management Studio*, нажмите на кнопку **Создать запрос** (New Query) на панели инструментов, после чего будет открыто окно редактора. Вставьте скопированный запрос и нажмите на кнопку **Выполнить** (Execute), чтобы запустить его.

Шаг 3. Создайте скрипт для загрузки таблицы DimDate

В данном скрипте мы будем использовать три библиотеки: *pandas*, *sqlalchemy* и *datetime*. Пакет *pandas* будет нашей рабочей лошадкой, поскольку именно с помощью него мы будем считывать данные для загрузки в модель Power BI и в виде логов в SQL Server. Функция *create_engine()* из библиотеки *sqlalchemy* пригодится нам для создания подключения к базе данных SQL Server, а модуль *datetime* из одноименного пакета используем для выполнения операций с датами. Следующий код поможет нам загрузить все описанные модули и библиотеки:

```

import pandas as pd
from sqlalchemy import create_engine
from datetime import datetime

```

Шаг 4. Определите переменную для хранения имени таблицы, предназначенной для загрузки в Power BI

Имя таблицы, которую мы будем загружать в модель данных Power BI, будет использоваться в нескольких фрагментах скрипта. С целью облегчения поддержки скрипта создадим специальную переменную, которую сможем использовать повторно:

```
tablename = "DimDate"
```

Шаг 5. Создайте подключение к базе данных с помощью библиотеки sqlalchemy

Чтобы создать подключение к базе данных SQL Server, достаточно ввести следующую инструкцию:

```
con = create_engine("mssql+pyodbc://SQLServer2017")
```

Как и в случае с R, использование DSN значительно облегчает задачу подключения к базе данных. В данном примере мы воспользовались функцией *create_engine()* из библиотеки *sqlalchemy*. Для соединения с SQL Server с помощью пакета *sqlalchemy* достаточно ввести строку подключения в формате «mssql+pyodbc://<DSN>», а в нашем случае она приобрела следующий вид: «mssql+pyodbc://SQLServer2017».

Шаг 6. Прочитайте содержимое таблицы DimDate и сохраните его в переменной df_read

На этом этапе мы используем метод *read_sql_table()* из библиотеки *pandas* для чтения содержимого таблицы *DimDate* и сохранения в датафрейме с именем *df_read*. Метод *read_sql_table()* принимает на вход два аргумента: имя таблицы, из которой вы желаете осуществлять чтение, и объект с подключением к базе данных. Код вызова этого метода будет следующим:

```
df_read = pd.read_sql_table(tablename, conn)
```

Шаг 7. Получите текущую дату и время и сохраните в переменной datestamp

Для этого вам достаточно выполнить следующую инструкцию:

```
datestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
```

Мы воспользовались методом *now()* из модуля *datetime* для получения актуальной даты и времени загрузки. К сожалению, этот метод возвращает результат в виде, не совместимом с SQL Server, так что нам необходимо будет преобразовать его. Формат, в котором возвращает результат метод *now()*, – это *datetime.datetime(YYYY, M, D, H, M, SS, MS)*. Таким образом, дата 1 января 2019 года и время 10 вечера будут возвращены как *datetime.datetime(2019, 1, 1, 22, 0, 0, 0)*. Мы изменим формат значения при помощи метода *strftime()* для приведения его к виду *YYYY-MM-DD HH:MM:SS* путем применения следующей форматирующей строки «%Y-%m-%d %H:%M:%S». Это позволит нам привести *datetime.datetime(2019, 1, 1, 22, 0, 0, 0)* к виду «2019-04-14 22:05:24». А уже этот формат вполне приемлем для SQL Server.

Шаг 8. Посчитайте количество записей в таблице DimDate

В наши планы входит фиксирование в логах количества строк, загруженных в модель данных из таблицы *DimDate*. Для получения этой информации можно воспользоваться свойством *shape* из библиотеки *pandas*, как показано ниже:

```
num_records = df.shape[0]
```

В результате мы получим *кортеж* (tuple), состоящий из двух элементов. В первом из них будет содержаться количество строк в датафрейме, а во втором – количество столбцов. Нам необходим только первый элемент, в связи с чем мы ограничили выборку при помощи синтаксиса [0].

Примечание. *Кортеж* (tuple) представляет собой особую структуру данных в Python для хранения последовательности неизменяемых элементов. Кортежи похожи на списки, за тем лишь исключением, что их элементы не могут быть изменены после создания.

Шаг 9. Добавьте запись в датафрейм с информацией для сохранения логов

Это можно сделать в два приема, как показано ниже:

```
dict_insert = {
    "DATESTAMP": [datestamp],
    "TABLENAME": [tablename],
    "NUM_RECORDS": [num_records]}

df_insert = pd.DataFrame.from_dict(dict_insert)
```

Сначала мы создали *словарь* (dictionary) с ключами и соответствующими им значениями. В качестве ключей мы использовали будущие имена столбцов, а в качестве значений – списки, элементы которых будут добавлены в столбцы. В данном случае каждый список состоит ровно из одного элемента.

Во второй строке мы воспользовались методом *from_dict()* класса *DataFrame* из библиотеки *pandas* для создания датафрейма. В качестве аргумента мы передали словарь *dict_insert*, а результат присвоили переменной *df_insert*.

Шаг 10. Добавьте информацию, добытую на предыдущем шаге, в таблицу логов

Чтобы перенести содержимое датафрейма *df_insert* в таблицу базы данных *LoadHistoryLog*, мы вызвали у объекта *df_insert* метод *to_sql()*, как показано ниже:

```
df_insert.to_sql(
    name='LoadHistoryLog',
    con=conn,
    index=False,
    if_exists = 'append'
)
```

В качестве аргумента *name* мы передали имя нашей таблицы – «LoadHistoryLog», аргумент *con* отвечает за подключение к базе данных, и мы ожидаемо передали ему созданный ранее объект *conn*. Параметр *index* установлен

в False, чтобы предостеречь *pandas* от попытки включения значения индекса при вставке записи, а в качестве аргумента *if_exists* мы передали значение «append» для добавления данных в таблицу при ее существовании.

Шаг 11. Скопируйте скрипт в Power BI

Ниже приведен полный скрипт для переноса в Power BI:

```
import pandas as pd
from sqlalchemy import create_engine
from datetime import datetime

tablename = "DimDate"

conn = create_engine("mssql+pyodbc://SQLServer2017")

df_read = pd.read_sql_table(tablename, conn)

timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
num_records = df_read.shape[0]

dict_insert = {
    "DATESTAMP": [timestamp],
    "TABLENAME": [tablename],
    "NUM_RECORDS": [num_records]}

df_insert = pd.DataFrame.from_dict(dict_insert)

df_insert.to_sql(
    name='LoadHistoryLog',
    con=conn,
    index=False,
    if_exists = 'append'
)
```



Как и в предыдущем примере, загрузите скрипт Python в Power BI посредством инструмента **Получить данные** (GetData). В нашем коде создаются два датафрейма – *df_read* и *df_insert*, – и вам нужно сообщить Power BI, какой из них вы хотите загрузить в модель данных. Нам нужно загрузить датафрейм *df_read*, так что выберите его и нажмите на кнопку **Изменить** (Edit). Теперь переименуйте запрос из *df_read* в *DimDate* и нажмите на кнопку **Заккрыть и применить** (Close & Apply), чтобы загрузить информацию в модель данных Power BI.

Шаг 12. Создайте скрипт для загрузки таблицы DimProduct на основе ReadLog_DimDate.py

Как и в случае с R, нам придется четыре раза воспроизвести свои действия для оставшихся четырех таблиц. Это можно сделать, четырежды скопировав скрипт из файла *ReadLog_DimDate.py* с внесением небольшого изменения,

касающегося значения переменной *tablename*. На этом шаге мы присвоим данной переменной значение «DimProduct». Скопируйте скрипт *ReadLog_DimDate.py* и поменяйте значение переменной *tablename* с «DimDate» на «DimProduct». Теперь повторите шаг 11 с тем лишь исключением, что запрос должен называться *DimProduct*. Вот как должен выглядеть скрипт после внесения изменений:

```
import pandas as pd
from sqlalchemy import create_engine
from datetime import datetime

tablename = "DimProduct"

conn = create_engine("mssql+pyodbc://SQLServer2017")

df_read = pd.read_sql_table(tablename, conn)

timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
num_records = df_read.shape[0]

dict_insert = {
    "DATESTAMP": [timestamp],
    "TABLENAME": [tablename],
    "NUM_RECORDS": [num_records]}

df_insert = pd.DataFrame.from_dict(dict_insert)

df_insert.to_sql(
    name='LoadHistoryLog',
    con=conn,
    index=False,
    if_exists = 'append'
)
```



Шаг 13. Создайте скрипт для загрузки таблицы DimPromotion на основе ReadLog_DimDate.py

Скопируйте скрипт *ReadLog_DimDate.py* и поменяйте значение переменной *tablename* с «DimDate» на «DimPromotion». После этого снова повторите шаг 11, но назовите запрос *DimPromotion*. Вот как будет выглядеть скрипт для очередной таблицы:

```
import pandas as pd
from sqlalchemy import create_engine
from datetime import datetime

tablename = "DimPromotion"

conn = create_engine("mssql+pyodbc://SQLServer2017")
```



```

df_read = pd.read_sql_table(tablename, conn)

datestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
num_records = df_read.shape[0]

dict_insert = {
    "DATESTAMP": [datestamp],
    "TABLENAME": [tablename],
    "NUM_RECORDS": [num_records]}

df_insert = pd.DataFrame.from_dict(dict_insert)

df_insert.to_sql(
    name='LoadHistoryLog',
    con=conn,
    index=False,
    if_exists = 'append'
)

```

Шаг 14. Создайте скрипт для загрузки таблицы DimSalesTerritory на основе ReadLog_DimDate.py

Сделайте еще одну копию скрипта *ReadLog_DimDate.py*, на этот раз присвоив переменной *tablename* значение «DimSalesTerritory». Так же назовите и запрос при выполнении шага 11. Вот как должен выглядеть измененный код:

```

import pandas as pd
from sqlalchemy import create_engine
from datetime import datetime

tablename = "DimSalesTerritory"

conn = create_engine("mssql+pyodbc://SQLServer2017")

df_read = pd.read_sql_table(tablename, conn)

datestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
num_records = df_read.shape[0]

dict_insert = {
    "DATESTAMP": [datestamp],
    "TABLENAME": [tablename],
    "NUM_RECORDS": [num_records]}

df_insert = pd.DataFrame.from_dict(dict_insert)

df_insert.to_sql(
    name='LoadHistoryLog',
    con=conn,
    index=False,
    if_exists = 'append'
)

```



Шаг 15. Создайте скрипт для загрузки таблицы FactInternetSales на основе ReadLog_DimDate.py

И в последний раз скопируйте скрипт *ReadLog_DimDate.py*, поменяв значение переменной *tablename* с «DimDate» на «FactInternetSales». При запуске скрипта переименуйте запрос *df_read* в *FactInternetSales*. Так должен выглядеть скрипт:



```
import pandas as pd
from sqlalchemy import create_engine
from datetime import datetime

tablename = "FactInternetSales"

conn = create_engine("mssql+pyodbc://SQLServer2017")

df_read = pd.read_sql_table(tablename, conn)

timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
num_records = df_read.shape[0]

dict_insert = {
    "DATESTAMP": [timestamp],
    "TABLENAME": [tablename],
    "NUM_RECORDS": [num_records]}

df_insert = pd.DataFrame.from_dict(dict_insert)

df_insert.to_sql(
    name='LoadHistoryLog',
    con=conn,
    index=False,
    if_exists = 'append'
)
```



Итак, вы довольно легко и просто справились при помощи языков R и Python с задачей, для решения которой средствами Power Query потребовалось бы куда больше времени. Да, Power Query позволяет читать данные из таблиц SQL Server, но с сохранением логов в базе данных ему будет справиться проблематично. Напротив, в языках R и Python есть богатый набор пакетов и библиотек, позволяющих решить столь сложный сценарий весьма быстро.

Глава 6

.....

Чтение в модель данных Power BI посредством API



Многие государственные организации и частные компании позволяют обращаться к своей информации посредством *API данных* (Data API). *API* (Application Programming Interface – интерфейс программирования приложений) являет собой интерфейс программного доступа к информации, предоставляемой *поставщиком или провайдером данных* (data provider). Наборы данных, предлагаемые поставщиками, способны значительно обогатить ваши модели данных Power BI.

Осуществить доступ к API данных посредством языка M в Power Query не так просто. И совсем другое дело – R и Python. В данной главе вы научитесь извлекать информацию из API данных Бюро переписи населения США (US Census) и загружать ее в модель данных Power BI с помощью R и Python. Начнем, как и всегда, с языка R, после чего сделаем те же операции на Python.

ЧТЕНИЕ И ЗАГРУЗКА ДАННЫХ В POWER BI из API с помощью R

Вот вам сценарий. Вы маркетолог-аналитик розничной сети магазинов Fleek, специализирующихся на продаже одежды для молодежи. У компании хорошо налажена торговля на юго-востоке США, но в штате Индиана (Indiana) нет ни одного магазина. Вам поручено провести исследование демографической ситуации в округах штата на предмет открытия точек продаж. Вы решили воспользоваться информацией от Бюро переписи населения США (US Census) и с ее помощью провести анализ данных в Power BI. В этом разделе мы покажем процедуру извлечения демографических данных через API посредством языка R.

Шаг 1. Получите персональный ключ API к Census

Для доступа к базе данных US Census вам понадобится персональный *ключ API к Census* (Census API key). Получить его можно по следующей ссылке: https://api.census.gov/data/key_signup.html. Для этого вам необходимо будет

заполнить небольшую форму регистрации и подписать пользовательское соглашение. После этого нажмите на кнопку **Submit Key Request**, и ваш ключ доступа к API будет выслан вам на почту. Убедитесь, что сохранили его в надежном месте.



Шаг 2. Загрузите необходимые пакеты R

В данном примере мы будем использовать пакеты *tidyverse* и *tidycensus*. Вы часто встречались в этой книге с первым пакетом, так что вам не потребуются дополнительные объяснения его важности. Что касается пакета *tidycensus*, то он позволяет вам считывать необходимую информацию из базы данных US Census и программы демографических исследований США (American Community Survey). При этом данные будут возвращены в виде датафреймов, пригодных для использования с пакетом *tidyverse*. По следующей ссылке вы можете ознакомиться с информацией о том, как использовать этот пакет и поставляемые наборы данных: <https://walkerke.github.io/tidycensus>. Используйте код, приведенный ниже, для загрузки необходимых пакетов и вашего персонального кода доступа к API, полученного на предыдущем шаге, на время текущей сессии:

```
library(tidycensus)
library(tidyverse)

census_api_key("<персональный код API>", install = FALSE)
```

Используя этот подход, вам придется включать свой код при каждом запуске. Если же вы хотите установить свой ключ API в окружение R для будущего использования, лучше воспользоваться следующим способом:

```
census_api_key("<персональный код API>", install = TRUE)
readRenvir("~/.Renvir")
```

Передача в качестве аргумента *install* значения TRUE позволяет установить ключ доступа к API в ваше окружение R. В следующей строке кода происходит перезагрузка окружения, чтобы у вас была возможность использовать свой код доступа к API в текущей сессии без необходимости перезапуска R.

Шаг 3. Определите переменные для возврата из вашего набора данных

US Census предоставляет на выбор несколько наборов данных. Информация, которая понадобится нам в данном примере, содержится в наборе данных *acs5*.

Примечание. Набор данных *acs5* основывается на исследованиях за период в пять лет. Также доступны следующие наборы данных: *sf1*, *sf3*, *acs1*, *acs3*, *acs5*, *acs1/profile*, *acs3/profile*, *acs5/profile*, *acs1/subject*, *acs3/subject* и *acs5/subject*. Больше информации о наборах данных, предоставляемых US Census, вы можете получить по адресу www.census.gov/programs-surveys/acs/guidance/handbooks.html.

Нам понадобится для анализа информация о населении в разрезе возрастных групп из набора данных *acs5* за 2015 год, но мы не знаем имен переменных, содержащих нужные нам сведения. Используйте следующий код для получения полного списка переменных, доступных в наборе данных *acs5* за 2015 год:

```
v15 <- load_variables(year = 2015, dataset= "acs5", cache = FALSE)
View(v15)
```

Давайте остановимся на минутку и внимательнее рассмотрим приведенный выше код. Функция *load_variables()* возвращает датафрейм с переменными в зависимости от переданных в нее аргументов. Параметр *cache* дает возможность кешировать данные локально для их быстрого извлечения. Вывод функции показан в табл. 6.1.

Таблица 6.1

Name	Label	concept
1	B06001_001e	Estimate!!Total
2	B06001_002e	Estimate!!Total!!Under 5 years
3	B06001_003e	Estimate!!Total!!5 to 17 years
4	B06001_004e	Estimate!!Total!!18 to 24 years
5	B06001_005e	Estimate!!Total!!25 to 34 years

В приведенном выводе показаны всего пять записей из набора данных. Но результирующий набор будет содержать более 22 тысяч записей! Использование функции *View()* позволяет вывести список переменных в виде таблицы с возможностью осуществления фильтрации. Однако, даже вооружившись фильтрами, бывает довольно сложно найти нужные вам переменные. В следующих главах мы обсудим более продвинутые техники фильтрации с помощью пакета *dplyr*, которые помогут вам лучше ориентироваться в этом ворохе переменных. Я применил некоторые из этих техник для поиска переменных, которые мы будем использовать на следующем шаге.

Шаг 4. Создайте символьный вектор, содержащий нужные вам переменные

После определения списка нужных вам переменных необходимо поместить их в символьный вектор. Ниже показан вектор переменных с демографической информацией о нужных нам возрастных группах:

```
cns_vars <- c("B06001_001E", "B06001_002E", "B06001_003E",
"B06001_004E", "B06001_005E", "B06001_006E", "B06001_007E",
"B06001_008E", "B06001_009E", "B06001_010E", "B06001_011E",
"B06001_012E")
```

В переменной *B06001_001E* хранится общая численность населения, а следующие переменные содержат информацию о разных возрастных груп-

пах. В частности, переменная *B06001_002E* отражает численность населения в возрасте до 5 лет, *B06001_003E* – от 5 до 17, *B06001_004E* от 18 до 24, *B06001_005E* – от 25 до 34, *B06001_006E* – от 35 до 44, *B06001_007E* – от 45 до 54, *B06001_008E* от 55 до 59, *B06001_009E* – от 60 до 61, *B06001_010E* – от 62 до 64, *B06001_011E* – от 65 до 74 и *B06001_012E* от 75 и старше.

Шаг 5. Сконфигурируйте функцию `get_acs`

На этом шаге вам необходимо будет настроить вызов функции `get_acs()` для извлечения информации, а результат присвоить переменной *IN_POP_BY_COUNTY_BY_AGE*. Функция `get_acs()` получает данные из API US Census и сохраняет их в датафрейме. Следующий фрагмент кода поможет вам извлечь нужную информацию по возрастным группам на уровне округов для штата Индиана:

```
IN_POP_BY_COUNTY_BY_AGE <-
  suppressMessages(
    get_acs(
      geography = "county",
      state = "IN",
      variables = cns_vars,
      survey = "acs5",
      year = 2015,
      output = "wide"
    )
  )
```



Давайте внимательнее присмотримся к приведенному коду. Аргумент *geography* говорит пакету *tidycensus* о том, на каком уровне географической детализации вы хотите получить данные. В нашем случае мы хотели бы собрать информацию в разрезе округов, поэтому присвоили этому аргументу значение «county». Для ограничения выборки конкретным штатом мы передали в качестве аргумента *state* строку «IN». Аргумент *variables* содержит вектор переменных, который мы создали на предыдущем шаге. Параметр *survey* отвечает за набор данных, в котором необходимо осуществлять поиск. Аргументом *year* мы ограничили поиск одним нужным нам годом, а в аргументе *output* обозначили желаемый формат вывода. Значение «wide» форматирует результат таким образом, чтобы каждая переменная располагалась в своем столбце. Обратите внимание, что сам вызов функции `get_acs()` обернут в другую функцию `suppressMessages()`, подавляющую вывод на консоль сообщений от вложенной функции, которые могут привести к проблемам в Power BI.

Шаг 6. Присвойте переменным (столбцам) осмысленные имена

По умолчанию US Census возвращает данные в виде датафрейма с весьма загадочными именами столбцов, которые мало что могут вам сказать. К сча-

стью, вы могли узнать, что находится в каждой переменной при вызове функции `load_variables()` на предыдущих шагах. Сейчас пришло время дать колонкам осмысленные имена, и это мы сделаем при помощи функции `rename()` из пакета `dplyr`, как показано ниже:

```
IN_POP_BY_COUNTY_BY_AGE =
  rename(
    IN_POP_BY_COUNTY_BY_AGE,
    `Total` = B06001_001E,
    `Under 5` = B06001_002E,
    `5 to 17` = B06001_003E,
    `18 to 24` = B06001_004E,
    `25 to 34` = B06001_005E,
    `35 to 44` = B06001_006E,
    `45 to 54` = B06001_007E,
    `55 to 59` = B06001_008E,
    `60 and 61` = B06001_009E,
    `62 to 64` = B06001_010E,
    `65 to 74` = B06001_011E,
    `75+` = B06001_012E
  )
```

Первым аргументом функция `rename()` принимает ссылку на набор данных, содержащий столбцы, которые вам необходимо переименовать. Далее следуют пары значений с новыми и старыми именами столбцов, разделенными знаком равенства.

Одним из преимуществ использования функции `rename()` из пакета `dplyr` является то, что вы можете использовать имена столбцов, неприемлемые в обычных датафреймах в R.

Шаг 7. Скопируйте скрипт в Power BI

Ниже приведен полный скрипт, созданный на предыдущих шагах:

```
library(tidycensus)
library(tidyverse)

census_api_key(
  "<census api key>", overwrite = FALSE,
  install = FALSE)

cns_vars <- c("B06001_001E", "B06001_002E", "B06001_003E", "B06001_004E", "B06001_005E", "B06001_006E", "B06001_007E", "B06001_008E", "B06001_009E", "B06001_010E", "B06001_011E", "B06001_012E")

IN_POP_BY_COUNTY_BY_AGE <-
  suppressMessages(
    get_acs(
      geography = "county",
      state = "IN",
      variables = cns_vars,
      survey = "acs5",
```

```

        year = 2015,
        output = "wide"
    )
)

IN_POP_BY_COUNTY_BY_AGE =
    rename(
        IN_POP_BY_COUNTY_BY_AGE,
        `Total`=B06001_001E,
        `Under 5`=B06001_002E,
        `5 to 17`=B06001_003E,
        `18 to 24`=B06001_004E,
        `25 to 34`=B06001_005E,
        `35 to 44`=B06001_006E,
        `45 to 54`=B06001_007E,
        `55 to 59`=B06001_008E,
        `60 and 61`=B06001_009E,
        `62 to 64`=B06001_010E,
        `65 to 74`=B06001_011E,
        `75+`=B06001_012E
    )

```



Скопируйте код в редактор скриптов R в Power BI. Имя датафрейма по умолчанию будет `IN_POP_BY_COUNTY_BY_AGE`. Если вы хотите переименовать запрос, перейдите в раздел изменения данных и сделайте это. Если же вас устраивает имя по умолчанию, вы можете загрузить набор данных в Power BI и использовать его в качестве источника данных в своих визуализациях.

ЧТЕНИЕ И ЗАГРУЗКА ДАННЫХ В POWER BI ИЗ API С ПОМОЩЬЮ PYTHON

Также для загрузки данных в Power BI вы можете использовать язык Python. И в этом разделе мы посмотрим, как извлекать в Power BI данные о численности населения при помощи этого языка. Сам процесс во многом будет напоминать предыдущий сценарий с использованием R.

Шаг 1. Получите персональный ключ API к Census



Для доступа к базе данных US Census вам понадобится персональный *ключ API к Census* (Census API key). Получить его можно по ссылке https://api.census.gov/data/key_signup.html. Для этого вам нужно заполнить небольшую форму регистрации и подписать пользовательское соглашение. После этого нажмите на кнопку **Submit Key Request**, и ключ доступа к API будет выслан вам на почту. Убедитесь, что сохранили его в надежном месте.

Для работы с данными из этого API в Python вам потребуется импортировать библиотеку *censusdata*.

Шаг 2. Загрузите необходимые библиотеки Python

В данном примере мы будем использовать пакеты *pandas* и *censusdata*. *Pandas* мы будем применять для манипуляций с данными, а *censusdata* – для взаимодействия с API Census. Используйте код, приведенный ниже, для загрузки необходимых пакетов:

```
import pandas as pd
import censusdata
```

Обратитесь к началу книги, чтобы узнать, как загружать и устанавливать нужные вам библиотеки в Python.

Шаг 3. Определите переменные для возврата из вашего набора данных

Нужная нам информация о численности населения за 2015 год по возрастным категориям содержится в наборе данных *acs5*.

Введите следующий код в консоли, чтобы выполнить поиск:

```
v15 = censusdata.search(
    'acs5', 2015, 'concept',
    'PLACE OF BIRTH BY AGE IN THE UNITED STATES')
```

Примечание. Этот фрагмент кода не будет являться частью основного скрипта. Мы запустили его, чтобы найти нужные нам переменные на основании введенных критериев. В репозитории на GitHub к данной главе содержится скрипт, который не только извлекает информацию о переменных, но и сохраняет ее в виде файла CSV на случай, если вам удобнее анализировать полученные данные в таком формате.

Как видите, в предыдущем коде мы воспользовались функцией *search()* из библиотеки *censusdata*. Первым аргументом мы передаем наименование набора данных, содержащего нужные нам переменные, далее указываем год для ограничения выборки, а третьим параметром передаем поле для поиска. Здесь вы можете указать «label» или «concept». Мы использовали второй вариант, поскольку это поле содержит описание переменных, которые нам нужны. Последним аргументом идет критерий поиска.

Функция *censusdata.search()* возвращает список кортежей.

Примечание. Кортеж представляет собой структуру данных в Python для хранения последовательности неизменяемых элементов. Кортежи похожи на списки за тем лишь исключением, что в списках элементы могут быть изменены после создания, а в кортежах – нет.

Каждый кортеж содержит три элемента. Первый – это *имя переменной* (name), второй – *описание* (concept), а третий – *метка* (label). В нашем случае мы получим на выходе список из 120 кортежей. Вы можете выбрать первые

пять элементов, чтобы лучше понять, с какой информацией имеете дело. Для этого выполните следующий код:



```
v15[0:5]
```

В результате мы получим пять элементов из списка v15 – с нулевой позиции по пятую, не включая ее. Отформатированная версия полученного подписка приведена ниже. С таким форматированием можно легко видеть все входящие в список кортежи:

```
[
    ('B06001_001E',
     'B06001. PLACE OF BIRTH BY AGE IN THE UNITED STATES',
     'Total:'),
    ('B06001_001M',
     'B06001. PLACE OF BIRTH BY AGE IN THE UNITED STATES',
     'Margin Of Error For!!Total:'),
    ('B06001_002E',
     'B06001. PLACE OF BIRTH BY AGE IN THE UNITED STATES',
     'Under 5 years'),
    ('B06001_002M',
     'B06001. PLACE OF BIRTH BY AGE IN THE UNITED STATES',
     'Margin Of Error For!!Under 5 years'),
    ('B06001_003E',
     'B06001. PLACE OF BIRTH BY AGE IN THE UNITED STATES',
     '5 to 17 years')
]
```

Шаг 4. Создайте список, содержащий нужные вам переменные



После определения перечня нужных вам переменных необходимо собрать их все в список. Ниже приведен код на Python для создания списка, содержащего переменные, отвечающие за нужные нам возрастные диапазоны:

```
cns_vars = [
    "B06001_001E", "B06001_002E", "B06001_003E", "B06001_004E",
    "B06001_005E", "B06001_006E", "B06001_007E", "B06001_008E",
    "B06001_009E", "B06001_010E", "B06001_011E", "B06001_012E"
]
```

Распределение возрастных диапазонов по переменным будет таким же, как в сценарии с R. В переменной B06001_001E хранится общая численность населения, а последующие переменные содержат информацию о разных возрастных группах. В частности, переменная B06001_002E отражает численность населения в возрасте до 5 лет, B06001_003E – от 5 до 17, B06001_004E от 18 до 24, B06001_005E – от 25 до 34, B06001_006E – от 35 до 44, B06001_007E – от 45 до 54, B06001_008E от 55 до 59, B06001_009E – от 60 до 61, B06001_010E – от 62 до 64, B06001_011E – от 65 до 74 и B06001_012E от 75 и старше.

Шаг 5. Создайте список кортежей с географическими фильтрами для набора данных

Теперь мы определим свои требования к географии запроса при помощи списка кортежей. Каждый кортеж в списке будет содержать два элемента: тип географического объекта и FIPS-код конкретного объекта, который мы хотим получить.

В данном примере нам необходимо получить данные по штату Индиана в разрезе округов. Такие требования можно сформулировать следующим образом:

```
geographies = [('state', '18'),('county', '*')]
```

Первый элемент первого кортежа («state») отражает требование отфильтровать штаты, а второй указывает на конкретное значение – «18», что является FIPS-кодом штата Индиана. Таким образом мы ограничим выборку нужным нам штатом. Если вы не знаете FIPS-код штата Индиана, можете воспользоваться пакетом *us* для получения этой информации следующим образом:

```
import us
us.states.IN.fips
```

Второй кортеж служит для фильтрации округов внутри штата. Первым элементом мы указали строку «county», чтобы дать *censusdata* понять, что нас интересуют округа, а звездочка во втором элементе означает, что мы хотим получить все 92 округа штата Индиана.

Шаг 6. Извлеките данные при помощи функции `censusdata.download()`

При помощи следующего фрагмента кода можно получить нужные данные о населении из US Census и сохранить в датафрейме с именем *IN_POP_BY_COUNTY_BY_AGE*:

```
IN_POP_BY_COUNTY_BY_AGE = censusdata.download(
    'acs5', 2015, censusdata.censusgeo(geographies),
    cns_vars)
```

В первом аргументе определяется источник информации – в нашем случае это набор данных *acs5*. Второй аргумент отвечает за год выборки. В третьем применяются географические критерии, определенные нами ранее. Помните, на пятом шаге мы сформулировали географические требования к выборке, и в этом аргументе передаем их посредством специальной функции *censusdata.censusgeo()*. Последний параметр – это наш список с переменными *cns_vars*, который мы создали на четвертом шаге.

Шаг 7. Переиндексируйте датафрейм, созданный на шестом шаге

Географическая информация содержится в индексе датафрейма, но для того чтобы Power BI мог воспользоваться ей, она должна располагаться в отдельном столбце. При помощи функции `reset_index()` можно выделить индексы в обособленную колонку следующим образом:

```
IN_POP_BY_COUNTY_BY_AGE = IN_POP_BY_COUNTY_BY_AGE.reset_index()
```

В результате выполнения этой строки кода индексы окажутся в столбце с именем `index`, и уже этой информацией легко сможет воспользоваться Power BI.

Шаг 8. Дайте столбцам осмысленные имена

Переменные, которые мы определили на четвертом шаге, будут использованы для именования столбцов в результирующем наборе данных. Но их имена мало о чем нам говорят. Библиотека *pandas* дает вам возможность переименовать столбцы датафрейма по вашему усмотрению. В следующем фрагменте кода мы определим словарь с информацией для переименования колонок:

```
new_names = {
    'index': 'Geography', 'B06001_001E': 'Total',
    'B06001_002E': 'Under 5', 'B06001_003E': '5 to 17',
    'B06001_004E': '18 to 24', 'B06001_005E': '25 to 34',
    'B06001_006E': '35 to 44', 'B06001_007E': '45 to 54',
    'B06001_008E': '55 to 59', 'B06001_009E': '60 and 61',
    'B06001_010E': '62 to 64', 'B06001_011E': '65 to 74',
    'B06001_012E': '75+'
}
```

В представленном выше коде мы создали *словарь* (dictionary), определяющий правила переименования столбцов в нашем датафрейме. В качестве ключей (слева от двоеточия) мы использовали старые имена столбцов, а в качестве значений (справа от двоеточия) – новые.

Шаг 9. Переименуйте столбцы в датафрейме

Переименовать столбцы можно при помощи следующего фрагмента кода:

```
IN_POP_BY_COUNTY_BY_AGE = IN_POP_BY_COUNTY_BY_AGE.rename(columns=new_names)
```

Метод `rename()` применительно к датафрейму `IN_POP_BY_COUNTY_BY_AGE` используется для переименования столбцов в нем. В качестве аргумента `columns` мы передали созданный словарь `new_names` со всей необходимой информацией для переименования.

Шаг 10. Скопируйте скрипт в Power BI

Ниже приведен полный скрипт, созданный на предыдущих шагах:

```
import pandas as pd
import censusdata

cns_vars = [
    "B06001_001E", "B06001_002E", "B06001_003E", "B06001_004E",
    "B06001_005E", "B06001_006E", "B06001_007E", "B06001_008E",
    "B06001_009E", "B06001_010E", "B06001_011E", "B06001_012E"
]

geographies = [('state', '18'), ('county', '*')]

IN_POP_BY_COUNTY_BY_AGE = censusdata.download(
    'acs5', 2015, censusdata.censusgeo(geographies),
    cns_vars)

IN_POP_BY_COUNTY_BY_AGE = IN_POP_BY_COUNTY_BY_AGE.reset_index()

new_names = {
    'index': 'Geography', 'B06001_001E': 'Total',
    'B06001_002E': 'Under 5', 'B06001_003E': '5 to 17',
    'B06001_004E': '18 to 24', 'B06001_005E': '25 to 34',
    'B06001_006E': '35 to 44', 'B06001_007E': '45 to 54',
    'B06001_008E': '55 to 59', 'B06001_009E': '60 and 61',
    'B06001_010E': '62 to 64', 'B06001_011E': '65 to 74',
    'B06001_012E': '75+'
}

IN_POP_BY_COUNTY_BY_AGE = IN_POP_BY_COUNTY_BY_AGE.rename(columns=new_names)
```

Скопируйте получившийся код и вставьте во встроенный редактор Python в Power BI. Имя датафрейма по умолчанию будет `IN_POP_BY_COUNTY_BY_AGE`. Если вы желаете переименовать запрос, перейдите в раздел изменения данных в Power Query и сделайте это. Если же вас устраивает имя по умолчанию, вы можете загрузить набор данных в Power BI и использовать его в качестве источника данных в своих визуализациях.

ЗАКЛЮЧЕНИЕ

В данной главе вы научились импортировать информацию из API данных US Census в модель данных Power BI при помощи R и Python. Загрузка разрозненных данных из внешних источников, таких как US Census, может позволить вам существенно обогатить свою модель данных. Разумеется, вы можете импортировать информацию не только с сайта Бюро переписи населения США, – многие компании настраивают собственный API данных для быстрого доступа к своей информации. А будучи добавленными в модель данных Power BI, эти источники могут быть использованы при построении наглядных и очень полезных визуализаций.

Часть III

.....

ПРЕОБРАЗОВАНИЕ ДАННЫХ ПРИ ПОМОЩИ R И PYTHON



Глава 7

.....

Продвинутые строковые операции и распознавание шаблонов



Базовые операции со строками и распознавание шаблонов можно выполнять при помощи огромного разнообразия строковых функций, входящих в инструмент Power Query. Многие из этих функций доступны прямо из интерфейса Power Query, кроме того, их можно использовать в пользовательских настраиваемых столбцах. Все эти инструменты прекрасно работают в относительно простых ситуациях, но когда сценарий становится более сложным, их зачастую оказывается недостаточно. К счастью, в языках R и Python есть немало продвинутых средств для распознавания шаблонов и выполнения строковых операций, которые придут вам на помощь тогда, когда потенциала богатого, но не всемогущего инструмента Power Query окажется маловато.

В данной главе мы рассмотрим несколько вариантов использования языков R и Python для манипулирования строками и их сопоставления с образцами и шаблонами. При этом мы в основном сосредоточимся на использовании *регулярных выражений* (regular expression). Регулярные выражения, часто сокращенно называемые *regex*, повсеместно применяются для работы с шаблонами поиска. Это мощнейший инструмент, позволяющий осуществлять невероятно гибкий строковый поиск и сопоставление с образцами с применением характерной для вашей области бизнес-логики, чего так не хватает базовому функционалу Power Query.

Итак, в этой главе вы узнаете, как можно:

- скрывать конфиденциальные сведения в данных;
- подсчитывать количество слов и предложений в текстовом поле большой длины;
- исключать из загрузки имена, не отвечающие утвержденному шаблону;
- проверять строки на соответствие шаблону.

Все эти возможности мы рассмотрим на примерах, с которыми вы можете столкнуться при решении реальных задач. И начнем с маскировки секретных данных.

ЗАЩИТА КОНФИДЕНЦИАЛЬНЫХ СВЕДЕНИЙ

Многие компании собирают конфиденциальные сведения о своих клиентах, что ставит под угрозу безопасность как самих компаний, так и их контрагентов. В связи с этим в большинстве организаций существуют строгие правила хранения и передачи информации, позволяющие минимизировать риски. Однако иногда компаниям все же приходится обмениваться данными со сторонними организациями с целью повышения эффективности деятельности фирмы. И, делая это, они стараются скрывать конфиденциальные сведения, касающиеся своих клиентов.

В данном сценарии вице-президенты двух компаний обсуждают возможность создания единой системы отчетности в Power BI. Одним из требований к будущей системе является выполнение *анализа тональности текста* (sentiment analysis) в комментариях, создаваемых после каждого разговора с должниками. Некоторые комментарии могут содержать номера социального страхования (SSN) и телефоны клиентов. Для выполнения анализа планируется использовать службу *Microsoft Cognitive Services*, но перед этим необходимо скрыть от посторонних глаз конфиденциальную информацию в виде номеров социального страхования и телефонов контрагентов.

Примечание. *Microsoft Cognitive Services* представляет собой службу, в состав которой входит набор алгоритмов машинного обучения, разработанных Microsoft для решения задач, связанных с применением искусственного интеллекта. Далее в этой книге мы будем подробно говорить о выполнении анализа тональности текста в Power BI.

Итак, наши вице-президенты обратились за советом к знакомому аналитику, который разработал для них два решения: одно – с применением языка R, второе – при помощи Python. Сначала мы посмотрим на сценарий с R.

Защита конфиденциальных сведений в Power BI с помощью R

На этом примере мы проиллюстрируем процесс использования регулярных выражений для скрытия данных в Power BI при помощи языка R. Реализация подобного сценария посредством стандартных средств Power Query будет весьма затруднительной задачей. Итак, давайте по традиции пройдем по шагам!

Шаг 1. Импортуйте пакеты tidyverse и stringr

Как и всегда, начнем с загрузки необходимых пакетов. Первый пакет, который нам понадобится, – это *tidyverse*. В частности, мы будем использовать

библиотеки *dplyr* и *readr*, входящие в его состав, для добавления поля в набор данных и чтения информации в датафрейм соответственно. Также большая часть работы в нашем сценарии выпадет на пакет *stringr*, из которого мы воспользуемся функцией *str_replace()* совместно с регулярными выражениями для удаления конфиденциальных данных. Ниже приведен код для импорта пакетов:

```
library(tidyverse)
library(stringr)
```



Шаг 2. Напишите функцию для очистки данных

Одним из главных преимуществ использования языка R при работе в Power BI является легкость, с которой в нем можно создавать собственные функции и многократно ими пользоваться. Функция, которую мы напомним в этом разделе, будет принимать на вход строку, выполнять проверку на присутствие в ней номеров социального страхования и телефонов клиентов с применением регулярных выражений, скрывать найденные совпадения и возвращать итоговый результат. Выполните следующие действия для написания функции:

1. Создайте оболочку функции, как показано ниже:

```
mask_text <- function(unmask_text){
}
```

Имя функции – *mask_text*, а на вход она будет принимать один аргумент с именем *unmask_text*.

2. Создайте регулярное выражение для американского формата телефонного номера и присвойте его переменной с именем *phone_pattern*, как показано ниже:

```
phone_pattern = "([2-9][0-9]{2})[- .]([0-9]{3})[- .]([0-9]{4})"
```

Это стандартное регулярное выражение для идентификации номеров телефонов. Такие шаблоны можно без труда найти в интернете.

3. Создайте регулярное выражение для номеров социального страхования и присвойте его переменной *ssn_pattern* следующим образом:

```
ssn_pattern = "\\d{3}-\\d{2}-\\d{4}"
```

Как и в случае с телефонными номерами, шаблон регулярных выражений для поиска SSN легко можно написать самому или отыскать на просторах интернета. Здесь я показал простейший вид выражения для номеров социального страхования.

Цифровые символы в регулярных выражениях обозначаются последовательностью *\d*, а поскольку обратная косая черта является специальным символом, необходимо применить экранирование посредством двойного написания.

Примечание. Символы вроде обратной косой черты и точки в регулярных выражениях имеют особое значение. Иначе они именуются *метасимволами* (metacharacter). Если вам необходимо указать их буквальное присутствие в регулярном выражении, экранируйте их дополнительной обратной косой чертой. К примеру, если вам нужно выполнить поиск точки в строке, предварите ее в регулярном выражении обратной косой чертой. Сам обратный слеш также нуждается в дополнительном экранировании. Что касается сочетания обратной косой черты и точки, такую последовательность можно обнаружить, указав в строке регулярного выражения два слеша и точку: «\\.».

Теперь, когда вы знаете, как в регулярных выражениях обозначаются цифровые символы, давайте посмотрим, как строится простейший шаблон для определения номера SSN. Базовый формат для номеров социального страхования определяется как сочетание из трех цифр, двух и четырех, разделенных дефисом. В переменной `ssn_pattern` для учета такого формата мы использовали сочетание последовательностей `\\d` и `{}`. Последовательность `\\d` говорит о том, что мы ждем цифровые символы, а `{}` указывает на их количество. Дефисы помещены в те места, где они встречаются в номере SSN.

4. Телефонные номера мы будем подменять с использованием следующего кода:

```
cleanned_text <- str_replace(
  unmask_text, pattern = phone_pattern,
  replacement = "XXX-XXX-XXXX")
```

Здесь мы воспользовались функцией `str_replace()` для замены всех вхождений телефонных номеров в исходном тексте. Что отличает эту функцию от функции `Text.Replace` из арсенала Power Query, так это то, что вы можете использовать в ней регулярные выражения. Так что в Power Query вы можете осуществить лишь простейший поиск по шаблону, а когда речь идет о сложных паттернах, на помощь придут регулярные выражения. Функция `str_replace()` принимает исходную строку в качестве первого аргумента и проверяет ее на совпадения с шаблоном, переданным следом. Все вхождения шаблонов заменяются в результате на значение, переданное в аргументе `replacement`. Таким образом, если в функцию передать строку «Ryan's phone number is 555-852-6301 and his SSN is 123-45-6789», на выходе мы получим строку «Ryan's phone number is XXX-XXX-XXXX and his SSN is 123-45-6789».

5. Маскировка номеров социального страхования в тексте выполняется по такому же принципу с помощью следующего кода:

```
cleanned_text <- str_replace(
  cleaned_text, pattern = ssn_pattern,
  replacement = "XXX-XX-XXXX")
```

С номерами SSN мы проворачиваем такой же трюк, как с номерами телефонов, и в итоге исходная строка «Ryan's phone number is XXX-XXX-XXXX and his SSN is 123-45-6789», переданная на вход функции,

превратится в строку «Ryan's phone number is XXX-XXX-XXXX and his SSN is XXX-XX-XXXX».

6. В заключительной строке функции мы возвращаем исправленную строку при помощи ключевого слова *return*:

```
mask_text <- function(unmask_text){
  phone_pattern = "([2-9][0-9]{2})[- .]([0-9]{3})[- .]([0-9]{4})"
  ssn_pattern = "\\d{3}-\\d{2}-\\d{4}"

  cleaned_text <- str_replace(
    unmask_text, pattern = phone_pattern,
    replacement = "XXX-XXX-XXXX")

  cleaned_text <- str_replace(
    cleaned_text, pattern = ssn_pattern,
    replacement = "XXX-XX-XXXX")

  return(cleaned_text)
}
```

Как видите, мы заключили логику замены номеров социального страхования и телефонных номеров в функцию *mask_text()*, и когда нам понадобится, сможем вызвать ее. Выделение логики в функции позволяет сделать код более простым для восприятия и облегчает его повторное использование без необходимости переписывать код.

Шаг 3. Считайте комментарии в датафрейм

Прочитайте содержимое файла *Comments.csv* и поместите его в переменную с именем *df*, как показано ниже:

```
df <- read_csv("Comments.csv")
```

Если вы хотите посмотреть, как выглядят данные, которые вы считали, можете ввести следующий код в консоли для отображения информации в виде датафрейма:

```
View(df)
```

Шаг 4. Скройте телефонные номера и номера социального страхования в поле комментария

На данном шаге мы применим функцию *mask_text()* к полю *Comment*. Сделаем это внутри функции *mutate()* из пакета *dplyr*, которая позволит провести модификацию данных в столбце *Comment* прямо в датафрейме. Представленный ниже код поможет это сделать:

```
df <-
  df %>%
  mutate(Comment = mask_text(Comment))
```

Шаг 5. Скопируйте скрипт в Power BI

Загрузите полученную информацию в модель данных Power BI посредством инструмента получения данных (GetData). Ниже приведен полный скрипт:

```
library(tidyverse)
library(stringr)

setwd("<путь к папке с файлом Comments.csv>")

mask_text <- function(unmask_text){
  phone_pattern = "([2-9][0-9]{2})[- .]([0-9]{3})[- .]([0-9]{4})"
  ssn_pattern = "\\d{3}-\\d{2}-\\d{4}"

  cleaned_text <- str_replace(
    unmask_text, pattern = phone_pattern,
    replacement = "XXX-XXX-XXXX")

  cleaned_text <- str_replace(
    cleaned_text, pattern = ssn_pattern,
    replacement = "XXX-XX-XXXX")

  return(cleaned_text)
}

df <- read_csv("Comments.csv")

df <-
  df %>%
  mutate(Comment = mask_text(Comment))
```

Полученный в результате датафрейм со скрытой конфиденциальной информацией будет доступен в Power BI в качестве набора данных, который вы можете при желании загрузить в модель. Эту информацию можно использовать для обогащения вашей модели данных, что позволит строить более разнообразные визуализации.

Защита конфиденциальных сведений в Power BI с помощью Python

В предыдущем примере мы использовали язык R для маскировки телефонных номеров и номеров социального страхования в поле комментариев. То же самое можно сделать с помощью Python – логика будет такая же, а язык другой. Вы можете найти массу задач, в которых языки R и Python будут взаимозаменяемы в плане логики применения. Язык, который вы будете использовать, – это только ваш выбор, зависящий исключительно от личных предпочтений. Итак, давайте выполним операцию маскировки конфиденциальной информации с помощью Python.

Шаг 1. Импортируйте библиотеки *pandas*, *os* и *re*

Как и всегда, начнем с загрузки необходимых библиотек. Первая библиотека, которая нам понадобится, – это *pandas*. Мы будем использовать ее для чтения данных и скрытия секретных сведений в поле *Comment*. Также импортируем пакеты *os* и *re*, которые понадобятся нам для работы с файловой системой и регулярными выражениями соответственно. Ниже приведен код для загрузки нужных нам библиотек:

```
import pandas as pd
import re
import os
```

Шаг 2. Напишите функцию *mask_text()*

Теперь нам необходимо написать свою функцию, которая будет выполнять маскировку конфиденциальной информации в поле *Comment*. Функция будет принимать на вход текст из поля *Comment*, находить по шаблону номера телефонов и номера социального страхования с использованием регулярных выражений, скрывать их в тексте и возвращать итоговую строку с комментарием. Вот что нужно сделать для написания такой функции:

1. Создайте оболочку функции, как показано ниже:

```
def mask_text (unmask_text):
```

Функция будет называться *mask_text()*, и на вход она будет принимать параметр *unmask_text*.

2. Создайте регулярное выражение, соответствующее шаблону американских телефонных номеров, и присвойте его переменной *phone_pattern* следующим образом:

```
phone_pattern = r"([2-9][0-9]{2})[- .]([0-9]{3})[- .]([0-9]{4})"
```

Это один из возможных шаблонов для поиска телефонных номеров в тексте. Этот и многие другие шаблоны для идентификации в том числе номеров телефона и номеров социального страхования вы можете найти на просторах интернета.

Примечание. Обратите внимание, что перед строкой регулярного выражения располагается буква *r*. В Python и в R, начиная с версии 4.0.0, вы можете предварять регулярное выражение буквой *r*, давая понять, что за ней следует *неформатированная строка* (raw string). В результате обратная косая черта будет восприниматься буквально, а не как специальный метасимвол. Это позволит вам не экранировать его, как показано ниже.

3. Напишите регулярное выражение, соответствующее стандартному шаблону номера социального страхования, и присвойте его переменной *ssn_pattern*:

```
ssn_pattern = r"\d{3}-\d{2}-\d{4}"
```

Формат SSN включает в себя три цифры, две и четыре, разделенные дефисами. В представленном шаблоне мы использовали последовательность `\d` для идентификации цифровых символов и `{n}` – для обозначения количества цифр, идущих подряд. Таким образом, последовательность `\d{3}` будет определять три цифровых символа подряд.

В данном примере мы серьезно упростили используемый шаблон, поскольку в действительности в номерах социального страхования могут использоваться не любые цифры. Есть и гораздо более сложные шаблоны для таких номеров, но мы остановимся на этом для легкости восприятия. При работе с регулярными выражениями важно искать компромисс между точностью шаблона и простотой его восприятия.

4. Выполните замену телефонных номеров при помощи следующего кода:

```
cleanned_text = re.sub(
    phone_pattern, "XXX-XXX-XXXX", unmask_text)
```

Здесь мы воспользовались функцией `re.sub()` для замены цифр в телефонных номерах на символы «X». От функции `Text.Replace` языка М, применяющегося в Power Query, ее отличает возможность использования регулярных выражений.

5. Аналогичным образом замаскируйте номера социального страхования в тексте:

```
cleanned_text = re.sub(
    ssn_pattern, "XXX-XX-XXXX", cleaned_text)
```

Шаблон замены мы применяем к результирующему тексту с уже замаскированными номерами телефонов.

6. Верните результат при помощи ключевого слова `return`.
У нас получился следующий код функции:

```
def mask_text (unmask_text):
    phone_pattern = \
        r"([2-9][0-9]{2})[- .]([0-9]{3})[- .]([0-9]{4})"
    ssn_pattern = r"\d{3}-\d{2}-\d{4}"
    cleaned_text = re.sub(
        phone_pattern, "XXX-XXX-XXXX", unmask_text)
    cleaned_text = re.sub(
        ssn_pattern, "XXX-XX-XXXX", cleaned_text)
    return cleaned_text
```

Как и в случае с языком R, мы выделили всю логику в отдельную функцию с именем `mask_text`. Таким образом, чтобы выполнить замену номеров телефонов и номеров социального страхования, нам достаточно будет вызвать нашу функцию. Обособление логики позволяет сделать код более легким для восприятия.

Шаг 3. Установите рабочую директорию

Сделайте рабочей директорией папку, в которой находится файл *Comments.csv*, следующим образом:

```
os.chdir("<путь к файлу Comments.csv")
```

Шаг 4. Считайте комментарии в датафрейм

Выполните чтение из файла *Comments.csv* с использованием библиотеки *pandas* и присвойте результат переменной *df*, как показано ниже:

```
df = pd.read_csv("Comments.csv")
```

Шаг 5. Выполните замену телефонных номеров и номеров социального страхования

В следующем фрагменте кода мы скроем конфиденциальную информацию в поле *Comment*, применив к нему функцию *mask_text()* посредством метода *apply()*:

```
df.Comment = df.Comment.apply(mask_text)
```

Шаг 6. Скопируйте скрипт в Power BI

Загрузите полученную информацию в Power BI посредством инструмента получения данных (GetData). Ниже показан полный скрипт:

```
import pandas as pd
import re
import os

def mask_text (unmask_text):
    phone_pattern = \
        r"([2-9][0-9]{2})[- .]([0-9]{3})[- .]([0-9]{4})"
    ssn_pattern = r"\d{3}-\d{2}-\d{4}"
    cleaned_text = re.sub(
        phone_pattern, "XXX-XXX-XXXX", unmask_text)
    cleaned_text = re.sub(
        ssn_pattern, "XXX-XX-XXXX", cleaned_text)
    return cleaned_text

os.chdir("<path to folder that contains the Comments.csv file")
df = pd.read_csv("Comments.csv")
df.Comment = df.Comment.apply(mask_text)
df
```

Полученный в результате датафрейм со скрытой конфиденциальной информацией будет доступен в Power BI в качестве набора данных, который вы можете при желании загрузить в модель и использовать по своему усмотрению.

ПОДСЧЕТ КОЛИЧЕСТВА СЛОВ И ПРЕДЛОЖЕНИЙ В ОБЗОРАХ

Обсуждая науку о данных (data science), многие сразу представляют алгоритмы, позволяющие вычислить вероятность наступления тех или иных событий. И далеко не все из них знают, что данная отрасль отнюдь не ограничивается такими видами вычислений. В частности, методы науки о данных могут быть использованы при работе с неструктурированным текстом. Эта область известна как *обработка естественного языка* (Natural Language Processing – NLP) и включает в себя различные техники, такие как *анализ тональности текста* (sentiment analysis) и *анализ темы* (topic analysis). В следующей главе мы воспользуемся службой *Microsoft Cognitive Services* для выполнения таких типов задач в Power BI. Но существуют и более простые техники NLP, не требующие выполнения вызовов API к службам вроде *Microsoft Cognitive Services*. Две из них мы рассмотрим в данной главе. А именно мы будем подсчитывать количество слов и предложений в тексте. Давайте опишем сценарий.

Сара работает аналитиком данных в Yelp. Ей поставили задачу сопровождать вспомогательной информацией отчеты для клиентов. В отчеты уже включены отзывы покупателей, а теперь требуется добавить отдельные столбцы с количеством слов и предложений в отзывах. С помощью языка R Саре удалось выполнить оба требования, а с Python она смогла подсчитать только количество слов. Начнем, как и раньше, с решения сценария при помощи R.

Подсчет количества слов и предложений в обзорах с помощью R

В данном разделе мы воспользуемся средствами языка R для анализа текста и подсчета в нем количества слов и предложений. Как и до этого, в основном мы будем пользоваться функционалом пакета *tidyverse*.

Шаг 1. Импортируйте библиотеки *tidyverse* и *stringr*

Два пакета, которые мы будем использовать при решении этого сценария, – это *tidyverse* и *stringr*. Пакетом *dplyr* из коллекции *tidyverse* мы воспользуемся для добавления столбцов в набор данных с количеством слов и предложений в отзывах покупателей. Основная нагрузка при этом ляжет на функцию *str_count()* из пакета *stringr*. Импортировать нужные нам библиотеки можно следующим образом:

```
library(tidyverse)
library(stringr)
```

Шаг 2. Измените рабочую директорию

Как всегда, воспользуемся функцией `setwd()` для установки рабочей директории на папку, в которой содержатся наши данные:

```
setwd("<путь к рабочему файлу>")
```

Шаг 3. Считайте информацию из файла

Воспользуемся функцией `read_csv()` из пакета `readr` для чтения данных в датафрейм. Переменную, хранящую датафрейм, назовем `df`:

```
df <- read_csv("yelp_training_set_review_sample.csv")
```

Шаг 4. Ограничьте набор данных требуемыми столбцами

Исходный набор данных, полученный из файла, содержит массу столбцов, но нам для выполнения задания нужны далеко не все из них. При помощи кода, показанного ниже, можно получить набор нужных нам столбцов:

```
df <-  
  df %>%  
  select(id, business_categories, business_review_count,  
         business_stars, business_state, business_type,  
         stars, text)
```

Шаг 5. Добавьте столбцы с количеством слов и предложений


Теперь добавим два столбца в наш датафрейм: один с именем `word_count`, второй – `sent_count`. В первом будет храниться информация о количестве слов в обзоре, во втором – о количестве предложений. Извлечь эти данные можно при помощи функции `str_count()` из пакета `stringr`, как показано ниже:

```
mutate(  
  word_count = str_count(text, boundary("word")),  
  sent_count = str_count(text, boundary("sentence"))  
)
```

Внутри функции `str_count()` мы вызвали функцию `boundary()`, указывающую на то, какой тип подсчета вы хотите выполнить. Обратите внимание, как легко и просто в R можно выполнить подсчет слов и предложений в тексте. Для этого достаточно вызвать одну функцию. Стандартными средствами Power Query вы не сможете подсчитать количество предложений в тексте, а при помощи языка R это делается элементарно.

Шаг 6. Скопируйте скрипт в Power BI

Полный скрипт приведен ниже:



```
library(tidyverse)
library(stringr)

setwd("<путь к рабочему файлу>")

df <- read_csv("yelp_training_set_review_sample.csv")

df <-
  df %>%
  select(
    id
    ,business_categories
    ,business_review_count
    ,business_stars
    ,business_state
    ,business_type,stars,text) %>%
  mutate(
    word_count = str_count(text, boundary("word"))
    ,sent_count = str_count(text, boundary("sentence"))
  )
```

Полученный в результате датафрейм с дополнительными столбцами будет доступен в Power BI в качестве набора данных, который вы можете загрузить в модель данных.



Подсчет количества слов в обзорах с помощью Python

В данном примере мы используем средства языка Python для подсчета количества слов в отзывах покупателей. Чаще всего языки R и Python бывают взаимозаменяемыми в отношении решения той или иной задачи, но случаются ситуации, когда один из них подходит лучше. Так произошло и на этот раз. В Python метод подсчета количества предложений в тексте оказался не столь очевидным, как в R, так что мы ограничились лишь подсчетом слов. Шаги для решения этой задачи представлены ниже.

Шаг 1. Импортируйте библиотеки *pandas* и *os*

Для выполнения этой задачи нам понадобятся две библиотеки: *pandas* и *os*. Пакет *pandas* мы будем использовать для чтения данных из исходного файла и выполнения подсчета количества слов в тексте, а при помощи библиотеки *os* будем взаимодействовать с операционной системой. Ниже приведен код для импорта библиотек:

```
import pandas as pd
import os
```

Шаг 2. Установите рабочую директорию

Теперь нам необходимо назначить в качестве рабочей директории папку, содержащую необходимый нам файл. Для этого воспользуемся функцией `os.chdir()`, как показано ниже:

```
os.chdir(r"<путь к исходному файлу>")
```

Вы можете использовать путь к файлу в традиционном формате, поскольку перед строкой указали букву *r*, что говорит о применении неформатированной строки. Например, путь вида «C:\Users\Public» прекрасно сработает, поскольку вы не обязаны будете экранировать обратные слешы. В Python такая возможность присутствует уже очень давно, а в R появилась только в версии 4.0.0.

Шаг 3. Считайте информацию из файла

Содержимое файла мы считаем при помощи метода `read_csv()` из пакета *pandas* и поместим его в датафрейм с именем *df*, как показано ниже:

```
df = pd.read_csv("yelp_training_set_review_sample.csv")
```



Шаг 4. Создайте в датафрейме столбец `word_count`

Добавить столбец в датафрейм можно следующим образом:

```
df["word_count"] = df["text"].str.split().apply(len)
```

Форма записи кода получилась очень краткой, но здесь производится масса действий. Давайте разобьем код на части. Первым делом мы вызываем метод `split()` класса *str* для деления исходного текста на слова и сохранения результата в виде списка. Таким образом, если в поле *text* находилась строка «The cat in the hat.», мы получим список, состоящий из пяти элементов: *[The, cat, in, the, hat.]*. Разделителем по умолчанию для функции `split()` является пробел, что нам и нужно в данной ситуации. Если бы мы остановились на этом, то в новом столбце с именем *word_count* оказались бы списки с перечислением слов из исходных строк.

Но мы преобразовали текст в список для того, чтобы позже воспользоваться очень удобной функцией `len()` для получения количества элементов в нем. В нашем случае список состоит из слов, которые были в исходном тексте, так что применение функции `len()` позволит получить их количество. Применить функцию к списку нам поможет метод `apply()`, что и показано во фрагменте кода выше.



Шаг 5. Скопируйте скрипт в Power BI

Полный скрипт получился таким:

```
import pandas as pd
import os
```

```
os.chdir(r"<путь к исходному файлу>")
```

```
df = pd.read_csv("yelp_training_set_review_sample.csv")
df["word_count"] = df["text"].str.split().apply(len)
```

Полученный в результате датафрейм с дополнительным столбцом будет доступен в Power BI в качестве набора данных, который вы можете при желании загрузить в модель.

УДАЛЕНИЕ ИМЕН НЕПОДХОДЯЩЕГО ФОРМАТА

Зачастую исходные данные в источнике находятся совсем не в том виде, в котором нам бы хотелось. В связи с этим в большинстве случаев информация нуждается в дополнительной обработке перед импортом в инструмент бизнес-аналитики.

В данном сценарии Джонатану, ответственному за операции ETL в компании SQLWeave, поставили задачу перед импортом в Power BI очистить таблицу *DimEmployee* от сотрудников, имена которых введены в неправильном формате. Под правильным форматом мы будем понимать один из следующих двух вариантов написания: FN MI LN или FN LN, где FN – это имя, LN – фамилия, а MI – средний инициал. Джонатан написал скрипты на языках R и Python для решения этой задачи. Сначала посмотрим, как можно реализовать наш скрипт в R.

Удаление имен неподходящего формата с помощью R

При решении поставленной задачи мы будем использовать инструменты из пакета *tidyverse*. Вот шаги, которые необходимо выполнить.

Шаг 1. Импортируйте пакеты *tidyverse* и *stringr*

Два пакета, которые мы будем использовать при решении этого сценария, – это *tidyverse* и *stringr*. Пакетом *readr* из коллекции *tidyverse* мы воспользуемся для считывания данных о сотрудниках из файла *DimEmployee.csv* в датафрейм R, а пакет *dplyr* используем для изменения значений в поле *Name*. Сама модификация будет произведена при помощи функции *str_extract()* из пакета *stringr*. Код для загрузки пакетов приведен ниже:

```
library(tidyverse)
library(stringr)
```

Шаг 2. Установите рабочую директорию

Как всегда, функция *setwd()* поможет вам при установке рабочей директории на папку с файлом *DimEmployee.csv*:

```
setwd("<путь к файлу DimEmployee.csv>")
```

Шаг 3. Создайте регулярное выражение с правильным шаблоном имени



Регулярное выражение с правильным шаблоном имени сохраним в переменной *nameWithOrWithoutMiddleInitial*, как показано ниже:

```
nameWithOrWithoutMiddleInitial = "(^[A-Z][a-z]{1,10}\\s[a-zA-Z]\\s[A-Z][a-z]{1,10}$)|(^[A-Z][a-z]{1,10}\\s[A-Z][a-z]{1,10}$)"
```

Внешне это регулярное выражение может выглядеть сложным, но на деле оно гораздо проще, чем кажется. Давайте разделим его на составляющие и разберемся.

1. Как мы уже сказали, наше регулярное выражение будет проверять текст на соответствие двум шаблонам: FN MI LN и FN LN. В связи с этим два шаблона в строке разделены вертикальной чертой, соответствующей логической операции ИЛИ (OR). Ниже часть шаблона, отвечающая за формат имени FN MI LN, написана синим цветом, а шаблон FN LN помечен красным:

```
(^[A-Z][a-z]{1,10}\\s[a-zA-Z]\\s[A-Z][a-z]{1,10}$)|(^[A-Z][a-z]{1,10}\\s[A-Z][a-z]{1,10}$)
```

2. Теперь давайте разберем левую секцию. Помните, что эта секция соответствует шаблону FN MI LN. Сначала рассмотрим первую часть – FN. Регулярное выражение для поиска имени сотрудника выглядит так: `^[A-Z][a-z]{1,10}`. Символ `^` говорит о том, что мы начинаем поиск совпадений с начала строки. Следом идет шаблон `[A-Z]`, указывающий на то, что первой в имени должна быть прописная буква в диапазоне между A и Z. После этого следует конструкция `[a-z]{1,10}`, указывающая на то, что дальше должны идти от одной до десяти строчных букв. Это позволит нам отследить всех сотрудников с длиной имени от 2 до 11 символов.
3. Шаблон среднего инициала представлен в регулярном выражении следующим образом: `\\s[a-zA-Z]\\s`. Последовательность `\\s` в регулярных выражениях означает пробельный символ. Помните, что обратная косая черта является специальным метасимволом, который необходимо экранировать. Средний инициал может быть представлен одиночной строчной или прописной буквой, и шаблон `[a-zA-Z]` в точности соответствует такому критерию. При этом между средним инициалом и фамилией должен стоять пробел, и мы выполнили это требование, вставив в этом месте последовательность `\\s`.
4. Осталось определить, указана ли фамилия сотрудника. Сделаем это при помощи такого регулярного выражения: `[A-Z][a-z]{1,10}$`. Шаблон `[A-Z]` говорит о том, что фамилия должна начинаться с прописной буквы. Далее, как в случае с именем, указана последовательность `[a-z]{1,10}`, позволяющая отследить от одной до десяти идущих подряд строчных букв. Замыкающий символ `$` указывает на окончание строки.
5. Правая часть выражения, соответствующая шаблону FN LN, отличается от левой лишь отсутствием последовательности `\\s[a-zA-Z]\\s`, по-

сколько в этом случае мы ищем только имя и фамилию, без среднего инициала.

Шаг 4. Считайте данные в датафрейм

Обратитесь к файлу *DimEmployee.csv*, считайте данные и сохраните их в датафрейм с именем *df*, как показано ниже:

```
df <- read_csv("DimEmployee.csv")
```

Шаг 5. Выполните обновление столбца Name

Код, необходимый для выполнения этого действия, приведен ниже:

```
df <-  
  df %>%  
  mutate(Name = str_extract(Name, nameWithOrWithoutMiddleInitial))
```

Функция *mutate()* используется для изменения столбца. Сначала функция *str_extract()* из пакета *stringr* проверяет поле *Name* на совпадение с регулярным выражением, сохраненным в переменной *nameWithOrWithoutMiddleInitial*. Если совпадение обнаружено, из столбца *Name* будет извлечен текст, соответствующий шаблону. В противном случае ничего не будет возвращено.

Шаг 6. Скопируйте скрипт в Power BI

На заключительном шаге нужно просто перенести полный текст скрипта, показанный ниже, во встроенный редактор R в Power BI:

```
library(tidyverse)  
library(stringr)  
  
setwd("C:/Users/rwade/Downloads/Chapter5/Chapter5")  
  
nameWithOrWithoutMiddleInitial = "(^[A-Z][a-z]{1,10}\\s[a-zA-Z]\\s[A-Z][a-z]{1,10}$)|(^[A-Z][a-z]{1,10}\\s[A-Z][a-z]{1,10}$)"  
  
df <- read_csv("DimEmployee.csv")  
  
df <-  
  df %>%  
  mutate(Name = str_extract(Name, nameWithOrWithoutMiddleInitial))
```

Результирующий набор данных будет доступен в Power BI и может быть загружен в модель данных.

Удаление имен неподходящего формата с помощью Python

Теперь посмотрим, как можно решить эту задачу при помощи Python. Для этого нам понадобятся библиотеки *pandas*, *re* и *os*. Пройдемся по шагам.

Шаг 1. Импортируйте библиотеки *pandas*, *re* и *os*

В данном скрипте мы будем использовать библиотеки *pandas*, *re* и *os*. Пакет *pandas* понадобится нам для считывания данных из файла *DimEmployee.csv* и загрузки в датафрейм, а также для выполнения изменений в столбце *Name*. При сопоставлении данных мы будем пользоваться инструментами для работы с регулярными выражениями, предоставляемыми библиотекой *re*, а пакет *os* мы используем для взаимодействия с файловой системой. Вот код, который поможет вам импортировать все нужные библиотеки:

```
import pandas as pd
import re
import os
```

Шаг 2. Установите рабочую директорию

Здесь мы снова воспользуемся функцией *os.chdir()* для установки рабочей директории в папку, содержащую нужный нам файл *DimEmployee.csv*:

```
os.chdir(r"<путь к файлу DimEmployee.csv>")
```

Шаг 3. Считайте данные из файла *DimEmployee.csv* в датафрейм

На данном шаге мы воспользуемся методом *read_csv()* из библиотеки *pandas*, чтобы считать содержимое файла *DimEmployee.csv* в датафрейм и сохранить в переменной *df*, как показано ниже:

```
df = pd.read_csv("DimEmployee.csv")
```

Шаг 4. Создайте регулярное выражение, соответствующее правильному формату имени

Регулярное выражение, которое мы используем для сопоставления имен сотрудников с принятым шаблоном, будет таким:

```
r'([A-Z][a-z]{1,10}\s[a-zA-Z]\s[A-Z][a-z]{1,10})|' \
r'([A-Z][a-z]{1,10}\s[A-Z][a-z]{1,10})'
```

Выражение будет компилироваться при помощи метода *compile()* из пакета *re*, а результат будет присвоен объектной переменной *nameWithOrWithout-MiddleInitial*.

Анализ регулярного выражения мы повторно приводить не будем, поскольку он будет таким же, как в примере с использованием языка R, за исключением того, что здесь мы использовали неформатированную строку, чтобы не было необходимости дополнительно экранировать все обратные слеша, а также не применяли в выражении символы *^* и *\$*.

Шаг 5. Скомпилируйте регулярное выражение

Код, необходимый для компиляции регулярного выражения в Python, приведен ниже:

```
namepattern = \
    r'([A-Z][a-z]{1,10})\s[a-zA-Z]\s[A-Z][a-z]{1,10})|' \
    r'([A-Z][a-z]{1,10})\s[A-Z][a-z]{1,10})'

nameWithOrWithoutMiddleInitial = re.compile(namepattern)
```

Первое, что мы сделали, — это присвоили наше регулярное выражение переменной *namepattern*. После этого мы создали объект шаблона с именем *nameWithOrWithoutMiddleInitial* при помощи метода *compile()* из библиотеки *re*. Этот объект мы будем использовать на следующих шагах для сопоставления имен сотрудников с принятым шаблоном.

Шаг 6. Напишите функцию для проверки имен на совместимость с шаблоном

Давайте создадим функцию с именем *scrubName*, которая будет выполнять проверку на соответствие регулярному выражению:

```
def scrubName(name):
    m = nameWithOrWithoutMiddleInitial.fullmatch(name)

    if m:
        return m.group(0)
    else:
        return None
```

Функция принимает единственный аргумент — имя, которое необходимо проверить. Далее это имя передается в метод *fullmatch()* созданного ранее объекта шаблона *nameWithOrWithoutMiddleInitial*. Этот метод проверяет переданное имя на полную совместимость с регулярным выражением. Заметьте, что здесь мы не использовали в регулярном выражении символы начала и конца шаблона *^* и *\$*, поскольку выполняем полный поиск.

Результат вызова метода сохраняется в переменной *m*. Если совпадение будет обнаружено, будет возвращен объект типа *Match Object*. В противном случае мы получим на выходе *None*, что является аналогом *null* в Python. Объект *Match Object* обладает разнообразными методами. Мы же воспользуемся методом *group()*. Давайте проиллюстрируем его на простом примере:

```
import re
p = re.compile("(Ryan)\\s(Wade)")
m = p.fullmatch("Ryan Wade")
m.group(0)
m.group(1)
m.group(2)
```

Здесь я использовал регулярное выражение с моим полным именем. При этом имя и фамилию я разнес по группам, заключив их в круглые скобки. Таким образом, если передать в метод `fullmatch()` строку «Ryan Wade», совпадение будет найдено, а результат будет разбит на группы. Использование метода `group()` с аргументом 0 (`m.group(0)`) вернет полное совпадение «Ryan Wade». Если же мне нужно вернуть лишь имя «Ryan», я могу использовать запись `m.group(1)`, а для возврата фамилии «Wade» – `m.group(2)`.

Но давайте вернемся к нашей функции `scrubName()`. При обнаружении совпадения нам нужно вернуть весь текст, поэтому мы использовали запись `m.group(0)`. В противном случае мы возвращаем `None`.

Шаг 7. Примените функцию к столбцу, чтобы избавиться от лишних имен

Это можно сделать при помощи следующего кода:

```
df["Name"] = df.Name.apply(scrubName)
```

Мы вызвали метод `apply()` у поля `Name`, чтобы применить функцию `scrubName` к его содержимому.

Шаг 8. Скопируйте скрипт в Power BI

На заключительном шаге просто перенесите полный текст скрипта, показанный ниже, во встроенный редактор Python в Power BI:

```
import pandas as pd
import re
import os

os.chdir(r"<путь к файлу DimEmployee.csv>")
df = pd.read_csv("DimEmployee.csv")

namepattern = \
    r'([A-Z][a-z]{1,10}\s[a-zA-Z]\s[A-Z][a-z]{1,10})|' \
    r'([A-Z][a-z]{1,10}\s[A-Z][a-z]{1,10})'

namewithOrWithoutMiddleInitial = re.compile(namepattern)

def scrubName(name):
    m = namewithOrWithoutMiddleInitial.fullmatch(name)
    if m:
        return m.group(0)
    else:
        return None

df["Name"] = df.Name.apply(scrubName)
```

Результирующий датафрейм будет доступен в Power BI для загрузки в модель данных.

ОПРЕДЕЛЕНИЕ ШАБЛОНОВ В СТРОКАХ НА ОСНОВАНИИ УСЛОВНОЙ ЛОГИКИ

Большинство операций со строками можно без труда выполнить при помощи стандартного инструментария Power Query. Однако если анализ строк основывается на условной логике, код может стать весьма запутанным и трудным для восприятия. К счастью, во многих случаях условную логику при обработке строк можно легко и просто реализовать в R или Python при помощи знакомых уже вам регулярных выражений. В данном примере мы посмотрим, как в вымышленной компании *MC Diesel* справились с задачей разбора строк на основании условной логики с применением регулярных выражений.

Родни, исполнительный директор компании, осведомлен о том, что в прошлом у *MC Diesel* были проблемы с двумя позициями: топливным насосом (внутренний номер 561769) и инжектором (номер 561394). Проблемы были обнаружены в производственном процессе на заводах в штатах Алабама (AL), Луизиана (LA) и Огайо (OH) и проявлялись только в моделях A и B.

Родни поручил аналитику данных Энни Сью разработать детализированный дашборд с целью дальнейшей диагностики и мониторинга похожих проблем. При этом в распоряжении Энни был единственный отчет, показанный в табл. 7.1.

Таблица 7.1. Отчет о производстве деталей

Date Key	SKU	Shift	Store ID	Recalled
20190813	75197726D3	3	263	False
20191211	70136522D1	1	221	False
20190401	70195609A3	3	92	False
20191228	75102709B1	1	91	False
20190304	70162016C3	3	161	True
20191218	70136526D2	2	263	False
20190322	70162029D1	1	291	False
20191012	70136539C2	2	395	False
20190318	56176926B2	2	262	False

На первый взгляд, в этом отчете нет никакой информации о месте производства детали и ее номере, а именно эти данные нужны нам для анализа. Но Энни оказалась очень сообразительной и вспомнила, что в поле SKU цифры и буквы собраны далеко не в случайном порядке, – каждый символ здесь что-то значит. Девушка покопалась в документации и обнаружила, что первые шесть цифр в поле означают номер детали, цифры в позициях 7 и 8 указывают на FIPS-код завода, на котором деталь была произведена, буква в девятой позиции характеризует модель, а заключительная цифра – смену, в которую она была сделана. Таким образом, SKU 56176922A2 может быть расшифровано следующим образом:

- это топливный насос, поскольку первые шесть цифр 561769 относятся именно к этой производственной группе;

- деталь была произведена в Луизиане, поскольку две двойки, расположенные в позициях 7 и 8, представляют собой FIPS-код этого штата;
- буква A в девятой позиции явно указывает на модель изделия;
- насос был произведен во вторую смену, о чем свидетельствует двойка на конце SKU.

Энни необходимо создать вычисляемый столбец, возвращающий номер детали, если выполняются следующие условия:

- это топливный насос модели A или B, произведенный в одном из трех штатов: Алабама (AL), Луизиана (LA) или Огайо (OH);
- это инжектор модели A или B, произведенный в одном из трех штатов: Алабама (AL), Луизиана (LA) или Огайо (OH).

Решение этого сценария в Power Query потребовало бы написания объемного и непонятного кода, тогда как при помощи R или Python задача решается весьма лаконично и изящно. Сначала, по традиции, посмотрим на решение в среде R.



Поиск шаблонов в строках на основании условной логики с помощью R

В нашем решении мы воспользуемся регулярными выражениями для поиска совпадений с определенным ранее критерием. На этом примере мы увидим, с какой легкостью можно описывать логику нахождения нужных нам SKU. В Power Query с применением языка R нам бы пришлось писать гораздо более запутанный и многословный код. Итак, пройдем по шагам.

Шаг 1. Импортируйте пакеты *tidyverse* и *stringr*

Сначала, как и всегда, загрузим нужные нам библиотеки. Пакет *readr* из коллекции *tidyverse* понадобится нам для чтения данных из файла *ProductionOrders.csv* и сохранения их в датафрейм. Библиотекой *dplyr* мы воспользуемся для выполнения манипуляций с датафреймом, а пакет *stringr* используем для операций со строками. Ниже представлен код для импорта нужных нам пакетов:

```
library(tidyverse)
library(stringr)
```

Шаг 2. Установите рабочую директорию

На этом шаге вам необходимо установить рабочую директорию на папку с файлом *ProductionOrders.csv* при помощи функции *setwd()*, как показано ниже:

```
setwd("<путь к файлу ProductionOrders.csv>")
```



Шаг 3. Напишите функцию для поиска изделий

Назовем нашу функцию `monitoredProducts()`. Ниже приведены шаги по ее созданию.

1. Напишите оболочку функции при помощи следующего кода:

```
monitoredProducts <- function(sku){
}
```

Функция принимает на вход единственный параметр – номер SKU.

2. Теперь нам необходимо написать регулярное выражение, которое отслеживало бы все топливные насосы и инжекторы (номера изделий 561769 и 561394) моделей А и В, произведенные в штатах Алабама (AL), Луизиана (LA) или Огайо (OH). При этом возвращать мы будем не весь номер SKU, а только номер изделия. Для этого идеально подойдет так называемая *опережающая проверка* (positive lookahead).

На сайте www.rexegg.com можно прочитать о том, что опережающая проверка позволяет определить совпадение с шаблоном в строке непосредственно за текущей позицией. Взгляните на регулярное выражение, приведенное ниже. Давайте вместе разберемся, что оно означает:

```
pattern = "^(561769|561394)(?=(01|22|39)(A|B))"
```

Часть выражения, отвечающая за опережающую проверку, выглядит так: `(?=(01|22|39)(A|B))`. Структурно выражение опережающей проверки разделено на группы в круглых скобках и начинается с последовательности символов `?=`. Шаблон после этих символов должен следовать сразу за предшествующим указанным шаблоном, чтобы совпадение было зафиксировано.

В нашем случае совпадение с шаблоном будет обнаружено при выполнении следующих трех условий:

- строка начинается с 561769 или 561394;
- далее должен следовать FIPS-код нужного нам штата (01, 22 или 39);
- символ, указывающий на модель изделия, должен быть А или В.

На шаге 5, где мы воспользуемся нашей функцией, вы лучше поймете, что происходит.

3. Теперь нам нужно извлечь номер изделия в случае совпадения с нашим регулярным выражением. Сделать это можно следующим образом:

```
mp = str_extract(sku, pattern = pattern)
```

4. Здесь мы воспользовались функцией `str_extract()` для извлечения строк, совпадающих с определенным шаблоном. При этом будет возвращена только часть кода SKU, соответствующая номеру изделия. Давайте посмотрим, как это работает, на примере. Если передать функции строку «56176939A2», она вернет «561769». Причина в том, что наш SKU не только начинается с нужного нам номера 561769, но и содержит далее входящие в наш список код штата (39) и модель (A). Напротив, если

передать функции SKU с номером «56176910D2», будет возвращено значение *NA*, поскольку код штата 10 и модель D не входят в сферу наших интересов.

Шаг 4. Считайте данные из файла *ProductionOrders.csv* в датафрейм

На этом шаге мы загрузим содержимое файла *ProductionOrders.csv* в датафрейм и сохраним в переменной с именем *df*. Сделать это можно так:

```
df <- read_csv("ProductionOrders.csv")
```



Шаг 5. Добавьте в датафрейм *df* столбец *Monitored Products*

Теперь мы напишем код, позволяющий добавить новый столбец с именем *Monitored Products* в наш датафрейм с помощью функции *mutate()* из пакета *dplyr*. Код представлен ниже:

```
df <-
  df %>%
  mutate(`Monitored Products` = monitoredProducts(SKU))
```

Здесь мы воспользовались функцией *mutate()* из пакета *dplyr* для создания новой колонки *Monitored Products*, значения в которой будут зависеть от результатов применения пользовательской функции *monitoredProducts()* к столбцу *SKU*.

Шаг 6. Скопируйте скрипт в Power BI

Полный скрипт приведен ниже:

```
library(tidyverse)
library(stringr)

setwd("<путь к файлу ProductionOrders.csv>")

monitoredProducts <- function(sku){
  pattern = "^(561769|561394)(?(01|22|39)(A|B))"
  mp = str_match(sku, pattern)[1,1]
  return_value = ifelse(is.na(mp), "Not Monitored", mp)
  return(return_value)
}

vmonitoredProducts = Vectorize(monitoredProducts)

df <- read_csv("ProductionOrders.csv")

df <-
  df %>%
  mutate(`Monitored Products` = vmonitoredProducts(SKU))
```



Функция `Vectorize()` производит векторизацию функции `monitoredProducts()` (прим. перев.).

Датафрейм `df`, полученный на выходе, может быть загружен в модель данных Power BI при помощи инструмента получения данных (`GetData`).

Поиск шаблонов в строках на основании условной логики с помощью Python

В данном разделе мы произведем ту же операцию, но с использованием языка Python. Как и в случае с R, здесь мы главным образом будем опираться на мощь регулярных выражений. Как вы знаете, взаимодействие с регулярными выражениями в Python реализовано несколько иначе по сравнению с R. Давайте пройдемся по шагам.

Шаг 1. Импортируйте библиотеки *pandas*, *re* и *os*

Здесь, как и в предыдущем примере, нам понадобятся три вспомогательные библиотеки: *pandas*, *re* и *os*. Пакет *pandas* мы используем для чтения данных из файла и добавления столбца в датафрейм. Библиотека *re* нам пригодится для работы с регулярными выражениями, а пакет *os* позволит выполнить нужное нам взаимодействие с файловой системой. Код импорта необходимых библиотек приведен ниже:

```
import pandas as pd
import re
import os
```

Шаг 2. Установите рабочую директорию

Здесь мы снова воспользуемся функцией `os.chdir()` для установки рабочей директории в папку, содержащую нужный нам файл *ProductionOrders.csv*:

```
os.chdir(r"<путь к файлу ProductionOrders.csv>")
```

Шаг 3. Скомпилируйте регулярное выражение

На этом шаге нам необходимо создать регулярное выражение для поиска топливных насосов (номер изделия 561769) и инжекторов (номер изделия 561394) модели A или B, произведенных в штатах Алабама (AL), Луизиана (LA) или Огайо (OH). Также нам необходимо получить только номер детали из поля *SKU*, а не весь код целиком. Как и в примере с R, мы будем делать это при помощи опережающей проверки.

Подробное описание регулярного выражения приведено в предыдущем разделе с R, здесь же мы просто повторим его:

```
r"^(561769|561394)(?=(01|22|39)(A|B)\d{1})$"
```

Здесь мы использовали букву *r* для сообщения информации о том, что имеем дело с неформатированной строкой. Окончание `\d{1}$` добавлено для более точного поиска – с финальной цифрой, характеризующей номер смены.

После определения регулярного выражения необходимо создать объект при помощи функции `compile()` из пакета `re`. Это можно сделать следующим образом:

```
pattern = re.compile(r"^(561769|561394)(?=(01|22|39)(A|B)\d{1}$")
```

Шаг 4. Напишите функцию для распознавания нужных нам деталей

Назовем нашу функцию `monitoredProducts()`. Вот шаги, которые необходимо выполнить для ее написания.

1. Для начала создайте оболочку функции, принимающей на вход единственный параметр, – SKU изделия:

```
def monitoredProducts(sku):
```

2. Далее вам необходимо создать объект *Match Object* с использованием шаблона, определенного ранее. Это можно сделать следующим образом:

```
m = pattern.match(sku)
```

Если переданный код SKU успешно пройдет проверку регулярным выражением, будет возвращен объект *Match* для извлечения из него групп. В противном случае вернется *None* – аналог *null* в Python.

3. Теперь извлечем полный шаблон возвращенного значения из объекта *Match*. Сделать это можно так:

```
if m == True:
    return m.group(0)
else:
    return "Not Monitored"
```



Как мы уже говорили, в случае нахождения соответствия будет возвращен объект *Match* с булевым значением *True*. В противном случае мы получим *None*. Условная конструкция *if* используется для определения того, что нам вернулся именно объект *Match*. Если это так, извлекаем группу с индексом 0, которая представляет собой весь шаблон целиком. Иначе возвращаем текст «Not Monitored».

Шаг 5. Считайте данные в датафрейм Pandas

На этом шаге мы должны считать содержимое файла *ProductionOrders.csv* и сохранить его в датафрейм с именем *df*:

```
df = pd.read_csv("ProductionOrders.csv")
```

Шаг 6. Создайте новый столбец с именем *Monitored Products*

Здесь мы создадим новую колонку в датафрейме с именем *Monitored Products* с использованием следующего кода:

```
df["Monitored Products"] = df["SKU"].apply(monitoredProducts)
```

На этом этапе мы применяем созданную ранее пользовательскую функцию *monitoredProducts()* к столбцу *SKU* и присваиваем результат новой колонке *Monitored Products*.

Шаг 7. Скопируйте скрипт в Power BI

Ниже приведен полный скрипт:

```
import pandas as pd
import re
import os

os.chdir(r"<путь к файлу ProductionOrders.csv>")

pattern = re.compile(r"^(561769|561394)(?=(01|22|39)(A|B)\d{1}$)")

def monitoredProducts(sku):
    m = pattern.match(sku)
    if m == True:
        return m.group(0)
    else:
        return "Not Monitored"

df = pd.read_csv("ProductionOrders.csv")

df["Monitored Products"] = df["sku"].apply(monitoredProducts)
```

Результирующий датафрейм будет доступен в Power BI для загрузки в модель данных.

ЗАКЛЮЧЕНИЕ

В данной главе вы освоили различные методы манипуляции со строками в Power BI при помощи языков R и Python. Конечно, рассмотренные здесь примеры не отвечают в полной мере вашим бизнес-требованиям. Но, уверен, с незначительной доработкой вы сможете с пользой применить их на практике.



Глава 8

.....

Вычисляемые столбцы с помощью R и Python

При помощи стандартных средств Power Query добавить в модель данных вычисляемые столбцы со сложными математическими формулами бывает не так просто. Но, как вы догадываетесь, в R и Python таких проблем не существует. Язык R изначально был создан для работы со статистикой, так что для него вполне естественно построение сложных вычислений. То же верно и для Python. Кроме того, Python активно используется не только в науке о данных, но и в области физики и инженерии. Подобно R, Python оптимизирован для произведения вычислений, недоступных в языке M, который используется в Power Query. Так что всякий раз, когда вам необходимо добавить вычисляемый столбец в модели данных Power BI, вы должны рассмотреть вариант использования R или Python. И дело не только в том, что в этих языках производить такие вычисления легче, — зачастую вы с удивлением будете обнаруживать, что в них уже есть готовые функции, которые выполняют всю тяжелую работу за вас.

Давайте проиллюстрируем сказанное на примере. Представьте, что вы старший аналитик компании Gee's Trucking, Inc. Недавно ваша компания приобрела другую компанию-перевозчика, и теперь в вашем штате числится более ста водителей, работающих в трех терминалах Индианаполиса.

Владелец компании задался вопросом о том, какой путь до работы преодолевают его сотрудники, чтобы вычислить среднюю дистанцию до работы. Целью этого исследования стала возможность сократить путь водителей до рабочего места путем обмена терминалами с коллегами. А вы наверняка со школьной скамьи помните, как вычислить расстояние между двумя географическими координатами с помощью *формулы гаверсинуса* (Haversine formula).

Примечание. Формулу гаверсинуса можно использовать для расчета расстояния между двумя точками на Земле, зная их широту и долготу. Данная формула учитывает кривизну земной поверхности.

Вы разработали для менеджера четыре решения: два с использованием R и два — с Python. При этом в каждом языке одно решение базировалось на собственных расчетах, а второе — на готовых средствах. Во всех четырех

случаях адреса проживания водителей и адреса терминалов должны быть геокодированы.

Примечание. *Геокодирование* (geocoding) представляет собой процесс преобразования адресов в географические координаты с использованием широты и долготы. Широта определяет, насколько географическая точка отстоит от экватора в сторону севера или юга, а долгота – насколько она отстоит от нулевого меридиана к западу или востоку.

Ниже представлен общий план решения задачи.

1. Сгенерируйте ключ Google API для обращения к службе *Google Geocoding API*.
2. Геокодируйте адреса с использованием службы *Google Geocoding API* при помощи R и Python.
3. Вычислите расстояние между всеми домашними адресами водителей и адресами их рабочих терминалов с использованием пользовательской функции в R и Python.
4. Вычислите расстояние между домашними адресами водителей и адресами их рабочих терминалов с использованием готовой функции в R и Python.

Начнем с генерирования ключа Google API.

СОЗДАНИЕ КЛЮЧА GOOGLE GEOCODING API

Шаг 1. Зайдите в консоль Google

Зайдите в консоль Google по адресу <https://console.cloud.google.com>. Если у вас еще нет аккаунта в Google, самое время его завести.

Шаг 2. Настройте учетную запись

Для использования Google API вам необходимо указать способ оплаты в вашем аккаунте. На момент написания этой книги Google предоставлял кредит на сумму \$300 для опробования своего API, чего будет вполне достаточно для выполнения представленных здесь заданий. При превышении кредитной суммы Google использует ваши данные для совершения новых платежей. Для настройки способа оплаты необходимо в меню **Billing** выбрать пункт **Payment method**, как показано на рис. 8.1.

Шаг 3. Добавьте новый проект

Теперь вам необходимо создать проект со включенной опцией геокодирования. Для этого в меню **IAM & admin** выберите пункт **Manage resources**, как показано на рис. 8.2.

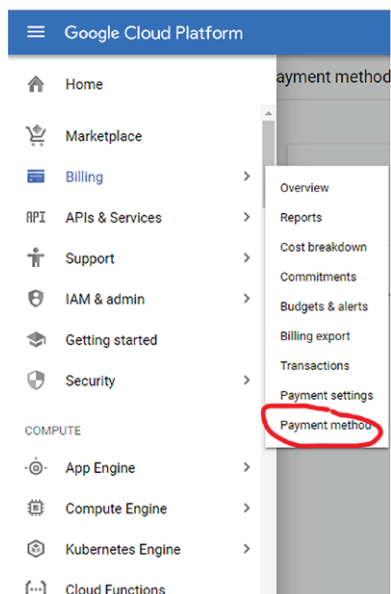


Рис. 8.1 ❖ Настройка способа оплаты в Google

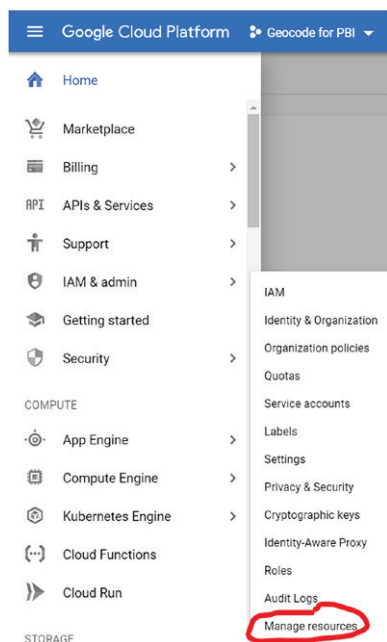


Рис. 8.2 ❖ Навигация к управлению ресурсами

Нажмите на кнопку **Create Project**, расположенную в верхнем правом углу экрана, как продемонстрировано на рис. 8.3.

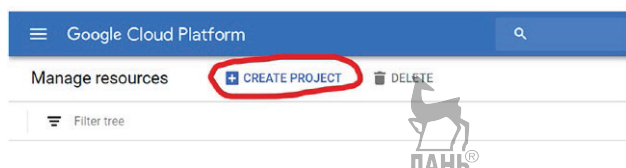


Рис. 8.3 ❖ Создание нового проекта Google

Заполните форму для создания проекта.

Шаг 4. Активируйте API геокодирования

Выполните следующие действия для активации функции геокодирования в вашем проекте.

1. В меню **APIs & Services** выберите пункт **Library**, как показано на рис. 8.4.
2. Введите *Geocoding API* в поле поиска **Search for APIs & Services**, как видно на рис. 8.5.

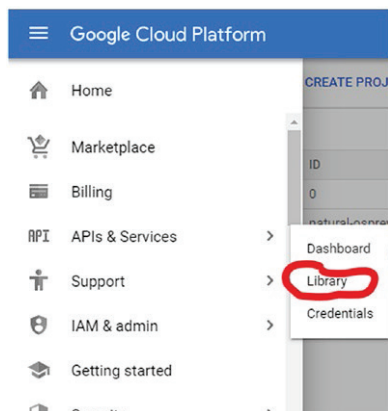


Рис. 8.4 ❖ Пункт Library в меню

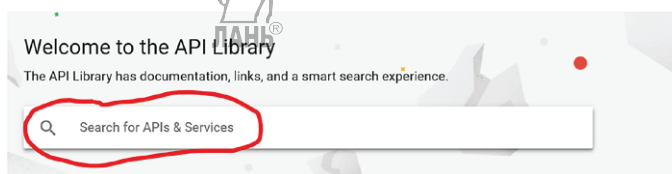


Рис. 8.5 ❖ Поиск Geocoding API

3. Активируйте **Geocoding API** путем нажатия на кнопку **Enable**.
4. Создайте учетную запись следующим образом:
 - выберите в меню **APIs & Services** пункт **Credentials**;
 - нажмите на кнопку **Create credentials** и выберите пункт **API key**, как показано на рис. 8.6;

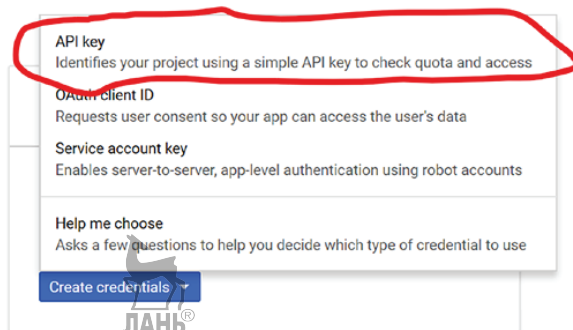
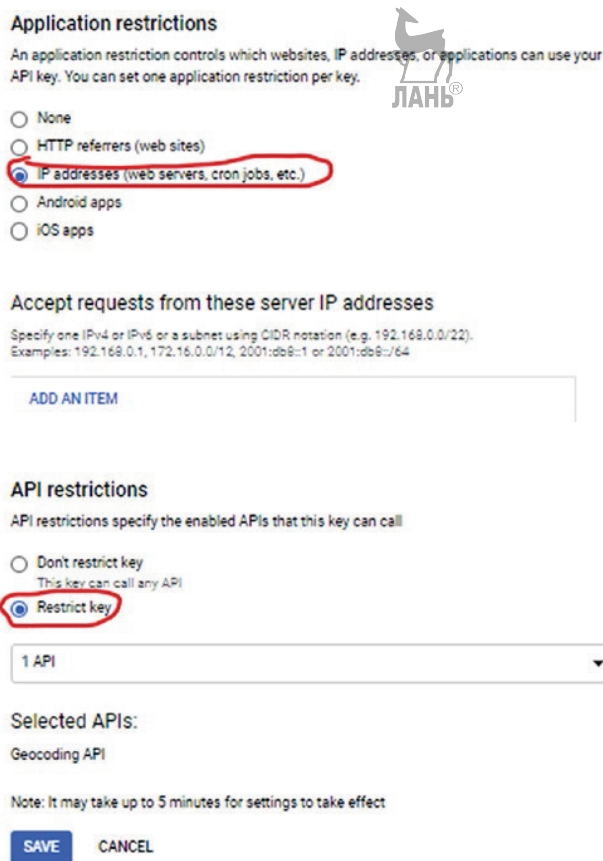


Рис. 8.6 ❖ Создание ключа API

- должен появиться ключ API, и у вас должна быть возможность ввести необходимые вам ограничения на его использование. Вам нужно защитить свой ключ API от несанкционированного использования.

Нажмите на кнопку **Restrict Key**, чтобы настроить ограничения. Вы увидите форму, показанную на рис. 8.7. Для данного примера я рекомендую разрешить доступ к API только с вашего IP-адреса и ограничить использование ключа одним *Geocoding API*. Вы можете сделать это, установив настройки так, как показано на рис. 8.7. После завершения настройки нажмите на кнопку **Save** для сохранения установок.



Application restrictions

An application restriction controls which websites, IP addresses, or applications can use your API key. You can set one application restriction per key.

- ☐ None
- ☐ HTTP referers (web sites)
- ☒ IP addresses (web servers, cron jobs, etc.)
- ☐ Android apps
- ☐ iOS apps

Accept requests from these server IP addresses

Specify one IPv4 or IPv6 or a subnet using CIDR notation (e.g. 192.168.0.0/22).
Examples: 192.168.0.1, 172.16.0.0/12, 2001:db8::1 or 2001:db8::/64

[ADD AN ITEM](#)

API restrictions

API restrictions specify the enabled APIs that this key can call

- ☐ Don't restrict key
This key can call any API
- ☒ Restrict key

1 API

Selected APIs:

Geocoding API

Note: It may take up to 5 minutes for settings to take effect

SAVE **CANCEL**

Рис. 8.7 ❖ Настройка ограничений API

5. Теперь у вас есть ключ API с настроенными ограничениями. Сохраните его в надежном месте. В случае утери ключа вы всегда можете зайти в консоль Google и получить его копию.

Геокодирование адресов с помощью R

Теперь, когда вы настроили свой ключ Google API, вы можете воспользоваться этим программным интерфейсом для геокодирования адресов в файле

EmployeeList.csv. Процесс геокодирования заключается в преобразовании адресов в географические координаты на основе *широты* (latitude) и *долготы* (longitude). Широта характеризует отстояние точки от экватора в сторону севера или юга, а долгота – от нулевого меридиана к западу или востоку. Наличие адресов в виде географических координат позволит вам применить к ним методы *пространственной аналитики* (spatial analytics). В частности, в данном примере мы будем рассчитывать расстояние между двумя адресами. В листинге ниже показано, как можно обращаться в языке R к Google API с целью геокодирования адресов:

```
library(tidyverse)
library(ggmap)

register_google(key = "<ваш ключ API>")

setwd("<путь к файлу EmployeeList.csv>")
EmployeeList <- read_csv("EmployeeList.csv")

EmployeeList <-
  EmployeeList %>%
    mutate_geocode(EmployeeAddress, sensor = FALSE) %>%
    rename(lon_EmployeeAddress = lon, lat_EmployeeAddress=lat) %>%
    mutate_geocode(TerminalAddress, sensor = FALSE) %>%
    rename(lon_TerminalAddress = lon, lat_TerminalAddress = lat)

write_csv(EmployeeList, "EmployeeList.csv")
```

На этом этапе чтения книги вы уже должны достаточно хорошо понимать синтаксис языка R, так что я не буду очень глубоко вдаваться в подробности приведенного выше фрагмента кода. Предоставлю лишь беглое описание с основными принципами.

1. Сначала мы импортируем необходимые пакеты. Здесь у нас появляется новый пакет *ggmap*, с которым мы раньше не сталкивались. Главное предназначение этого пакета заключается в визуализации пространственных данных, но в данном случае мы воспользуемся им для геокодирования адресов.
2. Мы уже решили, что будем использовать службу *Google Geocoding API* для геокодирования адресов. Для этого вам сначала необходимо зарегистрировать свой ключ API при помощи функции *register_google()*.
3. После этого считаем содержимое файла *EmployeeList.csv*.
4. Затем мы воспользуемся пакетом *dplyr* и функцией *mutate_geocode()* из библиотеки *ggmap* с целью добавления географических сведений в наш набор данных. Подобно функции *mutate()* из пакета *dplyr*, функция *mutate_geocode()* используется для добавления столбцов в датафрейм. По сути, функция *mutate_geocode()* добавляет колонки с географическими координатами, исходя из переданного адреса. Столбец с долготой получает имя *lon*, а столбец с широтой – *lat*.
5. В завершение мы воспользуемся функцией *rename()* из пакета *dplyr* для придания новым столбцам более осмысленных имен.

Примечание. С помощью пакета *dplyr* можно двумя разными способами переименовать столбцы в датафрейме – воспользоваться функцией *rename()* или функцией *select()*. Отличие состоит в том, что *rename()* сохраняет в выборке все остальные столбцы датафрейма, а *select()* оставляет в наборе данных лишь те колонки, которые указаны в аргументах явно.

6. Повторяем шаги 4 и 5 для поля *TerminalAddress*.

В рассмотренном примере мы продемонстрировали вариант взаимодействия с Google API в R для геокодирования адресов. Обращаться к API непосредственно из Power Query не так просто, если вы не обладаете специальными знаниями. В версии Power BI Premium вы можете воспользоваться службой *Microsoft Cognitive Services*. При использовании языков R и Python у вас не будет такого ограничения. В данном разделе мы обращаемся к службе Google API для геокодирования адресов, а далее в этой книге покажем, как можно взаимодействовать с другими API, такими как *IBM Watson Natural Language Understanding*. Сейчас же посмотрим, как можно реализовать геокодирование адресов при помощи Python.

ГЕОКОДИРОВАНИЕ АДРЕСОВ С ПОМОЩЬЮ PYTHON

В предыдущем разделе мы узнали, как можно выполнить геокодирование адресов с использованием Google API в R. Сейчас мы сделаем то же самое в Python. Как мы уже упоминали, работа с программными интерфейсами непосредственно из Power Query – задача не из легких, если вы не являетесь счастливым обладателем дорогостоящей версии Power BI Premium. В скрипте, приведенном ниже, показан пример геокодирования адресов в Python при помощи обращения к Google API:

```
import pandas as pd
import os
from geopy import geocoders

g_api_key = "<ключ API>"
g = geocoders.GoogleV3(g_api_key)

os.chdir("<путь к файлу EmployeeList.csv>")
EmployeeList = pd.read_csv("EmployeeList.csv")

EmployeeList["EmployeeAddressGC"] = \
    EmployeeList["EmployeeAddress"].apply(g.geocode)
EmployeeList["lat_EmployeeAddress"] = \
    EmployeeList["EmployeeAddressGC"].apply(lambda x: x.latitude)
EmployeeList["lon_EmployeeAddress"] = \
    EmployeeList["EmployeeAddressGC"].apply(lambda x: x.longitude)
EmployeeList["TerminalAddressGC"] = \
    EmployeeList["TerminalAddress"].apply(g.geocode)
EmployeeList["lat_TerminalAddressGC"] = \
    EmployeeList["TerminalAddressGC"].apply(lambda x: x.latitude)
```

```
EmployeeList["lon_TerminalAddress"] = \
    EmployeeList["TerminalAddressGC"].apply(lambda x: x.longitude)

cols = ["EmployeeAddressGC", "TerminalAddressGC"]
EmployeeList.drop(cols, inplace=True)

EmployeeList.to_csv("EmployeeList.csv", index = False)
```

Опять же, поскольку на данный момент вы должны уже неплохо ориентироваться в синтаксисе языка Python, мы лишь бегло пройдемся по шагам в этом скрипте.

1. Начнем с импорта нужных нам библиотек. Здесь мы впервые встречаемся с модулем *geocoders* из пакета *geopy*. Именно при помощи функций из этого модуля мы будем производить геокодирование адресов.
2. После этого создаем объект *g*, базирующийся на классе Google API *GoogleV3*. Этот объект будет использоваться для выполнения запросов к GoogleV3 API с целью геокодирования адресов.
3. Далее читаем содержимое файла *EmployeeList.csv*.
4. В нашем наборе данных есть два адреса, нуждающихся в геокодировании: *EmployeeAddress* и *TerminalAddress*. Сначала геокодируем поле *EmployeeAddress* путем применения к нему метода *geocode()* объекта *g* при помощи следующего кода:

```
EmployeeList["EmployeeAddressGC"] = EmployeeList["EmployeeAddress"].apply(g.geocode)
```

В результате будет создан объект *location.Location*, содержащий информацию о геокодировании. Этот объект хранится в столбце *EmployeeList["EmployeeAddressGC"]*. Извлечем из него нужные нам данные.

5. Теперь пришло время создать столбец *lat_EmployeeAddress* путем извлечения информации о широте из колонки *EmployeeAddressGC*, созданной на предыдущем шаге. Как мы уже упоминали ранее, в случае успешного геокодирования адреса Google создаст объект *location.Location* с данными об адресе, широте и долготе. Нам нужны два последних параметра. Нужную нам информацию можно извлечь из объекта *location.Location* путем применения к нему *лямбда-функции* (*lambda*) к столбцу *EmployeeAddressGC*.

Примечание. Лямбда-функция представляет собой анонимную функцию, состоящую из одного выражения. Определяется такая функция при помощи ключевого слова *lambda*, следом за которым идут аргументы, а далее – выражение: *lambda arguments: <expression>*. При помощи такой конструкции можно легко применять функции к датафреймам в *pandas*.

Используйте следующий синтаксис лямбда-функции для извлечения широты из столбца *EmployeeAddressGC*: *lambda x: x.latitude*. Аргумент *x* в этой анонимной функции представляет столбец *EmployeeAddressGC*. А конструкция *x.latitude* позволяет обратиться к свойству *latitude* объекта *x*.

6. Создадим столбец *lon_EmployeeAddress*, воспользовавшись той же техникой, но заменив *latitude* на *longitude*.

7. Теперь повторим шаги 4, 5 и 6 применительно к полю *TerminalAddress*.
8. В заключение нам необходимо избавиться от ненужных столбцов в датафрейме. Колонки *EmployeeAddressGC* и *TerminalAddressGC* – исключительно технические, они нам были нужны лишь для извлечения информации о широте и долготе адресов проживания водителей и мест их работы. Теперь нам эти поля не нужны, можем от них избавиться. Легче всего можно сделать это, воспользовавшись методом *drop()* объекта *EmployeeList*. Для этого нужно выполнить следующий код:

```
cols = ["EmployeeAddressGC", "TerminalAddressGC"]
EmployeeList.drop(cols, inplace=True)
```

В переменной *cols* содержится список наименований столбцов для удаления. По умолчанию метод *drop()* будет выполняться применительно к копии датафрейма, а не к нему самому. Чтобы удалить столбцы непосредственно в датафрейме, необходимо присвоить аргументу *inplace* значение *True*.

В представленном примере вы научились обращаться к Google API из Python. При этом для геокодирования адресов нам пришлось написать совсем немного строк кода. Пользуясь встроенными средствами Power Query, нам бы не удалось решить эту задачу столь изящно. В следующем разделе вы научитесь вычислять расстояние в милях между двумя геокодированными точками с использованием пользовательской функции в R.

ВЫЧИСЛЕНИЕ РАССТОЯНИЯ МЕЖДУ ТОЧКАМИ С ПОМОЩЬЮ ПОЛЬЗОВАТЕЛЬСКОЙ ФУНКЦИИ В R

В предыдущих разделах мы настроили доступ к *Geocoding API* и геокодировали нужные нам адреса, тем самым получив всю необходимую информацию для вычисления расстояния между географическими точками. Ниже представлен скрипт на языке R для вычисления расстояния в милях между домашними адресами водителей и адресами их рабочих терминалов:

```
library(tidyverse)

ComputeDist <-
  function(pickup_long, pickup_lat, dropoff_long, dropoff_lat) {
    R <- 6371 / 1.609344
    delta_lat <- dropoff_lat - pickup_lat
    delta_long <- dropoff_long - pickup_long
    degrees_to_radians = pi / 180.0
    a1 <- sin(delta_lat / 2 * degrees_to_radians)
    a2 <- as.numeric(a1) ^ 2
    a3 <- cos(pickup_lat * degrees_to_radians)
    a4 <- cos(dropoff_lat * degrees_to_radians)
    a5 <- sin(delta_long / 2 * degrees_to_radians)
    a6 <- as.numeric(a5) ^ 2
```

```

    a <- a2 + a3 * a4 * a6
    c <- 2 * atan2(sqrt(a), sqrt(1 - a))
    d <- R * c
    return(d)
  }

setwd("<путь к файлу EmployeeList.csv>")
EmployeeList <- read_csv("EmployeeList.csv")

EmployeeList <-
  EmployeeList %>%
  mutate(
    `Custom Distance Function Results` =
      round(
        ComputeDist(
          lon_EmployeeAddress,
          lat_EmployeeAddress,
          lon_TerminalAddress,
          lat_TerminalAddress
        ), 1
      )
  )

```



Что происходит в этом коде, понять несложно. Распишем по шагам.

1. Сначала, как и всегда, нам необходимо импортировать нужные пакеты. И единственным пакетом, который нам понадобится, является известный вам *tidyverse*, поскольку весь остальной функционал содержится в самом R.
2. Теперь напишем пользовательскую функцию *ComputeDist()* для расчета расстояния между адресами. На вход функция будет принимать четыре параметра: широту и долготу места жительства сотрудника и его рабочего терминала. Обратите внимание на вычисления внутри функции. В Power Query такие расчеты произвести будет сложнее, да и по производительности это решение будет не сравнить с нашим. Я не буду подробно описывать процесс вычислений, поскольку эта тема выходит за рамки данной книги. Цель этого примера в том, чтобы показать, как легко и просто R справляется со сложными математическими расчетами.
3. После этого загрузим информацию из файла *EmployeeList.csv*.
4. Далее при помощи функции *mutate()* из пакета *dplyr* добавим новый столбец с именем *Custom Distance Function Results* к датафрейму *EmployeeList*. Такое имя функции было выбрано не случайно. Здесь важно подчеркнуть, что в данном случае мы выполняем все расчеты при помощи собственной пользовательской функции. В функцию передается четыре входных аргумента, а ее результат округляется до одного знака после запятой при помощи функции *round()*.
5. Последнее, что нам нужно сделать, – это скопировать скрипт и перенести его во встроенный редактор R в Power BI для загрузки в модель данных.


В данном упражнении мы создали вычисляемый столбец при помощи пользовательской функции в R всего за пять шагов. Прелесть использова-

ния пользовательских функций заключается в возможности заключить в них сложную логику вычислений и обращаться к ним каждый раз, когда необходимо произвести нужные вам расчеты. В результате код становится более эффективным и простым для восприятия.

ВЫЧИСЛЕНИЕ РАССТОЯНИЯ МЕЖДУ ТОЧКАМИ С ПОМОЩЬЮ ПОЛЬЗОВАТЕЛЬСКОЙ ФУНКЦИИ В PYTHON

В данном разделе мы выполним те же действия по вычислению расстояния между географическими объектами, но уже средствами языка Python.

Посмотрите на листинг, приведенный ниже:



```
import pandas as pd
import numpy as np
import os
from math import cos, sin, atan2, pi, sqrt, pow

def ComputeDist(row):
    R = 6371 / 1.609344 #radius in mile
    delta_lat = row["lat_TerminalAddressGC"] -
        row["lat_EmployeeAddress"]
    delta_lon = row["lon_TerminalAddress"] -
        row["lon_EmployeeAddress"]
    degrees_to_radians = pi / 180.0
    a1 = sin(delta_lat / 2 * degrees_to_radians)
    a2 = pow(a1,2)
    a3 = cos(row["lat_EmployeeAddress"] * degrees_to_radians)
    a4 = cos(row["lat_TerminalAddress"] * degrees_to_radians)
    a5 = sin(delta_lon / 2 * degrees_to_radians)
    a6 = pow(a5,2)
    a = a2 + a3 * a4 * a6
    c = 2 * atan2(sqrt(a), sqrt(1 - a))
    d = R * c
    return d

os.chdir("<path to folder where the EmployeeList file is>")
EmployeeList = pd.read_csv("EmployeeList_Python.csv")

EmployeeList["Custom Function"] = \
    EmployeeList.apply(lambda row: ComputeDist(row), axis=1)
```

Давайте распишем логику приведенного фрагмента кода по шагам.

1. Начинаем, как обычно, с импорта необходимых библиотек. Из новинок здесь, пожалуй, отметим несколько математических функций из пакета *math*. Эти функции мы используем при расчете расстояния между точками.

2. Теперь создадим функцию *ComputeDist()*, которая и будет выполнять всю работу по вычислению дистанции между двумя географическими объектами. На вход, в отличие от реализации в R, функция будет принимать единственный параметр в виде целой строки из датафрейма. Извлечение нужной информации из этой строки будет производиться уже внутри функции. Само математическое выражение для расчета расстояния между точками является довольно сложным, и в Power Query будет очень трудно добиться схожей лаконичности и ясности кода, как в Python.
3. Загрузим данные из файла *EmployeeList.csv*.
4. После этого воспользуемся методом *apply()* для применения лямбда-функции к нашему датафрейму *EmployeeList*. Эта строка кода выглядит так:

```
EmployeeList["Custom Function"] = (
    EmployeeList.apply(lambda row: ComputeDist(row), axis=1))
```

Примечание. Использование скобок в предыдущей строке кода позволило нам продолжить выражение на следующей строке, что сделало код более легким для восприятия. Распределение кода по строкам в Python должно в точности отвечать правилам, сформулированным в руководстве по написанию кода на Python PEP8. В данном руководстве, с которым можно ознакомиться по адресу <https://realpython.com/python-pep8>, прописаны все правила форматирования кода на Python.

Ранее в этой главе вы уже встречались с лямбда-функциями. Тогда мы применяли анонимную функцию к конкретному столбцу в датафрейме. Здесь же функция будет применяться ко всему датафрейму в целом. При этом вы можете указать *pandas*, к какой оси желаете применить функцию. Если в качестве аргумента *axis* передать значение 0, лямбда-функция будет применена сверху вниз, а если 1 – слева направо. Давайте рассмотрим на примере, что имеется в виду.

Создадим простой датафрейм с помощью показанного ниже кода для демонстрации поведения лямбда-функций:

```
df = pd.DataFrame((
    {'A':[10, 20, 30], 'B':[15, 5, 10]}))
```

Выполнение этого кода приведет к созданию датафрейма, показанного в табл. 8.1.

Таблица 8.1. Датафрейм для проверки работы лямбда-функций

	A	B
0	10	15
1	20	5
2	30	10

Если вам необходимо просуммировать значения по колонкам, вы должны двигаться по оси 0, то есть сверху вниз. Сделать это можно при помощи следующего кода:

```
df.apply(sum, axis = 0)
```

В результате вы получите данные, показанные в табл. 8.2.

Таблица 8.2. Суммирование по столбцам

A	60
B	30

Установка аргумента *axis* в 0 приводит к выполнению агрегации сверху вниз по каждому столбцу, что позволяет просуммировать значения в каждой отдельной колонке. Если в качестве аргумента *axis* передать 1, суммирование будет выполняться слева направо, а значит, мы получим агрегацию по строкам, как показано в приведенном ниже коде и табл. 8.3:

```
df.apply(sum, axis = 1)
```

Таблица 8.3. Суммирование по строкам

0	25
1	25
2	40

Теперь вы понимаете, как работают лямбда-функции применительно к целому датафрейму. Давайте освежим в памяти код, который выполняется на данном шаге:

```
EmployeeList["Custom Function"] = (
    EmployeeList.apply(lambda row: ComputeDist(row),
        axis=1))
```

Здесь применение лямбда-функции выполняется по оси 1. Иными словами, функция вызывается по строкам слева направо. В функцию *ComputeDist()* передается единственный аргумент в виде целой строки из датафрейма. Внутри функции *ComputeDist()* происходит извлечение нужной для расчета расстояния между точками информации из строки. Обратите внимание, что вы не ограничены вызовом агрегирующих функций при вызове метода *apply()* применительно ко всему датафрейму.

5. Скопируйте получившийся скрипт во встроенный редактор Python в Power BI и загрузите данные в модель.

Как и в случае с R, вы научились создавать вычисляемые столбцы при помощи Python за пять простых шагов. Итоговый код получился довольно простым для восприятия и весьма эффективным.

ВЫЧИСЛЕНИЕ РАССТОЯНИЯ МЕЖДУ ТОЧКАМИ С ПОМОЩЬЮ ГОТОВОЙ ФУНКЦИИ В R

В предыдущих двух разделах вы научились выполнять сложные математические вычисления при помощи пользовательских функций в R и Python. И хотя написать такие функции было не так уж трудно, зачастую и этого не требуется, поскольку есть более простые решения. На момент написания книги язык программирования R насчитывал более 15 тысяч пакетов, а количество библиотек Python перевалило уже за 110 тысяч. Понятно, что в таком разнообразии сторонних модулей нередко можно найти уже готовое решение своего сценария.

Это напрямую относится к нашей задаче вычисления расстояния между двумя географическими объектами. Вместо написания сложных математических формул в R и Python в данном случае достаточно воспользоваться готовыми функциями. В данном разделе вы узнаете, как можно применить функцию из пакета *geosphere* в R для расчета расстояния между точками. Полный скрипт представлен ниже:

```
library(tidyverse)
library(geosphere)

setwd("<путь к файлу EmployeeList.csv>")
EmployeeList <- read_csv("EmployeeList.csv")

EmployeeList <-
  EmployeeList %>%
  rowwise() %>%
  mutate(
    `Geosphere Function Result` =
      distHaversine(
        c(lon_EmployeeAddress, lat_EmployeeAddress),
        c(lon_TerminalAddress, lat_TerminalAddress)
      ) / 1609.34
  )
```

Давайте посмотрим, что происходит в этом фрагменте кода.

1. Сначала подключаем нужные нам пакеты. Из новинок для нас здесь пакет *geosphere*. Мы воспользуемся этой библиотекой для вычисления расстояния между точками.
2. Далее загружаем данные из файла *EmployeeList.csv* в датафрейм *EmployeeList*.
3. После этого добавляем новый столбец *Geosphere Function* в датафрейм *EmployeeList*. Здесь мы использовали конвейерный метод для добавления столбца и вычисления в нем расстояния между нужными нам точками при помощи функции *distHaversine()*. На первом шаге конвейера мы говорим пакету *dplyr*, что начинаем с нашего исходного датафрейма. Далее наш датафрейм передается на вход функции *rowwise()*,

необходимой по причине того, что функция *distHaversine()*, которую мы будем использовать далее, не является векторизованной.

Примечание. Векторизованная функция (vectorized function) работает с целым вектором одновременно. Столбцы в датафрейме представляют собой векторы. Применяя векторизованную функцию к вектору, вы не обязаны реализовывать внутренние циклы.

Функция *rowwise()* предписывает всем последующим операциям в сценарии обрабатывать в вычислениях только элементы текущей строки. На заключительном шаге мы воспользовались функцией *distHaversine()* из пакета *geosphere*. Функция *distHaversine()* возвращает расстояние между точками на карте в метрах, а умножение его на 0,000621371 позволяет произвести пересчет в мили.

4. Как и всегда, вам необходимо скопировать получившийся скрипт во встроенный редактор R в Power BI, чтобы была возможность загрузить информацию в модель данных.

В этом разделе вы научились выполнять сложные математические расчеты на этапе подготовки данных к загрузке в Power BI при помощи готовых функций. На этот раз в своих вычислениях мы воспользовались функцией *distHaversine()* из пакета *geosphere*. Все, что вам оставалось сделать, – это снабдить функцию необходимыми аргументами, а все вычисления она сделала за вас. В следующем разделе мы сделаем все то же самое с использованием языка Python.

ВЫЧИСЛЕНИЕ РАССТОЯНИЯ МЕЖДУ ТОЧКАМИ С ПОМОЩЬЮ ГОТОВОЙ ФУНКЦИИ В PYTHON

Приведенный ниже скрипт на языке Python выполняет расчет расстояния между точками на карте:

```
import pandas as pd
import os
from haversine import haversine

os.chdir(r"<путь к файлу EmployeeList.csv>")
EmployeeList = pd.read_csv("EmployeeList.csv")

def useHaversine(row):
    point_one = \
        (row["lat_EmployeeAddress"], row["lon_EmployeeAddress"])
    point_two = \
        (row["lat_TerminalAddress"], row["lon_TerminalAddress"])
    return haversine(point_one, point_two, unit="mi")

EmployeeList["Haversine Function Result"] = \
    EmployeeList.apply(lambda row: useHaversine(row), axis=1)
```

Давайте посмотрим, что здесь происходит.

1. Сначала, как и всегда, мы импортируем необходимые библиотеки. Здесь новой для нас библиотекой будет *haversine*. Именно с ее помощью мы будем производить пространственные вычисления.
2. Загружаем данные из файла *EmployeeList.csv* в датафрейм *EmployeeList*.
3. Теперь напишем собственную функцию *useHaversine()*. На вход эта функция будет принимать целую строку, и внутри будет извлекаться информация из нужных нам столбцов, как мы уже показывали ранее. Внутри нашей пользовательской функции *useHaversine()* мы будем вызывать готовую функцию *haversine()* из одноименного пакета для вычисления расстояния между точками. На вход функция *haversine()* принимает три параметра: координаты первой и второй точки на карте, а также единицу измерения, в которой необходимо получить результат. При этом первые два параметра должны представлять собой кортежи, в которых первым элементом идет широта, а вторым – долгота. Причина, по которой мы обернули готовую функцию *haversine()* в собственную пользовательскую функцию *useHaversine()*, заключается в том, что функция *haversine()* не является векторизованной. Если бы она была векторизованной, она могла бы оперировать со всеми строками из датафрейма параллельно. Но она умеет работать только с одной строкой за раз. Именно поэтому мы передаем в функцию *useHaversine()* только одну строку.
4. На заключительном шаге мы создаем вычисляемый столбец в датафрейме с именем *Haversine Function Result*, который будет содержать расстояние между домом и работой конкретного сотрудника в милях. Вычисление производится следующим образом:

```
EmployeeList["Haversine Function Result"] = EmployeeList.apply(lambda row:
useHaversine(row), axis=1)
```

Здесь мы применяем к датафрейму *EmployeeList* лямбда-функцию по оси 1. Как вы уже знаете из предыдущих разделов, это позволяет передать в функцию *useHaversine()* целую строку.

5. В завершение скопируйте весь скрипт во встроенный редактор Python в Power BI, чтобы можно было загрузить данные в модель.

В данном примере вы научились пользоваться готовыми функциями из библиотек Python для произведения нужных вам вычислений на этапе подготовки данных к загрузке в инструмент бизнес-аналитики. В нашем случае мы воспользовались функцией *haversine()* для расчета расстояния между двумя точками на карте. Как и в примере с R, здесь готовая функция выполнила все вычисления за вас. Все, что вам оставалось сделать, – это вызвать ее в нужный момент и с нужными параметрами.

ЗАКЛЮЧЕНИЕ

В этой главе мы продемонстрировали две важные техники. Сначала мы использовали R и Python для создания вычисляемых столбцов на основании сложных математических выражений, в результате чего убедились, что с применением этих языков программирования решения получаются более изящными и эффективными по сравнению со встроенным функционалом инструмента Power Query. При сравнении R и Python с DAX первенство в отношении эффективности также в большинстве случаев будет на стороне традиционных языков программирования. Вторая техника, которую мы использовали, связана с применением готовых функций из великого множества пакетов и библиотек, написанных за последние годы для R и Python. Мы в своем примере воспользовались формулой гаверсина для вычисления расстояния между географическими объектами по их координатам. Но в 15 тысячах пакетов для R и 110 тысячах библиотек для Python наверняка найдется подходящий функционал для большинства задач, которые будут перед вами стоять.

Возможность использования R и Python в Power BI для создания вычисляемых столбцов поможет вам преодолеть многие ограничения, характерные для Power Query. Помимо выполнения сложных вычислений, языки R и Python могут оказаться полезными для:

- создания вычисляемых столбцов с заменой отсутствующих значений с использованием гораздо более точных алгоритмов по сравнению с применением средних значений или моды;
- написания формул вычисляемых столбцов, базирующихся на сложных манипуляциях со строками, как было показано в главе 7;
- создания вычисляемых столбцов на базе алгоритмов машинного обучения, о чем мы поговорим в главе 9;
- выполнения множества других вычислений.

Возможности R и Python применительно к Power BI поистине безграничны!



Часть **IV**



.....

**МАШИННОЕ
ОБУЧЕНИЕ
И ИСКУССТВЕННЫЙ
ИНТЕЛЛЕКТ
В POWER BI
ПРИ ПОМОЩИ R
И PYTHON**





Применение методов машинного обучения и искусственного интеллекта к моделям данных Power BI

В последние годы для многих организаций стало обычным делом создание развитого решения в области бизнес-аналитики в рамках предприятия. Некоторые компании проектируют собственные *корпоративные хранилища данных* (Enterprise Data Warehouses – EDW), обслуживаемые при помощи современных и сложных процессов ETL. Кроме того, в организациях повсеместно используются инструменты построения отчетности, подобные SQL Server Report Services (SSRS) и Power BI для обеспечения руководства ценными аналитическими сведениями о состоянии дел. Компании, активно использующие BI-системы, обычно нацелены на расширение своего стратегического арсенала за счет привлечения решений, базирующихся на *искусственном интеллекте* (artificial intelligence – AI). Вот лишь несколько способов использования инструментов искусственного интеллекта для повышения эффективности системы бизнес-аналитики:

- магазин, торгующий мотоциклами, может использовать *модель логистической регрессии* (logistic regression model) для расширения справочника клиентов за счет прогнозной информации о том, какие из покупателей с большой долей вероятности приобретут новый мотоцикл в течение следующего года;
- розничная компания может воспользоваться методом *кластеризации k-средних* (k-means clustering) для выполнения маркетинговой сегментации клиентской базы;
- онлайн-продавец может использовать модель логистической регрессии для определения списка покупателей, которые с большой вероятностью могут отказаться от его услуг;

- ресторан может применить алгоритм *анализа тональности текста* (sentiment analysis) для расширения справочника посетителей путем добавления информации об эмоциональном настрое клиента на основании его отзыва о заведении.

Такие атрибуты, созданные при помощи алгоритмов искусственного интеллекта, могут иметь большую ценность для компании. В то же время рассчитать их при помощи стандартных средств бывает не так просто. Будучи загруженными в модель данных, эти атрибуты могут быть использованы так же точно, как и любые другие, – их можно отображать в визуальных элементах, таблицах и матрицах в Power BI.

В данной главе мы поговорим о том, как можно воспользоваться средствами искусственного интеллекта в моделях данных Power BI. Здесь мы затронем вопросы использования простых методов, а позже в этой книге вы узнаете, как можно масштабировать внедренные модели ИИ без необходимости приобретать дорогостоящую версию Power BI Premium.

Примечание. В состав пакета Power BI Premium входит инструмент *AI Insights*, облегчающий применение методов искусственного интеллекта к вашим моделям данных Power BI. Здесь мы не будем рассматривать этот инструмент, поскольку он не является бесплатным. Все примеры, которые мы будем обсуждать в книге, могут быть реализованы без необходимости покупать версию Premium.

В этой главе вы познакомитесь сразу с несколькими способами применения алгоритмов искусственного интеллекта в Power BI. В частности, вы узнаете:

- как проводить оценку данных в Power BI с использованием пользовательских *моделей машинного обучения* (machine learning model), встроенных в R и Python;
- как использовать службу *Microsoft Cognitive Services* для проведения *анализа тональности текста* (sentiment analysis) в вашей модели данных в Power BI;
- как воспользоваться службой *IBM Watson Natural Language Understanding* для анализа настроения текста в вашей модели данных Power BI.

В следующей главе мы посмотрим, как можно применить описанные здесь техники для масштабирования решений до уровня предприятия. Давайте начнем с задачи оценки данных с использованием пользовательских моделей в R и Python.

ПРИМЕНЕНИЕ АЛГОРИТМОВ МАШИННОГО ОБУЧЕНИЯ К НАБОРУ ДАННЫХ ПЕРЕД ЗАГРУЗКОЙ В МОДЕЛЬ Power BI

Вот вам новый сценарий. Вы работаете в компании, занимающейся аналитикой в области недвижимости в Бостоне, штат Массачусетс. Вы поставили своему аналитику данных задачу спрогнозировать средние цены на дома

в Бостоне, в результате чего он разработал и сохранил для вас в общей директории модели машинного обучения на R и Python, основываясь на информации о недвижимости из наборов данных Kaggle. Вы будете использовать эти модели для оценки актуальных данных, содержащих список домов, по которым вам необходимо спрогнозировать цены.

В вашем рабочем наборе данных содержится 14 переменных, а в тестовом, который разработал аналитик, всего четыре следующие:

- CRIM: уровень преступности в расчете на человека по городам;
- RM: среднее количество комнат в квартире;
- TAX: налог на недвижимость в расчете на \$10 000;
- LSTAT: процент населения, занимающего более низкое общественное положение.

Наша цель – написать скрипты на языках R и Python для Power BI, которые будут выполнять нужную нам оценку. Начнем, как всегда, с R.

Прогнозирование цен на недвижимость с помощью R



Применить модели машинного обучения, разработанные для вас аналитиком, к вашим моделям данных в Power BI не так сложно. Шаги, которые необходимо выполнить для этого, описаны ниже. Здесь мы не будем обсуждать нюансы построения полноценных актуальных моделей, поскольку это выходит за рамки данной книги. Вместо этого мы сосредоточимся на том, как применять созданную для вас модель машинного обучения к модели данных в Power BI.

Шаг 1. Пусть аналитик данных сохранит для вас модель

Модель машинного обучения в R представляет собой объект, который может быть сохранен в файл. После создания модели аналитик данных может сохранить ее на диск и передать вам так же, как любой другой файл. Процесс сериализации модели в R крайне прост и может быть выполнен при помощи всего одной строки кода:

```
saveRDS(model, "model.rds")
```



Здесь просто берется модель, созданная аналитиком и хранящаяся в переменной *model*, и записывается на диск в рабочую директорию в виде файла с расширением *.rds при помощи функции *saveRDS()*. Эта функция входит в базовую установку языка R.

Шаг 2. Загрузите пакет tidyverse

На этом шаге мы приступим к написанию скрипта. И начнем, как и всегда, с загрузки необходимых пакетов. В данном случае нам понадобится всего один хорошо знакомый нам пакет *tidyverse*, а все остальное будет выполнять-

ся при помощи базового функционала языка R. Метапакет *tidyverse* можно загрузить следующим образом:

```
library(tidyverse)
```

Шаг 3. Загрузите объект модели и набор данных для оценки

Здесь мы просто инициализируем переменные, загружая сохраненную для вас модель машинного обучения и актуальный набор данных:

```
model <- readRDS("./Model/model.rds")  
boston_housing <- read_csv("./Data/BostonHousingData.csv")
```

Функция *readRDS()* позволяет выполнить десериализацию модели, сохраненной в файле, для дальнейшего использования в скрипте. Также мы использовали функцию *read_csv()* из пакета *readr* для загрузки набора данных в датафрейм *boston_housing*.

Шаг 4. Ограничьте датафрейм столбцами, необходимыми для вашей модели

Нам нужно всего четыре столбца из загруженного набора данных, и мы выберем их при помощи следующего кода:

```
model_data <- boston_housing[,c("crim", "rm", "tax", "lstat")]
```

Пропущенный первый индекс указывает на то, что мы хотим оставить все строки из исходного датафрейма. В символьном векторе, следующем в качестве второго индекса, перечислены столбцы, которые мы хотим оставить в итоговом датафрейме. В результате мы получаем датафрейм с именем *model_data*, в котором собраны все строки из исходного датафрейма и столбцы с именами, перечисленными в символьном векторе.

Шаг 5. Примените модель машинного обучения к своему набору данных для составления прогноза цен на недвижимость

Это можно сделать следующим образом:

```
pred_medv <- predict(model, model_data)
```

В результате запуска этой строки кода мы получим вектор, содержащий средние прогнозные цены на недвижимость для каждого исследования, переданного на вход.

Примечание. Формально в обычной жизни определение исследования относится к записи о количестве и качестве исследуемого феномена. В нашем примере мы исследуем информацию о недвижимости, по которой хотим получить прогнозируемые цены.

Обратите внимание, что датафрейм, передаваемый на вход функции *predict()*, должен содержать переменные (столбцы), требуемые для нашей модели.

Шаг 6. Добавьте прогноз к исходному набору данных

Код из предыдущего шага вернет нам лишь вектор данных, ничего не означающий в отрыве от остальной информации. Теперь нам необходимо объединить исходный датафрейм с полученными данными. Сделать это можно следующим образом:

```
final_output <- cbind(model_data, pred_medv)
```

В коде используется функция *cbind()* для присоединения полученного столбца справа к исходному датафрейму *model_data*. Итоговый датафрейм сохраняется в переменной *final_output*. Буква «с» в названии функции *cbind()* означает столбцы (column). Таким образом, в данном случае вектор *pred_medv* присоединяется к датафрейму *model_data* в качестве нового столбца, который получит имя вектора – *pred_medv*.

Шаг 7. Скопируйте скрипт в Power BI

Итоговый скрипт показан ниже:

```
library(tidyverse)

model <- readRDS("./Models/model.rds")
boston_housing <- read_csv("./Data/BostonHousingInfo.csv")

model_data <- boston_housing[, c("crim", "rm", "tax", "lstat")]
pred_medv <- predict(model, model_data)

final_output <- cbind(model_data, pred_medv)
```

Как и раньше, для загрузки скрипта в Power BI необходимо воспользоваться инструментом получения данных (GetData). Вставьте получившийся скрипт во встроенный редактор R, после чего нажмите на кнопку **ОК**. Датафрейм *final_output* будет виден в Power BI как любой другой набор данных, и его можно при необходимости загрузить в модель, создать меры, использующие прогнозные цифры, или построить визуализацию на основе проведенной оценки.

Прогнозирование цен на недвижимость с помощью Python

В данном разделе мы сделаем все то же, что и в предыдущем, но с использованием языка Python. Как и в случае с R, мы сосредоточимся на процессе применения созданной для вас модели машинного обучения к вашей модели данных, поскольку нюансы построения самой модели машинного обучения

выходят за рамки этой книги. Ниже приведены шаги решения поставленной задачи применительно к языку Python.



Шаг 1. Пусть аналитик данных сохранит для вас модель

Как и в R, в Python модель представляет собой объект, который может быть сохранен на диске и передан вам, как и любой другой файл. Процесс сериализации модели в Python не сложнее, чем в R, и может быть выполнен всего в одной строке кода, не считая загрузки библиотеки *joblib*:

```
import joblib
joblib.dump(model, "model")
```

Здесь мы берем модель, созданную аналитиком для вас, и сохраняем ее в файл в текущей директории при помощи функции *joblib.dump()* из библиотеки *joblib*. После этого созданный файл может быть передан вам любым способом, включая размещение на *OneDrive* или *Dropbox*. В более сложных системах может быть задействована концепция *MLOps*, которая позволит делиться готовыми моделями машинного обучения посредством инструментов *GitHub* или *Azure DevOps*.

Шаг 2. Загрузите необходимые библиотеки

На этом шаге мы приступим к написанию скрипта для применения модели машинного обучения к нашей модели данных в Power BI. Для выполнения задачи нам понадобятся следующие модули и пакеты:

```
import os
import joblib
from sklearn import preprocessing
import pandas as pd
```

Библиотека *joblib* содержит в себе средства для десериализации модели, которую аналитик сохранил в виде файла для вас. Модуль *preprocessing* из пакета *scikit-learn* пригодится для выполнения преобразований в наборе данных при подготовке к анализу, а библиотека *pandas* позволит манипулировать датафреймами.



Шаг 3. Загрузите объект модели и актуальный набор данных

Это можно сделать при помощи следующих двух строк кода, при условии что модель машинного обучения хранится в файле с именем *model_Python.pkl*:

```
model = joblib.load("../Models/model_Python.pkl")
boston_housing = pd.read_csv("../Data/BostonHousingInfo.csv")
```

Функция *joblib.load()* очень похожа на *readRDS()*, которую мы использовали в R. Она позволит вам десериализовать модель, сохраненную для вас на диске, чтобы можно было в дальнейшем использовать ее в скрипте. Далее

мы воспользуемся методом `read_csv()` из библиотеки *pandas* для чтения и сохранения нужного нам набора данных в датафрейме. Обратите внимание, что в обоих случаях в пути к файлу используется префикс в виде двух точек. Это позволяет вам начать адресацию из папки, находящейся на один уровень выше вашей текущей рабочей директории. В нашем случае папки *Models* и *Data* располагаются на одном уровне с папкой *Python*, которая установлена в качестве рабочей директории. Использование префикса с двумя точками приведет к подъему на уровень папки *Chapter09*, в которой и находятся нужные нам директории *Models* и *Data*.

Шаг 4. Извлеките нужную информацию из датафрейма

В датафрейме, который мы загрузили, содержится 13 переменных, тогда как нам нужны лишь четыре из них. После извлечения нужных нам столбцов из датафрейма *boston_housing* необходимо выполнить так называемую стандартизацию данных.

Примечание. Операцию *стандартизации* (standardization) зачастую приходится выполнять применительно к моделям в библиотеке *scikit-learn*. Эта операция рекомендована при работе с переменными, характеризующимися разным масштабом, как в нашем примере. В этом случае переменные с большим масштабом могут обладать и большим весом, что приведет к смещению всей модели в целом. Функция *scale()* из модуля *preprocessing* помещает все переменные в один масштаб, что позволяет избавиться от упомянутой проблемы. По следующей ссылке на YouTube можно посмотреть видео с подробным описанием операции стандартизации: www.youtube.com/watch?v=sh_tLn1phfc.

Оставить только требуемые колонки в датафрейме и выполнить операцию стандартизации можно при помощи следующей строки кода:

```
model_data = preprocessing.scale(boston_housing[["crim", "rm", "tax", "lstat"]])
```

Давайте внимательно посмотрим на то, что мы передаем в виде аргумента в функцию *scale()*, а затем расскажем, что делает эта функция. Итак, на вход функции *scale()* поступает следующее выражение:

```
boston_housing[["crim", "rm", "tax", "lstat"]]
```

Здесь мы выбираем из датафрейма *boston_housing* только нужные нам столбцы, перечисленные в виде списка в квадратных скобках. После этого полученный датафрейм передается в функцию *scale()*, где осуществляется его стандартизация. В результате в переменной *model_data* окажется датафрейм с переменными, приведенными к единому масштабу.

Шаг 5. Примените модель к подготовленному набору данных для расчета прогноза

Это можно сделать, выполнив следующий код:

```
pred_medv = model.predict(model_data)
```

В результате выполнения этой строки будет получен массив с прогнозными значениями по средним ценам на жилье в Бостоне для каждого переданного на вход исследования. Этот массив будет сохранен в переменной с именем *pred_medv*. Обратите внимание, что в датафрейме *model_data* находятся те данные, которые требуются нашей модели, сохраненной в переменной *model*.

Шаг 6. Добавьте прогнозные данные к исходному набору данных

На предыдущем шаге мы получили массив данных, который, как и в случае с R, необходимо объединить с исходным набором. Это несложно сделать при помощи следующего кода:

```
final_output = boston_housing.loc[:,["crim","rm","tax","lstat"]]
final_output["pred_medv"] = pred_medv
```

Назовем набор данных, который в результате передадим в Power BI, *final_output*. Для начала поместим в него только те столбцы из датафрейма *boston_housing*, которые нам нужны, а именно *crim*, *rm*, *tax* и *lstat*.

Для этого мы воспользуемся методом *loc* для ограничения датафрейма *boston_housing* необходимыми нам столбцами. Вообще, этот метод позволяет одновременно выбрать как определенные колонки, так и строки. Первый аргумент отвечает за строки, а второй – за столбцы. В нашем примере мы бы хотели оставить в датафрейме все без исключения строки, поэтому первым аргументом передаем знак двоеточия. Если бы мы хотели выбрать первые три строки, достаточно было бы написать *0:2*. Слева от двоеточия стоит индекс первой строки для выбора, а справа – последней, исключая ее саму. Напомним, что в Python индексация начинается с нуля. Следующим аргументом дадим *pandas* понять, какие столбцы хотим вернуть. Можно также использовать знак двоеточия, если вам нужны все столбцы в исходном датафрейме. Если же вам необходимо оставить в датафрейме определенные колонки, можете перечислить их имена в списке, как мы сделали в этом примере. После этого добавим новый столбец *pred_medv* к итоговому датафрейму *final_output* с использованием следующей строки кода: *final_output["pred_medv"] = pred_medv*.

Шаг 7. Скопируйте скрипт в Power BI

Полный скрипт на Python приведен ниже:

```
import os
import pandas as pd
import joblib
from sklearn import preprocessing

model = joblib.load("../Models/model_Python.pkl")
boston_housing = pd.read_csv("../Data/BostonHousingInfo.csv")

model_data = preprocessing.scale(boston_housing[["crim","rm","tax","lstat"]])
```

```
pred_medv = model.predict(model_data)

final_output = boston_housing.loc[:,["crim","rm","tax","lstat"]]
final_output["pred_medv"] = pred_medv
```

Как и в предыдущих примерах, загрузите скрипт в Power BI при помощи инструмента получения данных (GetData). Вставьте фрагмент кода во встроенный редактор Python и нажмите на кнопку **ОК**. Датафрейм *final_output* станет доступен в Power BI, как любой другой набор данных, и может быть загружен в модель.

При необходимости вы можете выполнить дополнительные преобразования в Power Query перед загрузкой данных в модель Power BI. Например, вы бы могли разбить цены на недвижимость на диапазоны. Допустим, дома с прогнозной стоимостью меньше 100K были бы отнесены к категории «<100K», остальные – к своим категориям: «100–200K», «200–300K» и т. д. Подобные преобразования могут позволить сделать ценные выводы о ситуации в компании.

ИСПОЛЬЗОВАНИЕ ГОТОВЫХ МОДЕЛЕЙ ИИ ДЛЯ РАСШИРЕНИЯ ФУНКЦИОНАЛА МОДЕЛЕЙ ДАННЫХ В POWER BI

Разработка моделей искусственного интеллекта – задача не из легких. На это может потребоваться немало времени и ресурсов с привлечением сильной команды специалистов в области науки о данных. К счастью, крупные производители программного обеспечения, такие как *Microsoft* и *IBM*, позаботились о том, чтобы разработать за вас модели ИИ, доступ к которым легко можно получить при помощи вызовов API. В основе этих моделей лежит многолетний опыт работы, и сделать что-то подобное своими силами будет очень и очень непросто.

В данном разделе мы покажем, как можно за счет готовых моделей искусственного интеллекта расширить функционал собственных моделей данных в Power BI. В частности, вы узнаете, как можно воспользоваться службой *Microsoft Cognitive Services* для выполнения анализа тональности текста.

Примечание. Анализ тональности текста (sentiment analysis) применяется для определения того, насколько позитивным или негативным является фрагмент текста, будь то предложение, абзац или сообщение в Твиттере. В *Microsoft Cognitive Services* оценке тексту дается по шкале от 0 до 1 – чем ближе к нулю, тем негативнее оттенок текста, и чем ближе к единице, тем он позитивнее.

Вы можете использовать анализ тональности текста для обогащения своей модели данных Power BI путем добавления соответствующих полей в таблицы измерений. К примеру, если речь идет о ресторане, можно дополнить

таблицу посетителей столбцом с указанием качества его отзывов в среднем. В дальнейшем эту информацию можно будет сопоставить с динамикой оттока клиентов для выявления ценных инсайтов. В варианте с рестораном вы также можете, к примеру, отследить закономерности в зависимости между качеством отзывов и заказываемыми блюдами, что поможет улучшить меню заведения.

Давайте посмотрим, что нужно сделать, чтобы добавить анализ тональности текста к модели данных Power BI с использованием *Microsoft Cognitive Services*. Этот пример будет показан только для языка Python, поскольку язык R в этой связке пока не поддерживается. Можно эту задачу также решить при помощи *AI Insights* через *Power BI Dataflows*, но для этого вам понадобится дорогостоящая версия Power BI Premium, а у большинства пользователей и разработчиков Power BI нет возможности ее приобрести.

Настройка Cognitive Services в Azure

Перед тем как воспользоваться службой *Microsoft Cognitive Services* в Power BI, вы должны настроить ее в Azure. Если вы выполнили все требования, указанные во вводной части к данной книге, у вас уже должен быть свой аккаунт в Azure. В противном случае я очень рекомендую вам создать свой бесплатный аккаунт в этой облачной платформе. На момент написания книги компания *Microsoft* предлагала всем новичкам кредит в размере \$200 на первый месяц. Этого будет вполне достаточно, чтобы выполнить все упражнения из данной книги. Если вы еще не зарегистрированы в Azure, обратитесь к введению в книгу, там есть все необходимые инструкции.

Итак, для продолжения вам нужно войти на портал Azure по ссылке <https://portal.azure.com>. Затем в выпадающем меню слева сверху выберите пункт **Create a resource**. На открывшейся странице в списке **Categories** слева выберите вариант **AI + Machine Learning** и на странице справа щелкните на пункт **Text Analytics**. Далее вам необходимо будет заполнить форму для настройки службы *Text Analytics*. В поле **Name** введите осмысленное имя ресурса, присоедините подписку в поле **Subscription**, в поле **Location** укажите ближайшее к вам местоположение, выберите подходящий вам тариф в поле **Pricing tier** и присоедините ресурс к уже имеющейся группе ресурсов или создайте новую группу, заполнив поле **Resource group**. Если вашему аккаунту не более месяца, вам будет доступна подписка с кредитом на \$200.

Виртуальная машина для анализа данных (Data Science Virtual Machine – DSVM)

Как мы уже сказали, представленная в данном разделе задача будет решена только при помощи Python, поскольку язык R в этой связке пока не поддерживается. Кроме того, реализация будет выполнена в рамках виртуальной машины для анализа данных (Data Science Virtual Machine – DSVM). Причина

такого выбора заключается в том, что виртуальная машина заранее сконфигурирована и содержит большинство инструментов, которые понадобятся для решения подобных сценариев. В частности, в ней установлены дистрибутив *Anaconda* для Python, *Power BI*, *SQL Server*, *Jupyter Lab*, *Azure Data Studio* и другие полезные сервисы. Настроить такое богатое окружение самостоятельно будет не так просто, а с помощью *DSVM* у вас все это уже есть в распоряжении.

Перед тем как продолжить работу, вам необходимо сделать еще кое-что в виртуальной машине, чтобы завершить настройку окружения. А именно вам нужно установить дополнительные пакеты для службы *Microsoft Cognitive Services*. Вот как это сделать.

1. Запустите *DSVM*. Вы можете заниматься разработкой на своем персональном компьютере, но я настоятельно рекомендую использовать виртуальную машину для анализа данных. Если вы решите пойти своим путем, начните с настройки окружения.
2. Запустите командную строку *conda*. Для этого откройте строку поиска, иконка которой расположена рядом со значком Windows, введите слово *Anaconda*, после чего в окне выбора появится вариант *Anaconda Prompt*. Щелкните по нему правой кнопкой мыши и запустите от имени администратора.
3. Активируйте окружение Python, которое вы будете использовать для разработки в *Azure*. Рекомендуется использовать окружение *AzureML*, предустановленное в *DSVM*. Для активации окружения *AzureML* введите следующую команду в командной строке *conda*:

```
conda activate AzureML
```

Если вы работаете в собственном окружении и не знаете, какое окружение лучше использовать, введите в командной строке *conda* команду *conda env list*, чтобы посмотреть список доступных окружений.

4. Введите показанную ниже команду для установки библиотеки. Убедитесь, что производите установку в правильном окружении, для чего предварительно активируйте нужное вам окружение, выполнив пункт 3:

```
pip install azure-cognitiveservices-language-textanalytics
```

Анализ тональности текста в Microsoft Cognitive Services при помощи Python

В данном примере мы будем использовать набор данных с отзывами с сайта *Yelp*, который можно скачать с *Kaggle*. Ниже описаны шаги, которые необходимо выполнить, чтобы получить набор данных и провести его анализ в *Power BI*.

Шаг 1. Загрузите набор данных с отзывами Yelp с сайта Kaggle

На момент написания книги нужный нам набор данных располагался по адресу <http://www.kaggle.com/yelp-dataset/yelp-dataset/version/4>. Убедитесь, что вы загружаете именно csv-версию набора данных *yelp_review*. Информация будет загружена в виде zip-архива. Разархивируйте файл *yelp_review.csv* и сохраните его в репозитории к данной главе в папке *Data*.

Файл *yelp_review.csv* довольно велик по объему – он содержит огромное количество отзывов, но нам в нашем задании нужны далеко не все они. В следующем фрагменте кода показан способ извлечения ста случайных отзывов из файла *yelp_review.csv* и сохранения их в новый файл *yelp_review_sample.csv*:

```
import pandas as pd
import numpy as np
import os

os.chdir("путь к рабочей директории")

dfYelpReviews = pd.read_csv("yelp_review.csv")
dfYelpReviewsSample = dfYelpReviews.sample(100, random_state = 1)
dfYelpReviewsSample["id"] = np.arange(len(dfYelpReviewsSample))
dfYelpReviewsSample = dfYelpReviewsSample[["id", "text"]]
dfYelpReviewsSample.to_csv("yelp_review_sample.csv", index = False)
```

Вот что происходит в этом коде.

1. Загружаем необходимые библиотеки. Пакет *pandas* мы используем для манипулирования с данными, функцию *arange()* из библиотеки *numpy* задействуем для создания столбца с порядковыми номерами строк, а пакет *os* нам понадобится для установки рабочей директории.
2. Считываем данные из файла *yelp_review.csv*. Для этого мы воспользовались методом *read_csv()* из библиотеки *pandas*.
3. Создаем на основании базового тестовый набор данных при помощи метода *sample()* и сохраняем его в датафрейм с именем *dfYelpReviewsSample*. Мы создали этот поднабор с использованием всего двух аргументов. Первым аргументом мы передали желаемый объем набора данных – в нашем случае это 100. Установив второй аргумент с именем *random_state* в единицу, вы при запуске этого фрагмента кода получите тот же набор данных, что и автор книги.

Шаг 2. Загрузите нужные библиотеки, модули и функции для скрипта

На этом шаге мы начинаем писать наш скрипт. И для начала импортируем необходимые пакеты и модули при помощи следующего фрагмента кода:

```
import pandas as pd
import ast
import os
```

```
from azure.cognitiveservices.language.textanalytics import TextAnalyticsClient
from msrest.authentication import CognitiveServicesCredentials
```

С библиотеками *pandas* и *os* вы уже знакомы, тогда как пакет *ast* встретился вам впервые. Мы будем использовать функцию *literal_eval()* из него для выполнения строки, представляющей выражение на языке Python. Мы объясним, что здесь имеется в виду, на следующем шаге. Функции *TextAnalyticsClient()* и *CognitiveServicesCredentials()* будут использованы для взаимодействия со службой *Microsoft Cognitive Services*.

Шаг 3. Инициализируйте переменные для работы скрипта

При помощи следующего фрагмента кода мы можем присвоить переменным нужные значения:

```
api_key = "<ключ API>"
endpoint = \
    "https://rwttextanalyticsapi.cognitiveservices.azure.com/"
credentials = CognitiveServicesCredentials(api_key)
text_analytics = TextAnalyticsClient(endpoint=endpoint, credentials=credentials)
```

Свой ключ API и ссылку для переменной *endpoint* вы можете найти в созданном вами ресурсе текстового анализа в *Azure* на странице *Overview*. Перейти на нее можно, щелкнув по соответствующей вкладке в правом верхнем углу. Ключ API мы используем для создания экземпляра класса *CognitiveServicesCredentials*. Переменные *credentials* и *endpoint* используются совместно для создания объекта *TextAnalyticsClient*, который мы сохранили в переменной *text_analytics*. Именно эта переменная и будет нашей главной рабочей лошадкой в деле анализа текста.


Шаг 4. Считайте фрагмент файла с отзывами в датафрейм

На этом шаге мы загрузим созданный нами ранее фрагмент набора данных с отзывами в датафрейм *pandas*. **Очень важно, чтобы вы использовали именно фрагмент набора данных, а не весь исходный файл!** В первоначальном наборе слишком много данных, чтобы их можно было одновременно обработать. Ста записей, которые мы оставили в нашем csv-файле, вполне достаточно для нашей задачи. Напомним, что в созданном нами наборе данных присутствует два столбца: в поле *id* находится идентификатор отзыва, а в поле *text* – его содержание. Добавим еще одну колонку с именем *language*, чтобы служба *Microsoft Cognitive Services* могла определить язык текста. Мы заполнили этот столбец значениями «en», поскольку наши отзывы написаны на английском языке.

```
df = pd.read_csv("<путь к файлу yelp_review_sample.csv>")
df["language"] = "en"
```

Шаг 5. Преобразуйте датафрейм к формату, приемлемому для службы *Microsoft Cognitive Services*

Служба *Microsoft Cognitive Services* требует, чтобы данные, поступающие на вход, были оформлены в виде списка словарей. Ниже показано, как примерно может выглядеть этот формат:



```
documents = [
    {"id": "1",
     "language": "en",
     "text": "Football is great sport."},
    {"id": "2",
     "language": "en",
     "text": "Lamar Jackson is the best quarterback."},
    {"id": "3",
     "language": "en",
     "text": "UK football sucks!!! "}
]
```

Метод `to_json()` из библиотеки *pandas* можно использовать для приведения датафрейма к требуемому формату. Для этого достаточно аргументу *orient* присвоить значение «records», как показано ниже:

```
documents = df.to_json(orient='records')
```

В результате мы получим необходимый нам формат данных в виде списка словарей, однако Python его так не воспринимает, а видит как строку. Нам необходимо явно указать Python, что строку `df.to_json(orient='records')` нужно воспринимать как выражение. Это можно сделать, воспользовавшись функцией `literal_eval()` из библиотеки *ast*, как показано ниже:

```
documents = ast.literal_eval(documents)
```

Теперь в переменной *documents* информация содержится в формате, приемлемом для службы *Microsoft Cognitive Services*. Каждый элемент списка при этом содержит словарь, состоящий ровно из трех пар ключ/значение. Ключ *id* уникально определяет отзывы в наборе данных, ключ *text* представляет текст самого отзыва, а ключ *language* служит для идентификации языка отзыва.

Шаг 6. Оцените отзывы посетителей при помощи метода `sentiment()`

В следующей строке кода производится оценка отзывов:

```
response = text_analytics.sentiment(documents=documents, raw = False)
```

В результате выполнения метода `sentiment()` мы получим список объектов *Batch result item*, каждый из которых содержит свойства, характеризующие отдельный отзыв. Нас интересуют два свойства: *id* и *score*. На следующем шаге мы используем их для построения датафрейма с оценками отзывов.

Шаг 7. Создайте датафрейм, содержащий оценки отзывов

Итак, мы произвели оценку отзывов из нашего набора данных и получили на выходе список объектов *Batch result item*. Теперь построим новый датафрейм на основе свойств *id* и *score* каждого объекта, возвращенного службой *Microsoft Cognitive Services*:

```
listSentiments = []
for document in response.documents:
    id = document.id
    score = document.score
    listSentiments.append([id, score])

dfSentiments = pd.DataFrame(listSentiments, columns=['ID', 'Score'])

dfSentiments
```

Давайте последовательно пройдемся по этому скрипту.

1. Сначала создаем пустой список с именем *listSentiments*. Этот список впоследствии будет заполнен выводом из службы *Microsoft Cognitive Services* и сконвертирован в датафрейм на следующем шаге.
2. Проходим в цикле по каждому документу в списке *response.documents*, получая на каждом шаге очередной отзыв.
3. Извлекаем свойства *id* и *score* из очередного элемента и помещаем их в соответствующие переменные следующим образом:

```
id = document.id
score = document.score
```

4. Добавляем значения из текущего элемента к списку *listSentiments*. Таким образом, мы постепенно заполняем список переменными, инициализированными на третьем шаге, при помощи метода *append()*.
5. Создаем датафрейм *dfSentiments*. Метод *DataFrame()* из библиотеки *pandas* используется для преобразования списка *listSentiments* в полноценный датафрейм. Список, содержащий имена нужных нам столбцов, передается в метод посредством аргумента *columns*.

Шаг 8. Скопируйте скрипт в Power BI

Полный скрипт на Python приведен ниже:

```
import pandas as pd
import json
import ast
from pandas.io.json import json_normalize
from azure.cognitiveservices.language.textanalytics import TextAnalyticsClient
from msrest.authentication import CognitiveServicesCredentials

api_key = "<ключ API>"
endpoint = \
    "https://rwttextanalyticsapi.cognitiveservices.azure.com/"
```

```

credentials = CognitiveServicesCredentials(api_key)
text_analytics = TextAnalyticsClient(
    endpoint=endpoint, credentials=credentials)

df = pd.read_csv("<путь к файлу yelp_review_sample.csv>")
df["language"] = "en"

documents = df.to_json(orient='records')
documents = ast.literal_eval(documents)
response = text_analytics.sentiment(
    documents=documents, raw = False)

listSentiments = []
for document in response.documents:
    id = document.id
    score = document.score
    listSentiments.append([id, score])

dfSentiments = pd.DataFrame(
    listSentiments, columns=['ID', 'Score'])

```

Скопируйте полный скрипт во встроенный редактор Python в Power BI. После этого вам будет доступен датафрейм *dfSentiments* в Power BI, как любой другой набор данных, и вы сможете загрузить его в модель данных.

ПРИМЕНЕНИЕ СТОРОННИХ МОДЕЛЕЙ МАШИННОГО ОБУЧЕНИЯ К МОДЕЛЯМ ДАННЫХ POWER BI

Служба *Microsoft Cognitive Services* насчитывает большое количество моделей искусственного интеллекта, которыми вы можете пользоваться при необходимости. Но их список, увы, не исчерпывающий. К счастью, другие производители программного обеспечения, такие как IBM и Google, не стоят на месте и также активно разрабатывают модели, которые могут вам пригодиться в работе. При ближайшем изучении вы можете обнаружить, что некоторые сервисы – к примеру, анализ тональности текста, – у разных производителей практически повторяются, но есть и уникальные инструменты. Одним из таких инструментов является *средство анализа настроения текста* (Tone Analyzer), входящее в состав службы *IBM Watson Natural Language Understanding*.

В данной главе мы посмотрим, как можно воспользоваться API этого инструмента в Power BI. Как ясно из названия, средство анализа настроения текста позволяет оценить эмоциональный окрас текстового послания. Служба умеет отличать веселые сообщения от грустных, а восторженные – от недовольных. Здесь, как и в предыдущем разделе, мы реализуем свою задумку исключительно с помощью языка Python, поскольку на момент написания книги в *IBM Watson* не было поддержки R. В следующем разделе мы подробно остановимся на том, как настроить аккаунт в службе *IBM Watson Natural Language Understanding*, после чего расскажем, как можно воспользоваться этим средством в Power BI при помощи скрипта на Python.

Конфигурирование средства анализа настроения текста в IBM Watson

Как и в случае с *Azure*, вам необходимо будет сначала завести аккаунт, а затем настроить *анализатор настроения текста* (Tone Analyzer). Ниже приведены шаги, которые необходимо выполнить.

Шаг 1. Заведите аккаунт в IBM Cloud

Откройте ссылку www.ibm.com/cloud/watson-natural-language-understanding/pricing и заведите аккаунт для доступа к службе *IBM Watson Natural Language Understanding*.

Шаг 2. Выполните вход в IBM Cloud

После того как вы зарегистрировались, перейдите по ссылке <https://cloud.ibm.com/login> для входа в систему.

Шаг 3. Перейдите на страницу Tone Analyzer

1. Нажмите на кнопку **Create Resource** в верхней части страницы. Откроется страница с выбором доступных вам служб.
2. Введите в поле поиска с надписью **Search the catalog** в правой верхней области страницы слова **Tone Analyzer**. При появлении нужной вам службы выберите ее.

Шаг 4. Настройте службу Tone Analyzer

1. В разделе **Select a region** выберите ближайший к вам регион.
2. Выберите план **Lite** в разделе **Select a pricing plan**. Этого плана будет вполне достаточно для выполнения задания из данной главы. Если в будущем вам потребуется выполнять более 2500 запросов к API или у вас появятся дополнительные требования к безопасности, вам может понадобиться перейти на другой план.
3. В разделе **Configure your resource** выполните следующие действия:
 - дайте своему плану осмысленное имя в секции **Service name**;
 - выберите группу ресурса. Для первого раза вы можете оставить группу по умолчанию. Если у вас уже есть группы, они будут доступны для выбора. Как и в *Azure*, группы ресурсов позволяют вам объединять ресурсы *IBM Watson Natural Language Understanding*, что облегчает доступ к ним;
 - при желании вы можете добавить теги в разделе **Tags**. Это может в дальнейшем помочь вам организовать и сгруппировать ваши ресурсы.
4. Нажмите на кнопку **Create** в левом верхнем углу окна для создания ресурса.

Шаг 5. Получите ключ API

Этот ключ вы будете использовать для доступа к службе *IBM Watson Natural Language Understanding* и осуществления вызовов API. Очень важно хранить ключи API в безопасном месте.

После нажатия на кнопку **Create** на четвертом шаге должна была открыться страница администрирования созданного ресурса. Если по какой-то причине этого не произошло, вы можете перейти на эту страницу, щелкнув по иконке навигации, показанной на рис. 9.1, в правом верхнем углу страницы.

Среди прочих вы увидите пункт **Resource List**. Выберите его, чтобы перейти к списку ресурсов. Там в разделе **Services** вы обнаружите ресурс **Tone Analyzer**. Щелкните по нему, чтобы перейти на страницу конфигурирования ресурса.

После этого перейдите на вкладку **Manage** в правом верхнем углу экрана, как показано на рис. 9.2.

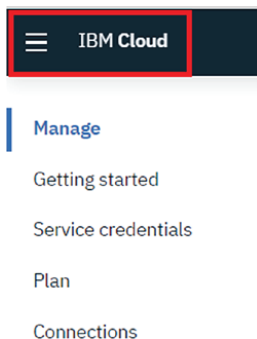


Рис. 9.1 ❖ Навигационное меню IBM Cloud

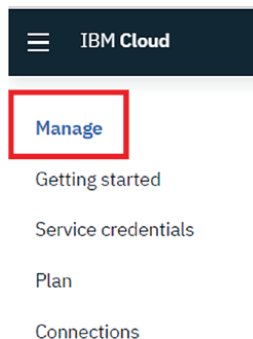


Рис. 9.2 ❖ Пункт для перехода на страницу управления службой

Здесь вы можете скопировать ваш ключ API, нажав на соответствующую иконку справа, как показано на рис. 9.3.



Рис. 9.3 ❖ Доступ к ключу API службы Tone Analyzer

Написание скрипта на Python для анализа настроения текста в IBM Watson

После настройки ресурсов службы *IBM Watson Natural Language Understanding* вы можете приступить к написанию программного кода на Python для анализа текста. Как и в случае с *Microsoft Cognitive Services*, скрипт будет производить оценку данных и сохранять результаты в виде датафрейма, который может быть впоследствии загружен в модель данных Power BI. Ниже приведены шаги, необходимые для реализации нашей задумки.

Шаг 1. Импортируйте необходимые библиотеки и модули

Как и всегда, начнем с загрузки нужных нам библиотек. Здесь их будет немало:

```
import json
import pandas as pd
import ast
import os
from ibm_watson import ToneAnalyzerV3
from ibm_watson.tone_analyzer_v3 import ToneInput
from ibm_cloud_sdk_core.authenticators import IAMAuthenticator
```

Библиотеку *pandas* мы будем использовать для выполнения манипуляций с данными и работы с JSON. Функция *literal_eval()* из библиотеки *ast* понадобится нам для выполнения строк, являющихся выражениями Python, а классы *ToneAnalyzerV3()* и *IAMAuthenticator()* мы будем использовать для выполнения анализа настроения текста. Подробнее о них мы поговорим далее.

Шаг 2. Создайте экземпляр класса IAMAuthenticator

Чтобы сделать это, достаточно запустить следующую строку кода:

```
authenticator = IAMAuthenticator("<ключ API>")
```

На этом шаге выполняется аутентификация в службе. Мы иницилируем создание объекта *IAMAuthenticator*, передавая на вход свой ключ API, полученный ранее. Созданный объект сохраним в переменной *authenticator*.

Шаг 3. Создайте экземпляр класса ToneAnalyzerV3

Для этого запустите следующий фрагмент кода:

```
service = ToneAnalyzerV3(
    version='2019-12-22',
    authenticator=authenticator
)
```

Здесь мы создаем экземпляр класса *ToneAnalyzerV3*, который будем использовать для взаимодействия со службой *Tone Analyzer*. При создании экземпляра нам необходимо передать два параметра: *version* и *authenticator*. Первый из них говорит службе, какую версию API вы хотите использовать. Если версия API была создана в ту дату, которую вы указали, будет использоваться именно она. В противном случае будет использована самая поздняя версия, созданная раньше этой даты. Очень важно указывать ту же версию, которая использовалась во время разработки, чтобы скрипт гарантированно выполнялся.

Шаг 4. Установите ссылку на службу для созданного объекта

Чтобы это сделать, достаточно выполнить следующую строку кода:

```
service.set_service_url(
    "https://gateway.watsonplatform.net/tone-analyzer/api")
```

Ссылку на службу можно найти, перейдя на вкладку **Manage** на странице вашего ресурса.

Шаг 5. Создайте датафрейм с исходными данными для анализа

Здесь все просто. Мы воспользуемся уже хорошо знакомым нам методом *read_csv()* из пакета *pandas*, как показано ниже:

```
dfDocuments = pd.read_csv(
    "<путь к файлу CSV>")
```

Шаг 6. Создайте основу для датафрейма с оценочными данными

Для начала создадим пустой список следующим образом:

```
listReturnedUtterance = []
```

Анализ настроения текста будет выполняться путем отправления каждого документа по отдельности службе *IBM Watson Natural Language Understanding* и сбора ответов в формате JSON.

Примечание. Документом (document) в анализе настроения текста считается группа слов или предложений, представляющая уровень вашего анализа. К примеру, если вы анализируете сообщения в Твиттере, то каждое сообщение будет являться документом. Если речь идет об анализе отзывов посетителей ресторанов, то документом будет считаться каждый отдельный отзыв.

Полученный результат в формате JSON мы будем разбирать, извлекая оценки настроения текста и сохраняя их в созданном ранее списке. Именно

этот список впоследствии станет основой для датафрейма, который будет загружаться в Power BI.

Шаг 7. Определите циклическую конструкцию для отправки документов в службу IBM Watson

Это можно сделать при помощи следующей строки кода:

```
for index, row in dfDocuments.iterrows():
```

Служба анализа настроения текста от *IBM* позволяет посылать до 50 документов в одном пакете, но зачастую вам может потребоваться передавать на анализ больше текстовых фрагментов. Обойти это ограничение можно, посылая документы в службу отдельно – по одному за раз, а не пакетами. Это можно сделать, пройдя по исходному датафрейму *dfDocuments* при помощи метода *iterrows()*. Этот метод позволяет пройти по датафрейму и извлечь из него пары из индекса и строки. При этом во внутренней переменной цикла *index* сохраняется значение индекса строки, а в переменной *row* – соответствующая строка в виде объекта *Series*. Такой способ перебора датафрейма позволяет взаимодействовать со всеми нужными нам столбцами построчно.

Шаг 8. Отформатируйте и оцените настроение текста в документе

Фрагмент кода, который поможет нам выполнить эту задачу, приведен ниже:

```
submissionText = row["text"].replace("'", "")
submissionText = submissionText[0:500]
PythonExpression = "[{'text': '" + submissionText + "'}]"
Submission = ast.literal_eval(PythonExpression)
tone_chat = service.tone_chat(Submission).get_result()
```

Служба *IBM Watson Natural Language Understanding* принимает на вход документы в виде списка словарей. Мы решили посылать по одному документу за раз так, что наш запрос должен быть оформлен примерно следующим образом:

```
[{'text': '<документ для анализа>'}]
```

Как видите, это список с единственным элементом, представляющим собой словарь. Наши текстовые фрагменты для анализа содержатся в датафрейме, так что нам необходимо выполнить кое-какие предварительные действия для преобразования их в требуемый формат.

В первой строке представленного выше фрагмента кода мы избавляемся в оригинальном тексте для анализа от одинарных кавычек. Мы будем использовать одинарные кавычки для обрамления текстовых данных, так что в источнике нам необходимо избавиться от них, чтобы не возникало ошибок.

Во второй строке кода мы оставляем для анализа только первые 500 символов из исходного текста, поскольку служба *IBM Watson* налагает именно

такое ограничение. Цифра 0 говорит о том, что мы будем брать 500 знаков, начиная с первого символа строки.

Примечание. В Python индексы начинаются с нуля, а в R – с единицы. Именно по этой причине минимальным индексом в создаваемых в Python конструкциях будет 0, а в R – 1.

Число 500 указывает на индекс последнего символа в диапазоне, не включая его самого. Таким образом, мы извлечем все символы, начиная с нулевого и заканчивая тем, что предшествует 500-му. В случае если в строке будет меньше 500 знаков, будет извлечено все ее содержимое. Таким образом мы ограничили максимальное количество возвращаемых символов пятьюстами – именно такой предел принят в службе *IBM Watson*.

Далее мы создаем строку, представляющую выражение на языке Python. В данном случае результат будет являться списком с единственным элементом в виде словаря. Возвращенное значение присвоим переменной *Submission*.

В заключительной строке кода мы вызвали метод *tone_chat()* созданного ранее объекта *service*, чтобы отправить наш документ службе *IBM Watson Natural Language Understanding*. Результат сохраним в переменной *tone_chat*.

Шаг 9. Извлеките результат анализа текста и инициализируйте переменные исходными значениями

Это позволит сделать следующий фрагмент кода:

```
utterances = tone_chat["utterances_tone"]
```

```
sad, frustrated, satisfied, excited, \
polite, impolite, sympathetic, unknown = \
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
```

В первой строке представленного кода мы получаем словарь *utterances_tone*, возвращенный службой *IBM Watson Natural Language Understanding*, и присваиваем его переменной *utterances*. Формат полученного словаря будет представлен документом JSON примерно такого вида:

```
{'utterances_tone':
  [{ 'utterance_id': 0,
     'utterance_text': '<документ для анализа>',
     'tones': [{ 'score': <между 0 и 1 для tone 1>,
                  'tone_id': '<tone_id 1>',
                  'tone_name': '<tone_name 1>' },
               .....
               { 'score': <между 0 и 1 для tone n>,
                  'tone_id': '<tone_id n>',
                  'tone_name': '<tone_name n>' }
             ]
    ]
}
```

В представленном выше формате JSON нас главным образом будет интересовать узел *tones*. Обратите внимание, что для одного посланного документа может быть возвращено сразу несколько тонов. Все возможные оттенки тонов представлены переменными, которые мы инициализировали нулями на этом шаге.

Примечание. Обратите внимание на то, как был использован символ обратной косой черты в коде для переноса строк. Три строки кода по своей сути представляют одну строку, которая оказалась бы слишком длинной. И для переносов мы воспользовались обратными слешами.

Для каждого полученного тона у нас есть параметры *score*, *id* и *tone_name*. На следующем шаге мы пройдемся по полученным из службы тонам для оценки их характеристик.

Шаг 10. Пройдите по тонам и присвойте их значения соответствующим переменным

Для нашего цикла мы используем следующий код:

```
tones = utterances[0]["tones"]
for tone in tones:
    toneid = tone["tone_id"]
    if toneid == "sad":
        sad = tone["score"]
    elif toneid == "frustrated":
        frustrated = tone["score"]
    elif toneid == "satisfied":
        satisfied = tone["score"]
    elif toneid == "excited":
        excited = tone["score"]
    elif toneid == "polite":
        polite = tone["score"]
    elif toneid == "impolite":
        impolite = tone["score"]
    elif toneid == "sympathetic":
        sympathetic = tone["score"]
    else:
        unknown = tone["score"]

listReturnedUtterance.append(
    [index, sad, frustrated, satisfied, excited,
     polite, impolite, sympathetic, unknown])
```

Поскольку мы передавали в службу по одному документу за раз, возвращаемые значения будут характеризовать один конкретный документ или фрагмент текста. И хотя информация, которую мы запросили, относится к единственному документу, нам необходимо дать Python явно понять, что нам нужен первый элемент. И мы сделали это при помощи следующего выражения: `utterances[0][“tones”]`. Нулевой индекс позволяет обратиться к пер-

вому элементу в переменной *utterance*, а при помощи [“tones”] мы извлекаем все тона из документа. Результат сохраним в переменной *tones*.

После этого пройдемся по элементам этой переменной и присвоим оценки соответствующим тонам. К примеру, если значение атрибута *toneid* текущего элемента итерации будет равно строке «impolite», значение одноименной переменной, которую мы ранее инициализировали нулем, будет изменено на текущее значение атрибута *score*. После завершения цикла индекс текущей итерации вместе с переменными тонов, которым были присвоены новые значения, объединяются в список и добавляются в изначально пустой список *listReturnedUtterance*.

Шаг 11. Создайте датафрейм на основе списка *listReturnedUtterance*

Для создания датафрейма достаточно будет выполнить следующие строки кода:

```
colnames = ["index", "sad", "frustrated", "satisfied", "excited",
"polite", "impolite", "sympathetic", "unknown"]

dfReturnedUtterance = pd.DataFrame(
    listReturnedUtterance, columns=colnames)
```

Преобразовать список *listReturnedUtterance* в датафрейм можно проще простого, применив знакомый уже нам метод *DataFrame()* из библиотеки *pandas*. На вход методу мы передадим два аргумента. Первый – это список, который мы хотим преобразовать в датафрейм, – в нашем случае это список из переменной *listReturnedUtterance*. Второй аргумент с именем *columns* отвечает за имена столбцов, которые мы хотим видеть в датафрейме. Передадим их в виде списка.

Шаг 12. Объедините датафреймы *dfReturnedUtterance* и *dfDocuments*

Ниже приведен код, который необходимо выполнить для этого:

```
dfOutput = pd.merge(
    dfDocuments, dfReturnedUtterance,
    how='inner', left_on = 'id', right_on = 'index')

dfOutput = dfOutput[
    ['id', 'text', 'sad', 'frustrated', 'satisfied',
    'excited', 'polite', 'impolite', 'sympathetic', 'unknown']]
```

Здесь мы объединяем датафреймы *dfReturnedUtterance* и *dfDocuments* при помощи метода *merge()* из библиотеки *pandas*. Для этого мы передаем в метод в качестве первого аргумента датафрейм, который в объединении должен находиться слева, а в качестве второго – датафрейм, который будет справа. Далее в аргументе *how* мы указываем тип объединения. В данном примере

мы использовали значение «inner» (внутреннее объединение). В заключение даем *pandas* понять, по каким полям хотим связать наши датафреймы. Имя связующей колонки из левого датафрейма мы передаем в аргументе *left_on*, а из правого – в аргументе *right_on*.

Шаг 13. Скопируйте скрипт в Power BI

Ниже приведен полный скрипт, который вы можете скопировать во встроенный редактор Python в Power BI при помощи инструмента получения данных (GetData). Скрипт возвращает сразу несколько датафреймов, которые будут доступны в качестве исходных таблиц в Power BI. Итоговым датафреймом, в котором собрана вся интересующая нас информация, является *dfOutput*:



```
import json
import pandas as pd
import ast
import os

from ibm_watson import ToneAnalyzerV3
from ibm_watson.tone_analyzer_v3 import ToneInput
from ibm_cloud_sdk_core.authenticators import IAMAuthenticator

authenticator = IAMAuthenticator("<ваш ключ API>")

service = ToneAnalyzerV3(
    version='2019-12-22',
    authenticator=authenticator)

service.set_service_url(
    "https://gateway.watsonplatform.net/tone-analyzer/api")

dfDocuments = pd.read_csv(
    "<путь к файлу CSV>")

listReturnedUtterance = []

for index, row in dfDocuments.iterrows():
    submissionText = row["text"].replace("'", "")
    submissionText = submissionText[0:500]
    PythonExpression = "[{'text': '" + submissionText + "'}]"
    Submission = ast.literal_eval(PythonExpression)
    tone_chat = service.tone_chat(Submission).get_result()

    utterances = tone_chat["utterances_tone"]

    sad, frustrated, satisfied, excited, \
        polite, impolite, sympathetic, unknown = \
        0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0

    tones = utterances[0]["tones"]
    for tone in tones:
        toneid = tone["tone_id"]
        if toneid == "sad":
            sad = tone["score"]
```

```

elif toneid == "frustrated":
    frustrated = tone["score"]
elif toneid == "satisfied":
    satisfied = tone["score"]
elif toneid == "excited":
    excited = tone["score"]
elif toneid == "polite":
    polite = tone["score"]
elif toneid == "impolite":
    impolite = tone["score"]
elif toneid == "sympathetic":
    sympathetic = tone["score"]
else:
    unknown = tone["score"]

listReturnedUtterance.append(
    [index, sad, frustrated, satisfied, excited,
     polite, impolite, sympathetic, unknown])

colnames = ["index", "sad", "frustrated", "satisfied", "excited",
            "polite", "impolite", "sympathetic", "unknown"]

dfReturnedUtterance = pd.DataFrame(
    listReturnedUtterance, columns=colnames)

dfOutput = pd.merge(
    dfDocuments, dfReturnedUtterance,
    how='inner', left_on = 'id', right_on = 'index')

dfOutput = dfOutput[
    ['id', 'text', 'sad', 'frustrated', 'satisfied',
     'excited', 'polite', 'impolite', 'sympathetic', 'unknown']]

```

В данной главе мы рассмотрели технические нюансы применения сторонних моделей искусственного интеллекта к вашим моделям данных в Power BI при помощи R и Python. Вы также узнали, как можно легко преобразовывать данные в требуемый для моделей машинного обучения формат. Сделать подобные преобразования при помощи встроенных средств Power Query было бы не так просто. Наконец, мы посмотрели, как можно использовать готовые модели облачных служб, таких как *Microsoft Cognitive Services* и *IBM Watson Natural Language Understanding*. В заключительной главе данной книги вы узнаете о применении похожих техник в Power BI на корпоративном уровне.

Глава 10

.....

Создание моделей анализа данных и скриптов для обработки информации

Скрипты обработки и оценки данных, которые мы написали в предыдущей главе, прекрасно работают в небольших компаниях, где за модели данных в Power BI отвечает один человек. Причина в том, что в таких компаниях вам может быть достаточно использовать персональный режим *локального шлюза данных* (on-premises data gateway). В стандартном режиме, который зачастую требуется для больших корпоративных решений, не допускается использование встроенных скриптов на языках R и Python в Power BI. К счастью, есть способ обойти это ограничение путем использования такого относительно нового инструмента в составе *SQL Server*, как *служба машинного обучения SQL Server (SQL Server Machine Learning Services – SSMLS)*. С применением *SSMLS* допустимо выполнять серьезные вычисления, связанные с анализом данных, непосредственно в базе данных посредством скриптов на языках R и Python, облаченных в специальные хранимые процедуры T-SQL. А поскольку вы имеете полное право обращаться к данным в хранимых процедурах при использовании стандартного режима локального шлюза данных, вы вполне можете переделать свои скрипты из предыдущей главы для использования совместно с хранимыми процедурами в корпоративном решении Power BI.

В данной главе мы разберем несколько примеров *рефакторинга* (refactor) скриптов, а в качестве бонуса покажем, как использовать встроенную в *SSMLS* модель для проведения анализа тональности текста абсолютно бесплатно. Для начала давайте проведем рефакторинг скрипта для оценки цен на недвижимость в Бостоне из предыдущей главы под использование службы *SSMLS*.

ПРОГНОЗИРОВАНИЕ ЦЕН НА НЕДВИЖИМОСТЬ В POWER BI С ПОМОЩЬЮ R СО СЛУЖБОЙ SSMLS

В данном разделе мы переработаем скрипт, который написали в главе 9, об-
лачив его в специальную хранимую процедуру. Это позволит использовать
скрипт, написанный на языке R, в корпоративном решении со стандартным
режимом локального шлюза данных. Ниже приведены необходимые шаги
для выполнения этих действий.

Написание скрипта на языке R для добавления модели в SQL Server

Ниже приведен скрипт, необходимый для добавления модели в *SQL Server*:

```
library(RODBC)

# Загрузка модели R в нашу сессию
model <- readRDS("./Models/Model.rds")

# Подключение к базе данных
server.name = "DSVM2019"
db.name = "BostonHousingData"
connection.string = paste(
  "driver={SQL Server}", ";",
  "server=", server.name, ";",
  "database=", db.name, ";",
  "trusted_connection=true", sep = "")

conn <- odbcDriverConnect(connection.string)

# Определение параметров
model_name <- "R Model"
modelbin <- serialize(model, NULL)
modelbinstr = paste(modelbin, collapse = "")

# Создание и запуск выражения SQL
# с использованием команды sqlQuery
sql_code <- paste0(
  "EXEC AddModel_R ",
  "@ModelName='",
  model_name, "'",
  "@Model_Serialized='",
  modelbinstr, "'")

sqlQuery(conn, sql_code)

odbcClose(conn)
```



Теперь пройдем по этому листингу более детально.

Шаг 1. Загрузите необходимые пакеты

```
library(RODBC)
```

Здесь нам понадобится всего один пакет с именем *RODBC*. Этот пакет будет использован для осуществления взаимодействий с *SQL Server*.

Шаг 2. Загрузите модель R в вашу сессию

Чтобы загрузить модель с диска, достаточно выполнить следующую строку кода:

```
model <- readRDS("./Models/Model.rds")
```

На данном этапе мы загружаем модель в R с использованием функции *readRDS()*, входящей в базовую установку R. Вы можете использовать относительный путь, если ваша рабочая директория установлена в папку R из главы 10.

Шаг 3. Подключитесь к базе данных

Следующий фрагмент кода поможет вам выполнить подключение к базе данных *SQL Server*:

```
# Подключение к базе данных
server.name = "DSVM2019"
db.name = "BostonHousingData"
connection.string = paste(
  "driver={SQL Server}", ";",
  "server=", server.name, ";",
  "database=", db.name, ";",
  "trusted_connection=true", sep = "")
conn <- odbcDriverConnect(connection.string)
```

Строка подключения собирается на основании имени сервера и базы данных. Для этого мы предварительно инициализируем переменные *server.name* и *db.name* соответственно. Эта информация соединяется с другими конфигурационными установками для выполнения подключения, в результате чего мы получаем строковую переменную *connection.string*, в которой собраны все необходимые сведения для соединения с нашей базой данных. На заключительном этапе переменная *connection.string* передается в функцию *odbcDriverConnection()* для создания объекта подключения, которое сохраняется в переменной *conn*.

Шаг 4. Определите переменные модели

Код, необходимый для определения переменных модели, приведен ниже:

```
model_name <- "R Model"
modelbin <- serialize(model, NULL)
modelbinstr = paste(modelbin, collapse = "")
```

Переменная *model_name* используется для хранения имени модели. На следующем шаге мы используем ее для идентификации модели в таблице *dbo.Models*. Модель сериализуется при помощи функции *serialize()*, а результат сериализации сохраняется в переменной *modelbin*. Затем созданная переменная *modelbin* преобразуется в скалярный символьный вектор с использованием аргумента *collapse* функции *paste()*, а результат попадает в переменную *modelbinstr*. Именно эта переменная теперь представляет модель, которая будет впоследствии загружена в базу данных.

Шаг 5. Напишите выражение на T-SQL для добавления модели в базу данных

Ниже приведен код для запроса:

```
# Создание и запуск выражения SQL
# с использованием команды sqlQuery
sql_code <- paste0(
  "EXEC AddModel_R ",
  "@ModelName=",
  model_name, "'",
  "@Model_Serialized=",
  modelbinstr, "'")
```



Функция *paste0()* используется для построения запроса на языке T-SQL, в котором мы запускаем хранимую процедуру (stored procedure) с именем *AddModel_R* в *SQL Server*. В процедуре *AddModel_R* используется инструкция *INSERT* для вставки модели в *SQL Server* с применением параметров, определенных нами на шаге 4. Инструкция *VALUES* используется совместно с *INSERT* для передачи имени модели, определенного в переменной *model_name*, и текстового представления модели, хранящегося в переменной *modelbinstr*. Процедура *AddModel_R* уже будет сконфигурирована в базе данных, которую вы создадите на сервере на одном из следующих шагов. После добавления базы данных вы можете внимательно изучить текст хранимой процедуры *dbo.AddModel_R*, присутствующей в базе.

Шаг 6. Добавьте код, необходимый для запуска выражения T-SQL из R

Фрагмент кода для этого приведен ниже:

```
sqlQuery(conn, sql_code)
odbcClose(conn)
```

Запрос на языке T-SQL, построенный на шаге 5, отправляется на выполнение с помощью функции *sqlQuery()* из пакета *RODBC*. В качестве аргументов мы передали объект подключения *conn*, инициализированный ранее, и переменную *sql_code*, в которой хранится выражение T-SQL. После выполнения запроса мы закрываем подключение к базе данных при помощи функции *odbcClose()*.

Шаг 7. Сохраните скрипт

Убедитесь, что вы сохранили свой скрипт, если вы писали его с нуля. Нам он понадобится на следующих этапах для добавления модели в *SQL Server*. Обратите внимание, что скрипт есть в репозитории к этой книге.

Использование SSMLS совместно с R для оценки данных

В данном разделе мы опишем шаги, необходимые для добавления на сервер базы данных, использующейся в нашем примере. Также мы посмотрим, как сконфигурировать базу данных и провести оценку информации для нашей задачи.

Шаг 1. Запустите *SQL Server Management Studio*

Если вы используете рекомендованную установку *Windows 2019 DSVM*, значит, *SQL Management Studio (SSMS)* и *SQL Server 2019* с запущенной службой *SQL Server Machine Learning Services* у вас уже есть. Обратитесь к инструкции во введении к этой книге, если какие-то из этих ресурсов не установлены в вашем окружении.

Шаг 2. Создайте подключение к серверу, который хотите использовать

При запуске *SQL Server Management Studio (SSMS)* имя сервера, к которому вы подключаетесь, нужно ввести в соответствующем поле. Если его нет в списке и вы не знаете имя сервера, вы можете ввести точку, и *SSMS* подключится к локальному серверу в вашем окружении.

Шаг 3. Добавьте базу данных *BostonHousingInfo* на ваш сервер

Резервная копия базы данных, которую мы будем использовать в этом примере, располагается в папке *Databases* из этой главы. Имя файла нужной вам резервной копии – *BostonHousingData.bak*. Выполните следующие шаги, чтобы восстановить базу данных из резервной копии.

1. Скопируйте файл *BostonHousingData.bak* в папку *Backup* вашего *SQL Server*. В виртуальной машине путь будет следующим: *C:\Program Files\Microsoft SQL Server\MSSQL15.MSSQLSERVER\MSSQL\Backup*.
2. Откройте *SQL Server Management Studio (SSMS)*, щелкните правой кнопкой мыши по пункту **Базы данных** (Databases) и выберите пункт **Восстановить базу данных** (Restore Database).
3. В открывшемся окне **Восстановление базы данных** (Restore Database) на вкладке **Общие** (General) установите переключатель **Источник** (Source) в положение **Устройство** (Device).

4. Щелкните по кнопке с тремя точками и в открывшемся диалоговом окне нажмите на кнопку **Добавить** (Add), после чего выберите путь к файлу *BostonHousingData.bak*.
5. Выберите файл *BostonHousingData.bak* и нажмите на кнопку **ОК**, чтобы закрыть окно **Выбор устройств резервного копирования** (Locate Backup File).
6. Нажмите на кнопку **ОК**, что приведет к добавлению новой базы данных в список.

База данных, добавленная на ваш сервер на этом шаге, содержит все необходимые объекты и данные для проведения оценки.

Шаг 4. Добавьте модель в базу данных

Вернитесь к скрипту *AddModelToDatabase.R* в *R Studio*. Убедитесь, что рабочая директория установлена на папку *R*. Запустите скрипт. Если вы все настроили правильно, модель R будет добавлена в таблицу *dbo.Models* в базе данных *BostonHousingInfo*.

Шаг 5. Создайте в базе данных хранимую процедуру для прогноза

Ниже приведен текст хранимой процедуры:

```
CREATE PROCEDURE [dbo].[uspPredictHousePrices_R]
AS
BEGIN
    -- Определяем переменные
    DECLARE @model varbinary(max) =
        (SELECT MODEL
         FROM [dbo].[Models]
         WHERE ModelName = 'R Model');
    DECLARE @RScript nvarchar(max);
    DECLARE @Query nvarchar(max);
    DECLARE @InputDFName nvarchar(25);
    DECLARE @OutputDFName nvarchar(25);

    -- Определяем источник данных
    SET @Query='SELECT [crim], [rm], [tax], [lstat]
    FROM [dbo].[BostonHousingInfo]'

    -- Скрипт на языке R для оценки данных
    SET @RScript = N'
    bhmodel_deserialized <-
        unserialize(as.raw(bhmodel_serialized));
    model_data <- dfInputData
    pred_medv <-
        predict(bhmodel_deserialized, model_data)
    dfOutputData <-
        cbind(model_data, pred_medv)'
```

```

SET @InputDFName = 'dfInputData'
SET @OutputDFName = 'dfOutputData'

EXEC sp_execute_external_script
    @language = N'R'
    ,@script = @RScript
    ,@input_data_1 = @Query
    ,@input_data_1_name = @InputDFName
    ,@output_data_1_name = @OutputDFName
    ,@params = N'@bhmodel_serialized varbinary(max)'
    ,@bhmodel_serialized = @model

WITH RESULT SETS((
    [crim] float
    ,[rm] float
    ,[tax] float
    ,[lstat] float
    ,[pred_medv] float
));
END

```

Рассмотрим подробно, что происходит в приведенной выше процедуре.

1. Определяются следующие переменные, необходимые для скрипта:
 - *@model* – переменная для хранения модели. Саму модель мы извлекаем из таблицы *dbo.Models* с помощью инструкции T-SQL;
 - *@RScript* – в этой переменной мы будем хранить скрипт на языке R для проведения оценки;
 - *@Query* – здесь будет храниться скрипт на языке запросов T-SQL для определения набора данных для оценки;
 - *@InputDFName* – в этой переменной мы будем хранить имя входного набора данных, к которому будет обращаться R;
 - *@OutputDFName* – в этой переменной мы будем хранить имя выходного набора данных, к которому будет обращаться R.
2. В переменную *@Query* записывается инструкция на языке запросов T-SQL для получения входного набора данных. В этом наборе будут представлены данные для оценки в R.
3. В переменной *@RScript* хранится скрипт на языке R для выполнения прогноза. В этом скрипте выполняются следующие действия:
 - в переменную *bhmodel_serialized* загружается модель в необработанном формате, после чего производится десериализация путем запуска функции *unserialize()*, и итоговая модель присваивается переменной *bhmodel_deserialized*;
 - создается переменная *model_data* на основе датафрейма со входным набором данных *dfInputData*. Хорошей практикой считается не использовать напрямую наборы данных, переданные в R из *SQL Server*;
 - вызывается функция *predict()* для выполнения прогноза. Первым аргументом в нее передается модель, а вторым – данные, которые не-

обходимо проанализировать. На выходе мы получаем вектор с выполненным прогнозом, который сохраним в переменной *pred_medv*;

Примечание. Обратите внимание, что мы включили полный датафрейм, поскольку он содержит только столбцы, необходимые для формулы регрессии. В других ситуациях вам может понадобиться провести предварительные преобразования датафрейма перед тем, как передать его в скрипт R, чтобы он отвечал требованиям формулы модели.

- создается датафрейм *dfOutputData* с использованием функции *cbind()* путем объединения по столбцам набора данных *model_data* с вектором *pred_medv*.
- 4. Устанавливаются имена для входного и выходного датафреймов.
- 5. Конфигурируется хранимая процедура *sp_execute_external_script*. По большей части входные параметры этой функции вам должны быть понятны, мы лишь остановимся чуть подробнее на параметрах *@params* и *@bhmodel_serialized*. Параметр *@params* позволяет вам определить дополнительные аргументы, необходимые для модели. В нашем случае мы определили параметр с именем *@bhmodel_serialized* с типом данных *varbinary(max)*. С его помощью мы передадим в скрипт R модель и сделаем ее доступной для скрипта по имени без знака *@* – в данном случае это *bhmodel_serialized*. Значение параметра *@bhmodel_serialized* устанавливается в следующей строке.
- 6. Инstrukция *WITH RESULT SETS* позволяет установить имена и типы данных для столбцов. Это поможет Power BI определиться с именами и типами колонок в наборе данных, переданном ему из *SQL Server*.

Выполните полученный скрипт, чтобы создать хранимую процедуру в базе данных. Теперь вы можете проводить оценку данных путем вызова процедуры.

Шаг 6. Извлеките данные с прогнозами из SQL Server в Power BI

Для этого вам необходимо открыть Power BI, нажать на выпадающую кнопку **Получить данные** (GetData) и выбрать пункт **SQL Server**. Заполните поля **Сервер** (Server) и **База данных** (Database). Затем раскройте секцию **Расширенные параметры** (Advanced options) и введите следующий текст в поле **Инструкция SQL** (SQL statement): EXEC [dbo].[uspPredictHousePrices_R]. Флажок **Включить столбцы отношений** (Include relationship columns) по умолчанию установлен, но нам он не нужен, так что снимите его. На рис. 10.1 показано, как должна выглядеть заполненная форма.

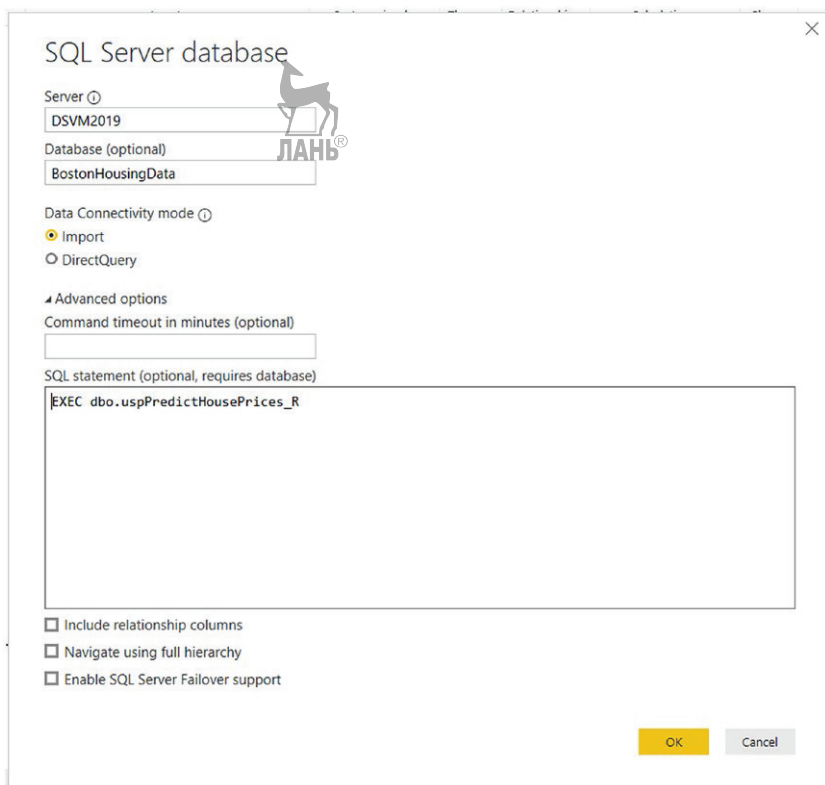


Рис. 10.1 ❖ Окно получения данных
для запуска процедуры `dbo.uspPredictedHousePrices_R`

Нажмите на кнопку **ОК** для добавления информации в модель данных Power BI.

В данном примере не стоит обращать внимания на модель, которая базировалась на простой формуле линейной регрессии. Мы решили упростить ее до предела, чтобы сконцентрировать внимание на применении модели R для оценки данных в Power BI при помощи службы *SSMLS*. Мы не собирались строить сложную модель, поскольку проектирование моделей машинного обучения выходит за рамки этой книги. Но если вы или аналитик данных в вашей компании знаете, как строить полноценные модели прогнозирования в R, вы можете использовать приведенные в данном разделе шаги для применения их к моделям данных Power BI с использованием службы *SSMLS*. В следующем разделе мы посмотрим, как можно работать с *SSMLS* при помощи языка Python.



ПРОГНОЗИРОВАНИЕ ЦЕН НА НЕДВИЖИМОСТЬ В POWER BI С ПОМОЩЬЮ PYTHON СО СЛУЖБОЙ SSMLS

В данном разделе мы произведем рефакторинг скрипта на языке Python, который написали в главе 9 для прогнозирования цен на недвижимость, заключив его в специальную хранимую процедуру. Это позволит запускать скрипт в рамках корпоративного решения с использованием стандартного режима локального шлюза данных. Ниже приведены шаги для реализации такого алгоритма.

Написание скрипта на языке Python для добавления модели в SQL Server

Шаг 1. Подберите версии библиотек

Библиотеки, которые нам понадобятся в данном примере, – это *pickle*, *os*, *pyodbc*, *scikit-learn* и *pandas*. Вам необходимо убедиться, что версии библиотек, которые вы используете при разработке, совпадают с версиями, установленными в SSMLS. Узнать, какие версии пакетов используются в SSMLS, можно, запустив следующий скрипт T-SQL в SQL Server:

```
EXECUTE sp_execute_external_script
    @language = N'Python'
    ,@script = N'
import pkg_resources
import pandas as pd
installed_packages = pkg_resources.working_set
installed_packages_list =
    sorted(["%s=%s" % (i.key, i.version) for i in installed_packages])

df = pd.DataFrame(installed_packages_list)
OutputDataSet = df'

WITH RESULT SETS (( PackageVersion nvarchar (150) ))
```

На выходе мы получим список всех пакетов, установленных в Python в службе SSMLS. На момент написания книги версии библиотек в службе SSMLS 2019 были следующими:

- pyodbc 4.0.25;
- scikit-learn 0.20.2;
- pandas 0.23.4.

Обратите внимание на отсутствие библиотек *os* и *pickle* в представленном списке. Причина в том, что эти библиотеки включены во все установки Python по умолчанию.

Шаг 2. Создайте окружение *conda*

Нужно обязательно убедиться, что разработку вы ведете в той же версии Python, которая установлена в службе *SSMLS*, во избежание дальнейшего конфликта версий. Для этого требуется настроить отдельное окружение *conda*.

Примечание. Окружение *conda* позволяет вам полностью изолировать свой проект, определив свою версию Python, список загруженных библиотек и их версии. Также вам доступно множество других полезных настроек в окружении. Что отличает окружение *conda* от других, так это то, что оно не зависит от языка, так что вы можете использовать его с любым другим языком программирования.

Выполните следующие действия для создания окружения *conda*.

1. Откройте *VS Code*. Для этого запустите командную строку и введите следующую команду для изменения рабочей директории:

```
cd <"путь к папке Python из главы 1">
```

После этого введите следующую инструкцию для открытия *VS Code*:

```
code .
```

2. Откройте *командную оболочку* (terminal shell) в *VS Code* по указанному пути. Нажмите сочетание клавиш **CTRL+SHIFT+`**, чтобы открыть оболочку. В данном примере мы используем *Command Prompt*. Если требуется изменить оболочку, откройте окно настройки оболочки, расположенное в верхней правой области, как показано на рис. 10.2.

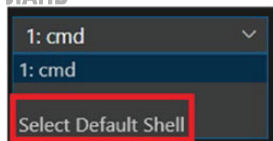


Рис. 10.2 ❖ Окно настройки оболочки в *VS Code*

В этой области вам будет предоставлен выбор, показанный на рис. 10.3.

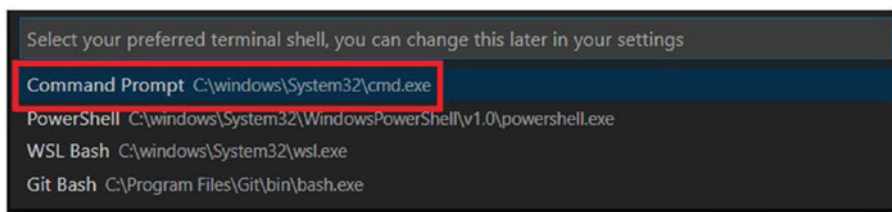


Рис. 10.3 ❖ Опции выбора оболочки

Управляйте выделенным пунктом, чтобы выбрать оболочку *Command Prompt*.

3. Создайте окружение *conda* с именем *ssmls*. Вам понадобится окружение, в котором будет использоваться та же версия Python и необходимых библиотек, что и в службе *SSMLS*. Версия Python, используемая в *SSMLS 2019*, – это Python 3.7.3. Введите следующую инструкцию в командную строку, чтобы создать окружение с использованием этой версии Python:

```
conda create --name ssmls python=3.7.3
```

4. Переключитесь на созданное окружение. На шаге 3 мы создали окружение, но не переключились на него. Если вы работаете в рекомендованной виртуальной машине для анализа данных, в вашем распоряжении, помимо созданного, будет множество других предустановленных окружений. Вы можете ввести следующую команду, чтобы получить полный список доступных вам окружений:

```
conda env list
```

Если вы работаете в *DSVM*, вы увидите несколько окружений и созданное вами окружение *ssmls*. Работая на персональном компьютере, вы можете увидеть только окружение *ssmls*, базовое окружение и другие, которые создавали лично. Введите представленную ниже команду, чтобы активировать окружение *ssmls*:

```
activate ssmls
```

Теперь вы находитесь в окружении *ssmls*, базирующемся на той же версии Python, которая используется в службе *SSMLS 2019*.

5. Установите требуемые библиотеки в окружение *ssmls*. Вам понадобятся библиотеки *pandas*, *scikit-learn* и *pyodbc*. Начнем с установки библиотеки *pandas*. Для этого введите следующую команду:

```
conda install pandas=0.23.4
```

После этого начнется процесс установки. То же самое вам нужно будет сделать с пакетами *scikit-learn* версии 0.20.2 и *pyodbc* версии 4.0.25. Убедитесь, что поставили знаки равенства между названиями пакетов и их версиями, как показано выше на примере библиотеки *pandas*.

Шаг 3. Напишите код для загрузки модели в SQL Server

Ниже представлен скрипт на языке Python для загрузки модели в базу данных:

```
import os
import pyodbc
import pickle

server = 'DSVM2019'
```

```

database = 'BostonHousingData'
con = pyodbc.connect(
    'Trusted_Connection=yes',
    driver = '{SQL Server}',
    server = server,
    database = database
)

cursor = con.cursor()

model = pickle.load(open("./Models/model.pkl", "rb"))
modelstr = pickle.dumps(model)

cursor.execute(
    "INSERT INTO [dbo].[Models](ModelName, Model) VALUES (?, ?)",
    "Python Model", modelstr
)

con.commit()
con.close()

```



Библиотеку *os* мы, как и всегда, используем для осуществления взаимодействия с файловой системой, при помощи пакета *pyodbc* подключаемся к *SQL Server*, а библиотеку *pickle* используем для загрузки и сохранения объекта модели. Ниже приведены действия, которые выполняются в скрипте.

1. Создается объект подключения к базе данных с именем *con* с использованием режима проверки подлинности Windows. При подключении используются имя сервера и базы данных из переменных, инициализированных ранее.
2. Создается объект *cursor*, при помощи которого будет выполняться запрос на языке T-SQL.
3. Заранее сохраненная на диске модель загружается в сессию Python при помощи функций *open()* и *pickle.load()*. Первая открывает модель в режиме доступа «rb», что означает *read binary*, то есть доступ двоичного файла на чтение. После этого модель загружается в сессию с помощью функции *pickle.load()*, а результат присваивается переменной *model*. Далее модель сериализуется при помощи функции *pickle.dumps()*, а вывод помещается в переменную *modelstr*.
4. Вызывается метод *execute()* объекта *cursor* для выполнения запроса на T-SQL в нужной нам базе данных. Первым аргументом в метод передается выражение T-SQL. В нашем случае это инструкция *INSERT*, позволяющая вставить модель с ее именем в заранее подготовленную таблицу. Вопросительные знаки используются в качестве заполнителей для параметров *ModelName* и *Model*. Следующими двумя аргументами как раз передаются эти параметры. Заметьте, что порядок следования аргументов крайне важен.
5. Скрипт сохраняется в файле с именем *AddModelToDatabase.py*. Оставьте файл открытым, позже мы вернемся к нему.

Использование SSMLS совместно с Python для оценки данных

В данном разделе мы создадим базу данных *BostonHousingData* на вашем сервере и используем скрипт, написанный на предыдущем шаге, для добавления в нее нашей модели. После этого напишем хранимую процедуру с использованием скрипта на Python для выполнения оценки данных. Шаги, необходимые для этого, приведены ниже.

Шаг 1. Запустите SQL Server Management Studio

Если вы используете рекомендованную установку *Windows 2019 DSVM*, значит, *SQL Management Studio (SSMS)* и *SQL Server 2019* с запущенной службой *SQL Server Machine Learning Services* у вас уже есть. Я настоятельно рекомендую использовать *DSVM*, поскольку это позволяет значительно снизить объем усилий по настройке окружения. В противном случае вам придется устанавливать *SQL Server 2019* с *SSMLS* самостоятельно. Обратитесь к инструкции во введении к этой книге, если какие-то из этих ресурсов не установлены в вашем окружении.

Шаг 2. Создайте подключение к серверу, который хотите использовать

При запуске *SQL Server Management Studio (SSMS)* имя сервера, к которому вы подключаетесь, нужно ввести в соответствующем поле. Если его нет в списке и вы не знаете имя сервера, вы можете ввести точку, и *SSMS* подключится к локальному серверу в вашем окружении.

Шаг 3. Добавьте базу данных *BostonHousingInfo* на ваш сервер

Резервная копия базы данных, которую мы будем использовать в этом примере, располагается в папке *Databases* из этой главы. Имя файла нужной вам резервной копии – *BostonHousingData.bak*. Все шаги для восстановления базы данных из резервной копии описаны в предыдущем разделе для R, так что мы их дублировать не будем.

База данных, добавленная на ваш сервер на этом шаге, будет содержать все необходимые объекты и данные для проведения оценки. Информация, используемая в этом примере, была получена при помощи функции *load_boston()* из *sklearn.datasets*. Код, использованный для извлечения данных, содержится в репозитории.

Шаг 4. Добавьте модель в базу данных

Вернитесь к скрипту *AddModelToDatabase.py* в *VS Code*. Убедитесь, что рабочая директория установлена на папку *Python*. Запустите скрипт. Если вы все

настроили правильно, модель Python будет добавлена в таблицу *dbo.Models* в базе данных *BostonHousingInfo*.

Шаг 5. Создайте в базе данных хранимую процедуру для прогноза

Ниже приведен текст хранимой процедуры:

```
CREATE PROCEDURE [dbo].[uspPredictHousePrices_Python]
AS
BEGIN

    DECLARE @model VARBINARY(max) =
        (SELECT MODEL
         FROM [dbo].[Models]
         WHERE ModelName = 'Python Model');
    DECLARE @PythonScript nvarchar(max);
    DECLARE @Query nvarchar(max);
    DECLARE @InputDFName nvarchar(25);
    DECLARE @OutputDFName nvarchar(25);

    -- Определяем источник данных
    SET @Query='SELECT [crim], [rm], [tax], [lstat]
    FROM [dbo].[BostonHousingInfo]';

    -- Скрипт на Python для оценки данных
    SET @PythonScript = N'
import pickle
from sklearn import linear_model
bhmodel_deserialized = pickle.loads(bhmodel_serialized)
pred_medv = bhmodel_deserialized.predict(dfInputData)
dfOutputData = dfInputData
dfOutputData["pred_medv"] = pred_medv
'

    SET @InputDFName = 'dfInputData'
    SET @OutputDFName = 'dfOutputData'

    EXECUTE sp_execute_external_script
        @language = N'Python'
        ,@script = @PythonScript
        ,@input_data_1 = @Query
        ,@input_data_1_name = @InputDFName
        ,@output_data_1_name = @OutputDFName
        ,@params = N'@bhmodel_serialized varbinary(max)'
        ,@bhmodel_serialized = @model

    WITH RESULT SETS((
        [crim] float
        ,[rm] float
        ,[tax] float
        ,[lstat] float
```



```

    ,[pred_medv] float
  ));
END;

```

Рассмотрим подробно, что происходит в приведенной выше процедуре.

1. Определяются следующие переменные, необходимые для скрипта:
 - *@model* – переменная для хранения модели. Модель Python, хранящаяся в этой переменной, извлекается из таблицы *dbo.Models* при помощи запроса на T-SQL;
 - *@PythonScript* – в этой переменной мы будем хранить скрипт на языке Python для проведения оценки;
 - *@Query* – здесь будет храниться скрипт на языке запросов T-SQL для определения набора данных для оценки;
 - *@InputDFName* – в этой переменной мы будем хранить имя входного набора данных, к которому будет обращаться Python;
 - *@OutputDFName* – в этой переменной мы будем хранить имя выходного набора данных, к которому будет обращаться Python.
2. В переменную *@Query* записывается инструкция на языке запросов T-SQL для получения входного набора данных.
3. В переменную *@PythonScript* записывается скрипт на языке Python для выполнения прогноза. В этом скрипте выполняются следующие действия:
 - загружается библиотека *pickle* для взаимодействия с моделью;
 - вызывается функция *pickle.loads()*, которая десериализует модель, хранящуюся в переменной *bhmodel_serialized*, и присваивает результат переменной *bhmodel_deserialized*;
 - вызывается метод *predict()* объекта *bhmodel_deserialized* для выполнения прогноза на основании датафрейма *dfInputData*;

Примечание. Обратите внимание, что мы включили полный датафрейм, поскольку он содержит только столбцы, необходимые для формулы регрессии. В других ситуациях вам может понадобиться провести предварительные преобразования датафрейма перед тем, как передать его в скрипт Python, чтобы он отвечал требованиям формулы модели.

- создается датафрейм *dfOutputData* на основании датафрейма *dfInputData* с добавленным столбцом *pred_medv*.
4. Устанавливаются имена для входного и выходного датафреймов.
 5. Конфигурируется хранимая процедура *sp_execute_external_script*. По большей части входные параметры этой функции вам должны быть понятны, мы лишь остановимся чуть подробнее на параметрах *@params* и *@bhmodel_serialized*. Параметр *@params* позволяет вам определить дополнительные аргументы, необходимые для модели. В нашем случае мы определили параметр с именем *@bhmodel_serialized* с типом данных *varbinary(max)*. С его помощью мы передадим в скрипт Python модель и сделаем ее доступной для скрипта по имени без знака *@* – в данном случае это *bhmodel_serialized*. Значение параметра *@bhmodel_serialized* устанавливается в следующей строке.


6. Инструкция *WITH RESULT SETS* позволяет установить имена и типы данных для столбцов. Это поможет Power BI определиться с именами и типами колонок в модели данных.

Выполните полученный скрипт, чтобы создать хранимую процедуру в базе данных. Теперь вы можете проводить оценку данных путем вызова процедуры.

Шаг 6. Извлеките данные с прогнозами из SQL Server в Power BI

Для этого вам необходимо открыть Power BI, нажать на выпадающую кнопку **Получить данные** (GetData) и выбрать пункт **SQL Server**. Заполните поля **Сервер** (Server) и **База данных** (Database). Затем раскройте секцию **Расширенные параметры** (Advanced options) и введите следующий текст в поле **Инструкция SQL** (SQL statement): EXEC [dbo].[uspPredictHousePrices_Python]. Флажок **Включить столбцы отношений** (Include relationship columns) по умолчанию установлен, но нам он не нужен, так что снимите его. На рис. 10.4 показано, как должна выглядеть заполненная форма.


SQL Server database

Server 

DSVM2019

Database (optional)

BostonHousingData

Data Connectivity mode 

☒ Import

☐ DirectQuery

Advanced options

Command timeout in minutes (optional)

SQL statement (optional, requires database)

[dbo].[uspPredictHousePrices_Python]

☒ Include relationship columns

☐ Navigate using full hierarchy

☐ Enable SQL Server Failover support

OK Cancel

Рис. 10.4 ❖ Окно получения данных для запуска процедуры dbo.uspPredictedHousePrices_Python

Нажмите на кнопку **ОК** для добавления информации в модель данных Power BI.

Как и в примере с R, в данном случае не стоит обращать внимания на саму модель на основе простой формулы линейной регрессии. Важно научиться применять модель R для оценки данных в Power BI при помощи службы SSMLS. Проектирование моделей машинного обучения выходит за рамки этой книги. Но если вы знаете, как строить полноценные модели прогнозирования в Python, вы можете использовать приведенные в данном разделе шаги для применения их к моделям данных Power BI с использованием службы SSMLS.

Анализ тональности текста в Power BI с помощью R со службой SSMLS

В главе 9 мы писали скрипт на Python для проведения анализа тональности текста с использованием службы *Microsoft Cognitive Services*. Однако большинство администраторов БД не позволяют выполнять обращение к API непосредственно из баз данных, так что этот план во многих случаях будет нежизнеспособным. К счастью, встроенные в службу SSMLS модели позволяют производить анализ тональности текста без необходимости обращаться к API *Microsoft Cognitive Services*. В данном разделе мы посмотрим, как можно воспользоваться готовой моделью из состава SSMLS с помощью R.

Добавление готовых моделей R в SSMLS с помощью PowerShell

Шаг 1. Проверьте, установлены ли предварительно обученные модели

Сделать это можно, перейдя по следующему пути: C:\Program Files\Microsoft SQL Server\MSSQL15.MSSQLSERVER\R_SERVICES\library\MicrosoftML\mxLibs\x64.

В этой папке вы должны обнаружить следующие файлы:

- AlexNet_Updated.model;
- ImageNet1K_mean.xml;
- pretrained.model;
- ResNet_101_Updated.model;
- ResNet_18_Updated.model;
- ResNet_50_Updated.model.

Если эти файлы есть, можете пропустить оставшиеся шаги и сразу переходить к разделу *Использование готовой модели R в SSMLS для анализа тональности текста в Power BI*. В противном случае переходите к следующему шагу.

Шаг 2. Откройте PowerShell от имени администратора

Открыть PowerShell можно множеством способов. Простейший из них – через поиск на панели задач Windows. В результате вы увидите ярлык приложения *Windows PowerShell*. Щелкните по нему правой кнопкой мыши и выберите пункт **Запустить от имени администратора** (Run as administrator).

Шаг 3. Загрузите скрипт PowerShell

Перейдите по адресу <https://aka.ms/mlm4sql> и скачайте файл *Install-MLModels.ps1*. Этот файл содержит скрипт PowerShell, необходимый для загрузки моделей в SSMLS. Убедитесь, что файл скачался в папку загрузок, поскольку на следующем шаге мы будем предполагать, что он у вас уже есть.

Шаг 4. Запустите загруженный скрипт в PowerShell

Выполните следующую команду в PowerShell:

```
C:\Users\<имя пользователя>\Downloads\Install-MLModels.ps1 MSSQLSERVER
```

Если вы скачали файл в папку загрузок, просто поменяйте <имя пользователя> на собственную учетную запись. Обратитесь к разделу *Решение проблем*, если у вас не получается запустить код.

Решение проблем

1. Если вы не можете запустить скрипт, возможно, у вас нет для этого прав. Ознакомиться со своими правами можно, введя в PowerShell следующую команду:

```
Get-ExecutionPolicy
```

2. Если ваши права доступа *ограничены* (restricted), вы можете изменить их на *неограниченные* (unrestricted), выполнив следующую команду:

```
Set-ExecutionPolicy unrestricted
```

3. После запуска предыдущего кода у вас должны появиться права для запуска скрипта. Вернуть ограниченные права в PowerShell можно, запустив следующую инструкцию:

```
Set-ExecutionPolicy restricted
```

Использование готовой модели R в SSMLS для анализа тональности текста в Power BI

Если служба *SQL Server Machine Learning Services* запущена и *предварительно обученные модели* (pretrained model) успешно загружены, вы можете присту-

путь к выполнению анализа тональности текста. Ниже приведены действия, которые необходимо для этого выполнить.

Шаг 1. Определите хранимую процедуру

Ниже приведен код оболочки хранимой процедуры на языке запросов T-SQL:

```
CREATE PROCEDURE [dbo].[getSentiments_R]
AS
BEGIN
END
```

Инструкция *CREATE PROCEDURE* используется для создания хранимой процедуры в базе данных. Здесь мы добавляем процедуру с именем *[dbo].[getSentiments_R]*. Приставка *[dbo]* в имени процедуры указывает на схему, которой она принадлежит. Схемы предназначены для объединения объектов баз данных в группы. Схемой по умолчанию является как раз *dbo*. Далее следует имя процедуры и через ключевое слово *AS* – тело процедуры, состоящее из инструкций T-SQL. В данном примере код процедуры будет обрамлен ключевыми словами *BEGIN...END*, а значит, запускаться будет в виде одного пакета.

Шаг 2. Определите переменные

Ниже перечислены переменные, которые будут использоваться в нашей хранимой процедуре:

```
DECLARE @RScript nvarchar(max);
DECLARE @Query nvarchar(max);
DECLARE @InputDataFrame nvarchar(128) = 'dfInput';
DECLARE @OutputDataFrame nvarchar(128) = 'dfOutput';
```

В переменной *@RScript* мы будем хранить скрипт на языке R для анализа тональности текста, переменная *@Query* будет отвечать за инструкции на языке T-SQL для определения исходного набора данных, который будет передан на вход R. Имя, по которому R будет обращаться ко входному набору данных, поместим в переменную *@InputDataFrame*, а к выходному – в переменную *@OutputDataFrame*.

Шаг 3. Инициализируйте переменную @Query

Ниже приведен код для инициализации переменной *@Query*:

```
SET @Query = 'SELECT [id], [text] ' +
'FROM [dbo].[SentimentData]'
```

Здесь мы выстраиваем инструкции на языке T-SQL, извлекающие исходные данные для скрипта. Текст, который мы будем анализировать, содержится в поле *text*, а поле *id* служит для уникальной идентификации записей в наборе данных.

Шаг 4. Инициализируйте переменную @RScript

Ниже приведен код на языке R, выполняющий анализ тональности текста:

```
SET @RScript = '
dfInput$text = as.character(dfInput$text)
sentimentScores <-
  rxFeaturize(
    data = dfInput,
    mlTransforms = getSentiment(
      vars = list(SentimentScore = "text"))
  )
sentimentScores$text <- NULL
dfOutput <- cbind(dfInput, sentimentScores)'
```

Скрипт получился довольно коротким, поскольку всю основную работу выполняет предварительно обученная модель. Здесь мы сначала преобразуем поле *text* в датафрейме *dfInput* в символьный тип. По умолчанию R конвертирует все символьные поля в тип данных *factor*, о котором мы уже говорили ранее. Этот тип используется для представления категориальных данных.

Данные, хранящиеся в виде факторов, по своей структуре напоминают словари в табличном движке Microsoft. Каждый уникальный элемент в поле заменяется на целочисленное значение, которое служит в качестве индекса, и отдельно создается таблица соответствия между индексами и уникальными элементами, которые они заменили. Такая техника весьма эффективна, поскольку целочисленные значения занимают меньше места при хранении по сравнению с длинными строками, и, кроме того, работать с ними можно гораздо более эффективно. Факторы идеально подходят для анализа информации, но в данном случае нам не понадобятся все эти преимущества – нам нужен только текст, в связи с чем мы и преобразовали данные в символьный формат при помощи функции *as.character()*.

Анализ тональности текста выполняется в следующем фрагменте скрипта:

```
sentimentScores <-
  rxFeaturize(
    data = dfInput,
    mlTransforms = getSentiment(
      vars = list(SentimentScore = "text"))
  )
```

Здесь вся работа ложится на функции *rxFeaturize()* и *getSentiment()* из пакета *RevoScaleR*. Этот пакет предназначен для облегчения работы с большими наборами данных.

Примечание. Пакет *RevoScaleR* бывает очень полезен в работе, и я настоятельно рекомендую вам изучить его функционал, если вы планируете активно работать с языком R с использованием службы *SSMLS*. По данной ссылке располагается подробная инструкция по работе с этим пакетом: https://packages.revolutionanalytics.com/doc/8.0.0/win/RevoScaleR_Users_Guide.pdf.

Функция `rxFeaturize()` используется для доступа к данным, прошедшим обработку алгоритмами машинного обучения из пакета *MicrosoftML*. Преобразования данных определяются в аргументе `mlTransforms`. В нашем случае преобразование ограничивается выполнением анализа тональности текста при помощи функции `getSentiment()`. Аргумент `vars` в функции `getSentiment()` используется для указания полей в наборе данных, к которым будет применен анализ. Если в этом аргументе передать *именованный список* (named list), то имя каждого элемента в списке будет определять название столбца, в котором будет храниться итоговая оценка, а значение будет указывать на поле для анализа. Очевидным преимуществом использования именovanного списка является возможность проведения параллельного анализа тональности для нескольких полей в наборе данных. Ниже показано, как выглядел бы наш код, если бы нам нужно было проанализировать еще два поля: `textB` и `textC`:

```
sentimentScores <-
  rxFeaturize(
    data = dfInput,
    mlTransforms = getSentiment(
      vars = list(
        SentimentScore = "text",
        SentimentScoreB = "textB",
        SentimentScoreC = "textC")
      )
  )
```

Функция `getSentiment()` возвращает числовое значение между 0 и 1 для каждого анализируемого поля. Чем ближе это значение к нулю, тем негативнее контекст сообщения, а чем ближе к единице, тем он позитивнее. В каждом отдельном случае вам необходимо выработать собственные правила категоризации полученных результатов. Обычно результаты переводятся в двоичный формат с использованием определенной граничной отсечки, разделяющей негативные сообщения от позитивных. Чаще всего в качестве этой граничной отсечки используется значение 0,5. В этом случае все текстовые сообщения, оценка которых превышает эту отметку, считаются позитивными, а остальные – негативными.

Но вам не обязательно приходить к двоичному формату. Например, вы можете разбить ответы на три категории: позитивные, негативные и нейтральные. В этом случае граничными отсечками могут служить значения 0,33 и 0,66. В общем, способ категоризации выходных данных полностью зависит от вас.

Шаг 5. Сконфигурируйте процедуру `sp_execute_external_script`

Ниже приведен код для конфигурирования процедуры `sp_execute_external_script`:

```
EXEC sp_execute_external_script
  @language = N'R'
```

```
,@input_data_1 = @Query
,@input_data_1_name = @InputDFName
,@output_data_1_name = @OutputDFName
,@script = @RScript
```

Хранимая процедура *sp_execute_external_script* служит для запуска в *SQL Server* скрипта, написанного на языке, отличном от T-SQL. На момент написания книги эта процедура умела обрабатывать скрипты на языках R, Python и Java. Параметры процедуры зависят от языка, на котором написан скрипт. В данном сценарии мы определили пять следующих параметров:

- *@language* – используется для указания языка скрипта;
- *@input_data_1* – представляет код на языке T-SQL для создания входного набора данных;
- *@input_data_1_name* – имя, по которому R будет обращаться ко входному датафрейму;
- *@output_data_1_name* – имя, по которому R будет обращаться к выходному датафрейму;
- *@script* – скрипт на языке R, выполняющий анализ тональности текста.

Шаг 6. Определите выходные данные

Вывод определяется следующим образом:

```
WITH RESULT SETS((
    [crim] float
    ,[rm] float
    ,[tax] float
    ,[lstat] float
    ,[pred_medv] float
));
```



Без этого фрагмента кода *SQL Server* вернет выходные данные без имен и определенных типов. А это может стать проблемой при вызове хранимой процедуры из Power BI. Инструкция *WITH RESULT SETS* позволяет определить имена и типы данных для вывода, чтобы у клиентских программ наподобие Power BI не было проблем с его обработкой.

Шаг 7. Создайте хранимую процедуру в базе данных

Ниже приведен полный код хранимой процедуры с именем *getSentiments_R*:

```
CREATE PROCEDURE [dbo].[getSentiments_R]
AS
BEGIN
    DECLARE @RScript nvarchar(max);
    DECLARE @Query nvarchar(max);
    DECLARE @InputDFName nvarchar(128) = 'dfInput';
    DECLARE @OutputDFName nvarchar(128) = 'dfOutput';
```

```

SET @Query = 'SELECT [id], [text], [likes] ' +
'FROM [dbo].[SentimentData]'

SET @RScript = '
dfInput$text = as.character(dfInput$text)
sentimentScores <-
  rxFeaturize(
    data = dfInput,
    mlTransforms = getSentiment(
      vars = list(SentimentScore = "text"))
  )
sentimentScores$text <- NULL
dfOutput <- cbind(dfInput, sentimentScores)'

EXEC sp_execute_external_script
  @language = N'R'
  ,@input_data_1 = @Query
  ,@input_data_1_name = @InputDFName
  ,@output_data_1_name = @OutputDFName
  ,@script = @RScript

WITH RESULT SETS (
  (
    [id] [bigint],
    [text] [varchar](8000),
    [score] [float]
  )
)
END

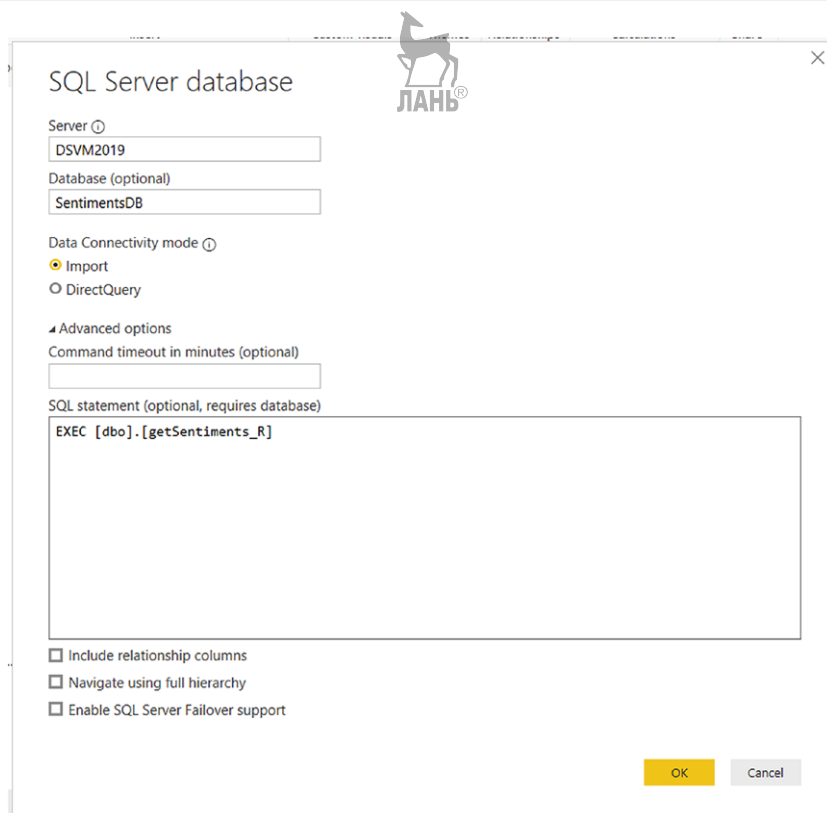
```

Запустите этот код в *SQL Server* для создания хранимой процедуры.

Шаг 8. Вызовите процедуру из Power BI

Откройте Power BI, нажмите на выпадающую кнопку **Получить данные** (Get-Data) и выберите пункт **SQL Server**. Заполните поля **Сервер** (Server) и **База данных** (Database). Затем раскройте секцию **Расширенные параметры** (Advanced options) и введите следующий текст в поле **Инструкция SQL** (SQL statement): EXEC [dbo].[getSentiments_R]. Флажок **Включить столбцы отношений** (Include relationship columns) по умолчанию установлен, но нам он не нужен, так что снимите его. На рис. 10.5 показано, как должна выглядеть заполненная форма.

Как и в случае с вызовом любой другой хранимой процедуры из Power BI, выходные данные будут доступны для загрузки в модель данных.

Рис. 10.5 ❖ Окно получения данных для запуска процедуры *getSentiments_R*

АНАЛИЗ ТОНАЛЬНОСТИ ТЕКСТА В POWER BI С ПОМОЩЬЮ PYTHON СО СЛУЖБой SSMLS

Как и в случае с R, здесь мы не сможем воспользоваться службой *Microsoft Cognitive Services* для выполнения анализа тональности текста в *SSMLS* по соображениям безопасности. Но встроенная в службу *SSMLS* модель позволит провести анализ без необходимости обращаться к API *Microsoft Cognitive Services*. В данном разделе мы посмотрим, как можно воспользоваться готовой моделью из состава *SSMLS* с помощью Python.

Добавление готовых моделей Python в SSMLS

Шаг 1. Проверьте, установлены ли предварительно обученные модели

Сделать это можно, перейдя по следующему пути: `C:\Program Files\Microsoft SQL Server\MSSQL15.MSSQLSERVER\PYTHON_SERVICES\Lib\site-packages\microsoftml\mxLibs`.

В этой папке вы должны обнаружить такие файлы:

- AlexNet_Updated.model;
- ImageNet1K_mean.xml;
- pretrained.model;
- ResNet_101_Updated.model;
- ResNet_18_Updated.model;
- ResNet_50_Updated.model.

Если эти файлы есть, можете пропустить оставшиеся шаги и сразу переходить к разделу *Использование готовой модели Python в SSMLS для анализа тональности текста в Power BI*. В противном случае переходите к следующему шагу.

Шаг 2. Откройте PowerShell от имени администратора

Открыть PowerShell можно множеством способов, и простейший из них – через поиск на панели задач Windows. В результате вы увидите приложение *Windows PowerShell*. Щелкните по нему правой кнопкой мыши и выберите пункт **Запустить от имени администратора** (Run as administrator).

Шаг 3. Загрузите скрипт PowerShell

Перейдите по адресу <https://aka.ms/mlm4sql> и скачайте файл *Install-MLModels.ps1*. Этот файл содержит скрипт PowerShell, необходимый для загрузки моделей в SSMLS. Убедитесь, что файл скачался в папку загрузок, поскольку на следующем шаге мы будем предполагать, что он лежит именно там.

Шаг 4. Запустите загруженный скрипт в PowerShell

Выполните следующую команду в PowerShell:

```
C:\Users\<имя пользователя>\Downloads\Install-MLModels.ps1 MSSQLSERVER
```

Если вы скачали файл в папку загрузок, просто поменяйте <имя пользователя> на собственную учетную запись. Обратитесь к разделу *Решение проблем*, если у вас не получается запустить код.

Решение проблем

1. Если вы не можете запустить скрипт, возможно, у вас нет для этого прав. Ознакомиться с правами доступа можно, введя в PowerShell следующую команду:

```
Get-ExecutionPolicy
```

2. Если ваши права *ограничены* (restricted), вы можете изменить их на *неограниченные* (unrestricted), выполнив следующую команду:

```
Set-ExecutionPolicy unrestricted
```

3. После запуска предыдущего кода у вас должны появиться права для запуска скрипта. Вернуть ограниченные права в PowerShell можно, запустив следующую команду:

```
Set-ExecutionPolicy restricted
```

Использование готовой модели Python в SSMLS для анализа тональности текста в Power BI

Если служба *SQL Server Machine Learning Services* запущена и предварительно обученные модели успешно загружены, вы можете приступить к выполнению анализа тональности текста. Ниже приведены действия, которые необходимо для этого выполнить.

Шаг 1. Определите хранимую процедуру

Ниже приведен код оболочки хранимой процедуры на языке запросов T-SQL:

```
CREATE PROCEDURE [dbo].[getSentiments_Python]
AS
BEGIN
END
```

Команда *CREATE PROCEDURE* используется для создания хранимой процедуры в базе данных. В данном случае мы добавляем процедуру с именем *[dbo].[getSentiments_Python]*. Приставка *[dbo]* в имени процедуры указывает на схему, которой она принадлежит. Схемы служат для группирования объектов баз данных. Схемой по умолчанию является *dbo*. Далее идет имя процедуры и через ключевое слово *AS* – тело процедуры, состоящее из инструкций T-SQL. В нашем примере код процедуры будет заключен в ключевые слова *BEGIN ... END*, а значит, запускаться будет в виде одного пакета.

Шаг 2. Определите переменные

Ниже перечислены переменные, которые будут использоваться в нашей хранимой процедуре:

```
DECLARE @PythonScript nvarchar(max);
DECLARE @Query nvarchar(max);
DECLARE @InputDataFrame nvarchar(128) = 'dfInput';
DECLARE @OutputDataFrame nvarchar(128) = 'dfOutput';
```

В переменной *@PythonScript* будет храниться скрипт на языке Python для анализа тональности текста, а переменная *@Query* будет содержать инструкции на языке T-SQL для определения исходного набора данных, который будет передан на вход Python. Имя, по которому Python будет обращаться ко входному набору данных, поместим в переменную *@InputDataFrame*, а к выходному – в переменную *@OutputDataFrame*.

Шаг 3. Инициализируйте переменную @Query

Ниже приведен код для инициализации переменной *@Query*:

```
SET @Query = 'SELECT [id], [text] FROM [dbo].[SentimentData]'
```

Это инструкция на языке T-SQL, извлекающая исходные данные для скрипта.

Шаг 4. Инициализируйте переменную @PythonScript

Ниже приведен код на языке Python, выполняющий анализ тональности текста:

```
SET @PythonScript = '
from microsoftml import rx_featurize, get_sentiment
sentiment_scores = rx_featurize(
    data=dfInput,
    ml_transforms=[get_sentiment(cols=dict(scores="text"))])
dfOutput = sentiment_scores'
```

В первой строке приведенного кода импортируются необходимые нам функции. Это функции *rx_featurize()* и *get_sentiment()* из библиотеки *microsoftml*. Далее мы вызываем эти функции совместно для проведения анализа тональности текста. Функция *rx_featurize()* используется для доступа к данным, прошедшим обработку алгоритмами машинного обучения из пакета *microsoftml*. Преобразования данных определяются в аргументе *ml_transforms*. В нашем случае преобразование ограничивается выполнением анализа тональности текста при помощи функции *get_sentiment()*. Аргумент *cols* в функции *get_sentiment()* применяется для указания столбцов в наборе данных, которые должны быть проанализированы. Здесь используется словарь с набором ключей и значений, при этом ключи определяют имена новых столбцов с оценочными данными, а значения – имена столбцов, которые должны быть проанализированы.

Обратите внимание, что вы можете выполнять анализ одновременно для нескольких полей в наборе данных. Для этого достаточно добавить необходимое количество пар ключ/значение в словарь, передаваемый в качестве аргумента *cols*.

Функция *get_sentiment()* возвращает числовое значение между 0 и 1 для каждого анализируемого поля. Чем ближе это значение к нулю, тем негативнее контекст сообщения, а чем ближе к единице, тем он позитивнее.

Шаг 5. Сконфигурируйте процедуру *sp_execute_external_script*

Ниже приведен код для конфигурирования процедуры *sp_execute_external_script*:

```
EXEC sp_execute_external_script
    @language = N'Python'
    ,@input_data_1 = @Query
    ,@input_data_1_name = @InputDFName
    ,@output_data_1_name = @OutputDFName
    ,@script = @PythonScript
```

Хранимая процедура *sp_execute_external_script* служит для запуска в *SQL Server* скрипта, написанного на языке, отличном от T-SQL. Конфигурирование этой функции почти не отличается от примера с R. Единственное отличие состоит в передаваемом значении в параметре *@language*.

Шаг 6. Определите выходные данные

Вывод определяется следующим образом:

```
WITH RESULT SETS((
    [crim] float
    ,[rm] float
    ,[tax] float
    ,[lstat] float
    ,[pred_medv] float
));
```

Без этого фрагмента *SQL Server* вернет выходные данные без имен и типов. А это может быть неудобно при вызове хранимой процедуры из Power BI. Инструкция *WITH RESULT SETS* позволяет определить имена и типы данных для вывода, чтобы у клиентских программ вроде Power BI не было проблем с его обработкой.

Шаг 7. Создайте хранимую процедуру в базе данных

Ниже приведен полный код хранимой процедуры:

```
CREATE PROCEDURE [dbo].[getSentiments_Python]
AS
BEGIN
```

```

DECLARE @PythonScript nvarchar(max);
DECLARE @Query nvarchar(max);
DECLARE @InputDFName nvarchar(128) = 'dfInput';
DECLARE @OutputDFName nvarchar(128) = 'dfOutput';

SET @Query = 'SELECT ID, [text] FROM dbo.SentimentData'

SET @PythonScript = '
import pandas as pd
from microsoftml import rx_featurize, get_sentiment
sentiment_scores = rx_featurize(
    data=dfInput,
    ml_transforms=[get_sentiment(cols=dict(scores="text"))])
dfOutput = sentiment_scores
'

EXEC sp_execute_external_script
    @language = N'Python'
    ,@input_data_1 = @Query
    ,@input_data_1_name = @InputDFName
    ,@output_data_1_name = @OutputDFName
    ,@script = @PythonScript

WITH RESULT SETS (
    (
        [ID] [bigint],
        [text] [varchar](8000),
        [score] [float]
    )
)
END

```

Запустите этот код в *SQL Server* для создания хранимой процедуры *[dbo].[getSentiments_Python]*.

Шаг 8. Вызовите процедуру из Power BI

Откройте Power BI, в выпадающей кнопке **Получить данные** (GetData) выберите пункт **SQL Server**. Заполните поля **Сервер** (Server) и **База данных** (Database). После этого раскройте секцию **Расширенные параметры** (Advanced options) и введите следующий текст в поле **Инструкция SQL** (SQL statement): EXEC [dbo].[getSentiments_Python]. Флажок **Включить столбцы отношений** (Include relationship columns) по умолчанию установлен, но нам он не нужен, так что снимите его. На рис. 10.6 показано, как должна выглядеть заполненная форма.

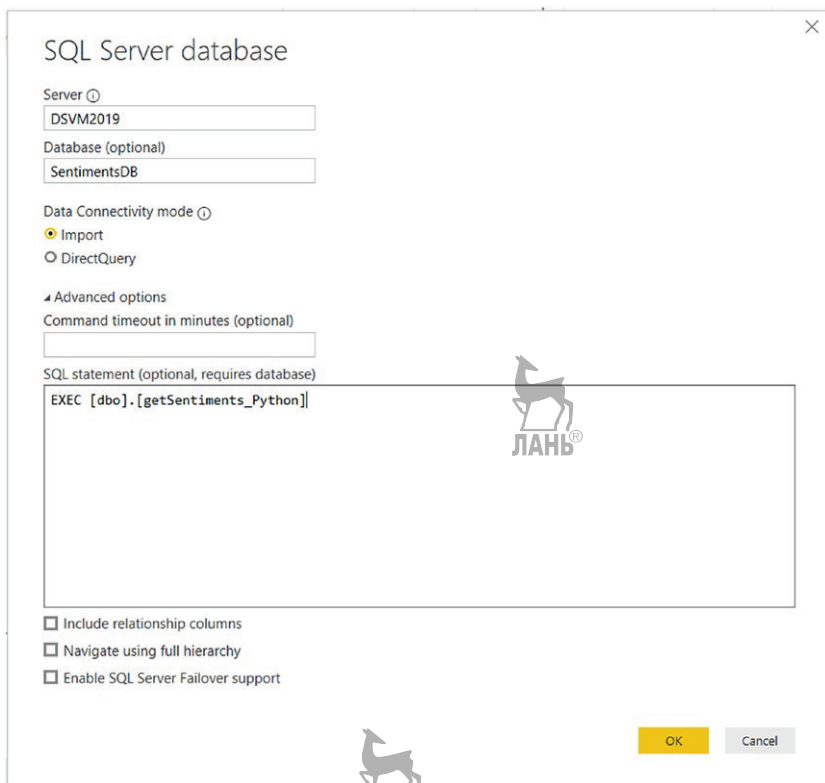


Рис. 10.6 ❖ Окно получения данных для запуска процедуры *getSentiments_Python*

Как и в случае с вызовом любой другой хранимой процедуры из Power BI, выходные данные будут доступны для импорта в модель данных.

ВЫЧИСЛЕНИЕ РАССТОЯНИЯ МЕЖДУ ТОЧКАМИ В POWER BI С ПОМОЩЬЮ R СО СЛУЖБой SSMLS

Преимущества доступа из *SQL Server* к языку R не ограничиваются возможностью оценки данных при помощи моделей машинного обучения. В частности, вы можете выполнять полный спектр преобразования данных и осуществлять сложнейшие математические расчеты, которые непросто реализовать посредством языка запросов T-SQL. В данном разделе мы приведем пример проведения математических вычислений с использованием языка R. Как мы уже делали ранее в этой книге, рассчитаем расстояние между двумя точками на земной поверхности с помощью *формулы гаверсина* (Haversine formula).

Давайте пройдемся по шагам!

Шаг 1. Убедитесь, что в SSMLS загружен пакет dplyr

Пакет *dplyr* будет доступен вам, если вы используете *DSVM*. В противном случае вам придется обратиться к инструкции во введении к данной книге, чтобы узнать, как загрузить пакеты R в службу *SSMLS*. Вы можете запустить скрипт, показанный ниже, для отображения пакетов R, доступных в *SSMLS*:

```
EXECUTE sp_execute_external_script
    @language=N'R',
    @script = N'
packagematrix <- installed.packages();
Name <- packagematrix[,1];
Version <- packagematrix[,3];
OutputDataSet <- data.frame(Name, Version);'

WITH RESULT SETS ((PackageName nvarchar(250), PackageVersion nvarchar(max)) )
```

Шаг 2. Запустите SSMS и подключитесь к SQL Server

Убедитесь, что вы подключены к *SQL Server* с установленной службой *SSMLS* и поддержкой R. Все это будет доступно по умолчанию при использовании виртуальной машины *DSVM*. Если вы настраивали свой экземпляр сервера без *DSVM*, вам придется самостоятельно позаботиться о том, чтобы все необходимое было установлено. Обратитесь к инструкции во вводной главе к этой книге, чтобы узнать, как включить поддержку R в *SSMLS*, если в вашем *SQL Server* она не активирована. Узнать, есть ли такая поддержка, можно, запустив следующий скрипт:

```
EXEC sp_execute_external_script @language = N'R',
@script = N'
Test <- "Test R"
OutputDataSet = data.frame(Test)'
```

Если поддержка языка R присутствует, будет возвращена таблица с единственной ячейкой с текстом «Test R».

Шаг 3. Добавьте базу данных CalculateDistance на ваш сервер

База данных *CalculateDistance* полностью сконфигурирована под требования данного примера. Ее резервную копию вы найдете в репозитории к этой главе – в папке *Databases*. Выполните следующие шаги, чтобы восстановить базу данных из резервной копии в *SQL Server*.

1. Скопируйте файл *CalculateDistance.bak* в папку *Backup* вашего *SQL Server*. В виртуальной машине путь будет следующим: *C:\Program Files\Microsoft SQL Server\MSSQL15.MSSQLSERVER\MSSQL\Backup*.

2. Откройте *SQL Server Management Studio (SSMS)*, щелкните правой кнопкой мыши по пункту **Базы данных** (Databases) и выберите пункт **Восстановить базу данных** (Restore Database).
3. В открывшемся окне **Восстановление базы данных** (Restore Database) на вкладке **Общие** (General) установите переключатель **Источник** (Source) в положение **Устройство** (Device).
4. Щелкните по кнопке с тремя точками и в открывшемся диалоговом окне нажмите на кнопку **Добавить** (Add), после чего выберите путь к файлу *CalculateDistance.bak*.
5. Выберите файл *CalculateDistance.bak* и нажмите на кнопку **ОК**, чтобы закрыть окно **Выбор устройств резервного копирования** (Locate Backup File).
6. Нажмите на кнопку **ОК**, что приведет к добавлению новой базы данных в список.

Шаг 4. Создайте хранимую процедуру для расчета расстояний

Ниже приведен текст хранимой процедуры, вычисляющей расстояние между двумя точками на земной поверхности по их координатам:

```
CREATE PROCEDURE [dbo].[calcDistance_R]
AS
BEGIN
    DECLARE @RScript nvarchar(max);
    DECLARE @Query nvarchar(max);
    DECLARE @InputDFName nvarchar(128) = 'dfInput';
    DECLARE @OutputDFName nvarchar(128) = 'dfOutput';

    SET @Query = 'SELECT [Employee_ID], [EmployeeAddress],
        [TerminalAddress], [lon_EmployeeAddress], [lat_EmployeeAddress],
        [lon_TerminalAddress], [lat_TerminalAddress]
        FROM [dbo].[EmployeeList]';

    SET @RScript = '
library(dplyr)
ComputeDist <-
function(addressA_long, addressA_lat, addressB_long,
addressB_lat) {
    R <- 6371 / 1.609344 #radius in mile
    delta_lat <- addressB_lat - addressA_lat
    delta_long <- addressB_long - addressA_long
    degrees_to_radians = pi / 180.0
    a1 <- sin(delta_lat / 2 * degrees_to_radians)
    a2 <- as.numeric(a1) ^ 2
    a3 <- cos(addressA_lat * degrees_to_radians)
    a4 <- cos(addressB_lat * degrees_to_radians)
    a5 <- sin(delta_long / 2 * degrees_to_radians)
    a6 <- as.numeric(a5) ^ 2

```

```

    a <- a2 + a3 * a4 * a6
    c <- 2 * atan2(sqrt(a), sqrt(1 - a))
    d <- R * c
    return(d)
}

dfOutput <-
dfInput %>%
mutate(
  Distance =
    round(ComputeDist(lon_EmployeeAddress,
      lat_EmployeeAddress, lon_TerminalAddress,
      lat_TerminalAddress), 1)
)'

EXEC sp_execute_external_script
  @language = N'R'
  ,@input_data_1 = @Query
  ,@input_data_1_name = @InputDFName
  ,@output_data_1_name = @OutputDFName
  ,@script = @RScript

WITH RESULT SETS (
  (
    [ID] [bigint],
    [AddressA] [varchar](50),
    [AddressB] [varchar](50),
    [lon_AddressA] [decimal](10,8),
    [lat_AddressA] [decimal](10,8),
    [lon_AddressB] [decimal](10,8),
    [lat_AddressB] [decimal](10,8),
    [Distance] [decimal](4,1)
  )
)

END

```



В этом скрипте используется та же логика, что и в главе 8. Разница состоит в том, что данное решение может быть реализовано в масштабах компании без необходимости ограничиваться персональным режимом локального шлюза данных в Power BI. Теперь вы можете использовать стандартный режим шлюза, поскольку все скрипты R будут аккуратно упакованы в хранимые процедуры в *SQL Server*.

Вы уже понимаете все преимущества написания хранимых процедур со встроенным в них кодом на языке R, так что давайте просто пройдемся по шагам.

1. Определяются переменные, необходимые для скрипта:
 - *@RScript* – в этой переменной мы будем хранить скрипт на языке R для проведения расчетов;
 - *@Query* – здесь будет храниться скрипт на языке запросов T-SQL для определения набора данных;

Примечание. Одним из преимуществ генерирования входного набора данных при помощи T-SQL является то, что вы можете использовать всю мощь этого языка запросов для реализации бизнес-логики или выполнения сложных преобразований данных при определении набора.

- *@InputDFName* – в этой переменной мы будем хранить имя входного набора данных, к которому будет обращаться R;
 - *@OutputDFName* – в этой переменной мы будем хранить имя выходного набора данных, к которому будет обращаться R;
2. В переменную *@Query* записывается инструкция на языке запросов T-SQL для получения входного набора данных.
 3. В переменной *@RScript* хранится скрипт на языке R для выполнения вычислений. В этом скрипте выполняются следующие действия:
 - загружается пакет *dplyr*. Этот пакет необходим для добавления столбцов к набору данных с вычисленным расстоянием между точками;
 - определяется функция *ComputeDist()*. По сути, это та же функция из главы 8. Мы просто используем скрипт на языке R в хранимой процедуре вместо вызова из Power Query;
 - создается датафрейм *dfOutput* на основании данных из входного датафрейма *dfInput*, переданного в скрипт R, после чего к нему добавляется столбец с именем *Distance*, вычисленный при помощи функции *ComputeDist()*;
 - конфигурируется хранимая процедура *sp_execute_external_script* с использованием ранее определенных переменных;
 - запускается инструкция *WITH RESULT SETS* для явной установки имен и типов данных столбцов на выходе.

Эти шаги были подробно описаны в главе 8. Обратитесь к этому описанию, если вам нужна дополнительная информация. Запустите скрипт на языке T-SQL в SSMS для создания хранимой процедуры *dbo.calcDistance_R* в базе данных.

Шаг 5. Вызовите процедуру из Power BI

Для этого вам необходимо открыть Power BI, нажать на выпадающую кнопку **Получить данные** (GetData) и выбрать пункт **SQL Server**. Заполните поля **Сервер** (Server) и **База данных** (Database). После этого раскройте секцию **Расширенные параметры** (Advanced options) и введите следующий текст в поле **Инструкция SQL** (SQL statement): `EXEC [dbo].[calcDistance_R]`. Флажок **Включить столбцы отношений** (Include relationship columns) по умолчанию установлен, но нам он не нужен, так что снимите его. На рис. 10.7 показано, как должна выглядеть заполненная форма.

Результирующий набор данных будет доступен для загрузки в модель данных Power BI.

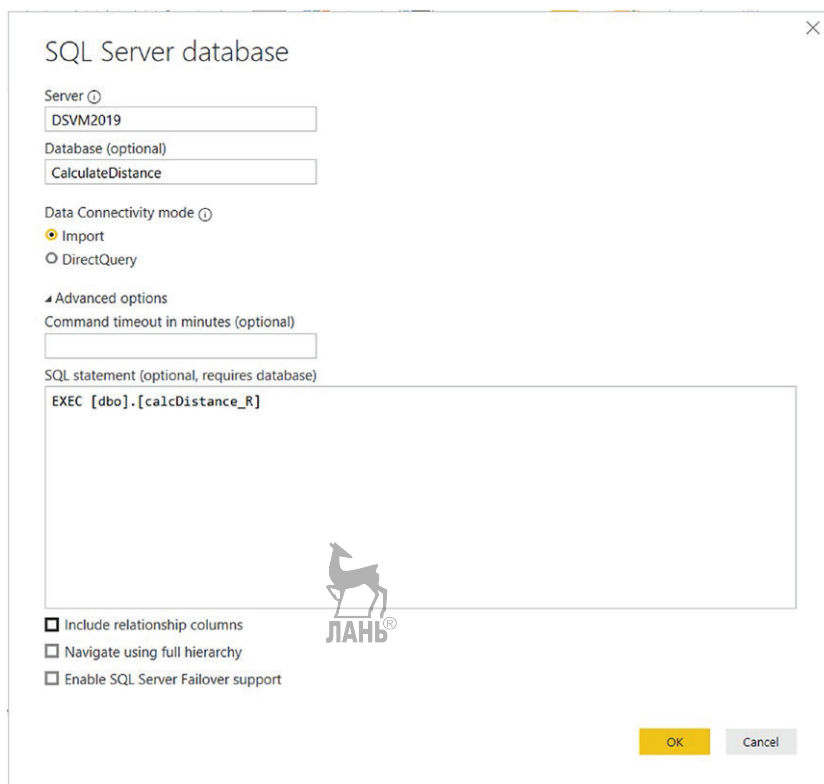


Рис. 10.7 ❖ Окно получения данных для запуска процедуры calcDistance_R

ВЫЧИСЛЕНИЕ РАССТОЯНИЯ МЕЖДУ ТОЧКАМИ В POWER BI С ПОМОЩЬЮ PYTHON СО СЛУЖБой SSMLS

Как и в случае с R, очень полезно иметь доступ к языку Python непосредственно из *SQL Server*. Это позволяет выполнять полный спектр преобразования данных и осуществлять сложные математические расчеты, которые непросто реализовать посредством языка запросов T-SQL. В предыдущем разделе мы произвели рефакторинг относительно сложного скрипта на языке R с использованием формулы гаверсина для расчета расстояний между двумя точками на земной поверхности. Теперь сделаем то же самое с помощью языка Python.

Шаг 1. Запустите SSMS и подключитесь к SQL Server

Убедитесь, что вы подключены к *SQL Server* с установленной службой *SSMS* и поддержкой Python. Все это будет доступно при использовании *DSVM*. Если же вы настраивали свой экземпляр сервера самостоятельно, вам придется позаботиться о том, чтобы все необходимое было установлено. Узнать, есть ли поддержка Python в вашем экземпляре, можно, запустив следующий скрипт:

```
EXEC sp_execute_external_script @language =N'Python',
@script=N'
OutputDataSet = InputDataSet;
',
@input_data_1 =N'SELECT 1 AS hello'
WITH RESULT SETS ([[hello] int not null]);
GO
```



Если поддержка языка Python есть, будет возвращена таблица с единственной колонкой и строкой. Столбец будет называться *hello*, а в ячейке будет содержаться значение 1.

Шаг 2. Добавьте базу данных CalculateDistance на ваш сервер

Резервную копию базы данных *CalculateDistance* вы найдете в репозитории к этой главе – в папке *Databases*. Выполните следующие шаги, чтобы восстановить базу данных из резервной копии в *SQL Server*.

1. Скопируйте файл *CalculateDistance.bak* в папку *Backup* вашего *SQL Server*. В виртуальной машине путь будет следующим: *C:\Program Files\Microsoft SQL Server\MSSQL15.MSSQLSERVER\MSSQL\Backup*.
2. Откройте *SQL Server Management Studio (SSMS)*, щелкните правой кнопкой мыши по пункту **Базы данных** (Databases) и выберите пункт **Восстановить базу данных** (Restore Database).
3. В открывшемся окне **Восстановление базы данных** (Restore Database) на вкладке **Общие** (General) установите переключатель **Источник** (Source) в положение **Устройство** (Device).
4. Щелкните по кнопке с тремя точками и в открывшемся диалоговом окне нажмите на кнопку **Добавить** (Add), после чего выберите путь к файлу *CalculateDistance.bak*.
5. Выберите файл *CalculateDistance.bak* и нажмите на кнопку **ОК**, чтобы закрыть окно **Выбор устройств резервного копирования** (Locate Backup File).
6. Нажмите на кнопку **ОК**, что приведет к добавлению новой базы данных в список.

Шаг 3. Создайте хранимую процедуру для расчета расстояний



Ниже показан текст хранимой процедуры, вычисляющей расстояние между двумя точками по их координатам:

```
CREATE PROCEDURE [dbo].[calcDistance_Python]
AS
BEGIN
    DECLARE @PythonScript nvarchar(max);
    DECLARE @Query nvarchar(max);
    DECLARE @InputDFName nvarchar(128) = 'dfInput';
    DECLARE @OutputDFName nvarchar(128) = 'dfOutput';

    SET @Query = 'SELECT [Employee_ID], [EmployeeAddress],
        [TerminalAddress], [lon_EmployeeAddress], [lat_EmployeeAddress],
        [lon_TerminalAddress], [lat_TerminalAddress]
        FROM [dbo].[EmployeeList]';

    SET @PythonScript = '
from math import cos, sin, atan2, pi, sqrt, pow

def ComputeDist(row):
    R = 6371 / 1.609344 #radius in mile
    delta_lat = row["lat_EmployeeAddress"] - row["lat_TerminalAddress"]
    delta_lon = row["lon_EmployeeAddress"] - row["lon_TerminalAddress"]
    degrees_to_radians = pi / 180.0
    a1 = sin(delta_lat / 2 * degrees_to_radians)
    a2 = pow(a1,2)
    a3 = cos(row["lat_TerminalAddress"] * degrees_to_radians)
    a4 = cos(row["lat_EmployeeAddress"] * degrees_to_radians)
    a5 = sin(delta_lon / 2 * degrees_to_radians)
    a6 = pow(a5,2)
    a = a2 + a3 * a4 * a6
    c = 2 * atan2(sqrt(a), sqrt(1 - a))
    d = R * c

    return d

dfOutput = dfInput

dfOutput["Distance"] = dfOutput.apply(lambda row: ComputeDist(row), axis=1)'

EXEC sp_execute_external_script
    @language = N'Python'
    ,@input_data_1 = @Query
    ,@input_data_1_name = @InputDFName
    ,@output_data_1_name = @OutputDFName
    ,@script = @PythonScript

WITH RESULT SETS (
    (
        [ID] [bigint],
```

```

[AddressA] [varchar](50),
[AddressB] [varchar](50),
[lon_AddressA] [FLOAT],
[lat_AddressA] [FLOAT],
[lon_AddressB] [FLOAT],
[lat_AddressB] [FLOAT],
[Distance] [FLOAT]
)
)
END

```

В этом скрипте используется практически та же логика, что и в главе 8. Разница лишь в том, что данное решение может быть реализовано в масштабах компании без необходимости ограничиваться персональным режимом локального шлюза данных в Power BI. Давайте, как и раньше, пройдемся по шагам в этой процедуре.

1. Определяются переменные, необходимые для скрипта:
 - *@PythonScript* – в этой переменной мы будем хранить скрипт на языке Python для проведения расчетов;
 - *@Query* – здесь будет храниться скрипт на языке запросов T-SQL для определения набора данных;
 - *@InputDFName* – в этой переменной мы будем хранить имя входного набора данных, к которому будет обращаться Python;
 - *@OutputDFName* – в этой переменной мы будем хранить имя выходного набора данных, к которому будет обращаться Python;
2. В переменную *@Query* записывается инструкция на языке запросов T-SQL для получения входного набора данных.
3. В переменной *@PythonScript* хранится скрипт на языке Python для выполнения вычислений. Здесь выполняются следующие действия:
 - загружаются функции *cos*, *sin*, *atan2*, *pi*, *sqr* и *pow* из библиотеки *math*. Именно они понадобятся нам для вычисления расстояния между точками;
 - определяется функция *ComputeDist()*. По сути, это та же функция из главы 8. Мы просто используем ее в хранимой процедуре вместо вызова из Power Query;
 - создается датафрейм *dfOutput* на основании данных из входного датафрейма *dfInput*, переданного в скрипт, после чего к нему добавляется столбец с именем *Distance*, вычисленный при помощи лямбда-функции, применяющей функцию *ComputeDist()*;
4. Конфигурируется хранимая процедура *sp_execute_external_script* с использованием ранее определенных переменных. Заметьте, что они отличаются от примера с оценкой цен на недвижимость.
5. Запускается инструкция *WITH RESULT SETS* для явной установки имен и типов данных столбцов на выходе.

Эти шаги были подробно описаны в главе 8. Обратитесь к данному описанию, если вам нужна дополнительная информация. Запустите скрипт на языке T-SQL в SSMS для создания хранимой процедуры *dbo.calcDistance_Python* в базе данных.

Шаг 4. Вызовите процедуру из Power BI

Откройте Power BI, нажмите на выпадающую кнопку **Получить данные** (Get-Data) и выберите пункт **SQL Server**. Заполните поля **Сервер** (Server) и **База данных** (Database). После этого раскройте секцию **Расширенные параметры** (Advanced options) и введите следующий текст в поле **Инструкция SQL** (SQL statement): `EXEC [dbo].[calcDistance_Python]`. Флажок **Включить столбцы отношений** (Include relationship columns) по умолчанию установлен, но нам он не нужен, так что снимите его. На рис. 10.8 показано, как должна выглядеть заполненная форма.

SQL Server database

Server ⓘ
DSVM

Database (optional)
CalculateDistance

Data Connectivity mode ⓘ
☒ Import
☐ DirectQuery

Advanced options

Command timeout in minutes (optional)

SQL statement (optional, requires database)
 EXEC [dbo].[calcDistance_Python]

☒ Include relationship columns
☐ Navigate using full hierarchy
☐ Enable SQL Server Failover support

OK Cancel

Рис. 10.8 ❖ Окно получения данных для запуска процедуры `dbo.calcDistance_Python`

Результирующий набор данных будет доступен для загрузки в модель данных Power BI.

В данной главе вы научились использовать скрипты на языках R и Python совместно с Power BI и службой *SSMLS*. При этом скрипты будут доступны в Power BI, поскольку они были заключены в специальные хранимые процедуры в *SQL Server*. Разумеется, службы *SSMLS* здесь мы коснулись довольно

поверхностно, поскольку для ее подробного описания нужно писать отдельную книгу. В состав *SSMLS* входят инструменты, позволяющие гораздо более эффективно работать с большими объемами данных, чем было показано в данной главе. Их применение ориентировано на инженеров и аналитиков данных. Лично я рекомендовал бы вам самостоятельно ознакомиться со следующими инструментами из состава службы *SSMLS*:

- *Native Scoring*: позволяет выполнять прогнозирование при помощи представлений T-SQL с использованием обученной модели, разработанной с участием алгоритмов *revoscalepy* или *RevoScaleR*. Механизм *Native Scoring* работает быстрее, чем пользовательские модели R и Python, поскольку использует нативные библиотеки C++ и не обращается к интерпретатору R или Python;
- *RevoScaleR*: коллекция функций R для загрузки, преобразования и анализа данных с возможностью масштабирования. Подробное описание пакета *RevoScaleR* можно найти по адресу https://packages.revolutionanalytics.com/doc/8.0.0/win/RevoScaleR_Users_Guide.pdf;
- *revoscalepy*: подборка функций Python для загрузки, преобразования и анализа данных с возможностью масштабирования. Подробное описание пакета *revoscalepy* можно найти по адресу <https://docs.microsoft.com/en-us/sql/machine-learning/python/ref-py-revoscalepy?view=sql-server-ver15>;
- *Real Time Scoring*: движок *Real Time Scoring* использует системную хранимую процедуру *sp_rxPredict* для выполнения прогнозирования. При этом он не зависит от R или Python и может быть запущен в экземпляре *SQL Server* без установки этих языков.

Методы, описанные ранее в данной главе, хорошо работают с относительно небольшими наборами данных в *SQL Server*, которые могут без проблем поместиться в памяти сервера. Что касается перечисленных выше инструментов, они могут пригодиться в случаях, когда объемы обрабатываемых данных слишком велики для того, чтобы полностью уместиться в памяти.

В дополнение к *SSMLS* компания Microsoft предлагает облачные решения корпоративного уровня, которыми вы можете воспользоваться из Power BI. Среди этих решений можно выделить *Azure Machine Learning Services*, *Azure Databricks* и *Azure Synapse*. Еще больше инструментов будет доступно при наличии версии Power BI Premium. Мы в данной книге коснулись только службы *SSMLS* по причине того, что это решение бесплатно доступно всем, у кого установлен *SQL Server 2016* или выше, и не требует приобретения дополнительных опций. Кто знает, возможно, облачных сервисов мы коснемся в следующем издании этой книги!

Я действительно очень рад, что вы приобрели и прочитали мою книгу. Надеюсь, ее содержимое показалось вам полезным и познавательным!

Предметный указатель

Символы

<-, 161
%m+%, 164

A

aes(), 48
AI Insights, 269
all.equal(), 74
alpha, 144
annotate(), 81, 144
API данных, 212
append(), 189, 282
apply(), 232, 236
as.character(), 314
as_data_frame(), 199
ast, 280
axis.title, 155
axis.title.x, 147
axis.title.y, 147

B

Batch result item, 281
boundary(), 234
bquote(), 145

C

cbind(), 117, 272
censusdata, 218
censusdata.censusgeo(), 220
chdir(), 168
CognitiveServicesCredentials, 280
color, 49
comma_format(), 21
compile(), 241
concat(), 172, 187
coord_flip(), 81
create_engine(), 205
cursor, 306

D

data_frame(), 163
data.frame(), 163
DataFrame(), 282
dense_rank(), 75
devtools, 30
distHaversine(), 264
dollar_format(), 52
dplyr, 47
drop(), 258

E

element_rect(), 147
element_text(), 55, 87, 147

ExcelFile, 186
excel_sheets(), 181
execute(), 306

F

factor, 75
factor(), 129
fct_reorder(), 75
fill, 78, 84, 130
filter(), 76, 176
for, 188
from_dict(), 207
fullmatch(), 241

G

geocode(), 257
geocoders, 257
geom_bar(), 58, 78
geom_col(), 78
geom_histogram(), 60
geom_hline(), 142
geom_label_repel(), 141
geom_line(), 62
geom_point(), 49
geom_polygon(), 130
geom_smooth(), 153
geom_text(), 79
geom_text_repel(), 96
geom_vline(), 142
geopy, 257
geosphere, 263
get_acs(), 215
getSentiment(), 315
ggmap, 255
ggplot(), 48
ggrepel, 67
ggthemes, 66
ggtitle(), 98, 122, 133, 154
group, 130
group(), 241

H

hjust, 55

I

ifelse(), 75
install.packages(), 30
isTRUE(), 74
iterrows(), 288

J

joblib, 273
joblib.dump(), 273



joblib.load(), 273

L

label, 78
labs(), 50, 83, 145
lambda, 257
len(), 236
listdir(), 169, 188
list.files(), 161, 184
literal_eval(), 281
load_variables(), 214
loc, 275
location.Location, 257

M

map(), 172
map_df(), 165
map_dfr(), 183
math, 260
max(), 77, 164, 170
merge(), 291
microsoftl, 321
min(), 77
mutate(), 75, 129
mutate_geocode(), 255

N

Native Scoring, 334
now(), 198, 206
nrow(), 198
ntile(), 129
nudge_y, 79

O

odbcClose(), 297
odbcConnect(), 198
odbcDriverConnection(), 296
open(), 306
os, 168
os.chdir(), 177

P

pandas, 168
panel.border, 147
paste(), 77
paste0(), 93, 104, 197, 297
percent_format(), 121
pickle, 306
pickle.dumps(), 306
pickle.load(), 306
plot.caption, 55
plot.subtitle, 55, 87, 147, 155
plot.title, 55, 87, 147, 155
predict(), 272
preprocessing, 273, 274
Prophet, 100
purrr, 159
pyodbc, 306

R

read_csv(), 165, 172
readr, 159

readRDS(), 296
read_sql_table(), 206
readxl, 180
Real Time Scoring, 334
regex, 224
register_google(), 255
relativedelta(), 168, 170
rename(), 216
rep(), 116
rescale(), 141
reset_index(), 221
re.sub(), 231
return, 183
revoscalepy, 334
RevoScaleR, 314, 334
rle(), 116
RODBC, 197, 296
round(), 259
rowwise(), 264
rxFeaturize(), 315

S

sample(), 279
scale, 50
scale(), 274
scale_color_manual(), 97
scale_fill_manual(), 97, 121, 131
scales, 47, 67
scale_x_continuous(), 51, 147
scale_y_continuous(), 51, 147
scikit-learn, 273
search(), 218
seq_along(), 116
serialize(), 297
Series, 170
set_names(), 182
setwd(), 160
shape, 206
sheet_names, 186
show.legend, 131, 142
sp_execute_external_script, 301, 309, 316, 322
split(), 236
sqlalchemy, 205
sqlQuery(), 198, 297
sqlSave(), 199
SQL Server Machine Learning Services, 294
stat_identity, 78
str(), 163
str_detect(), 176
strftime(), 206
str_replace(), 162, 227
str_sub(), 93
str_to_title(), 128
str_wrap(), 78
suppressMessages(), 215

T

TextAnalyticsClient, 280
theme(), 54, 86, 147
theme_economist(), 122
theme_map(), 133



theme_minimal(), 53
 theme_tufte(), 139
 tibble, 163
 tidycensus, 213
 tidyverse, 47, 66
 to_json(), 281
 tone_chat(), 289
 to_sql(), 207
 transmute(), 117

U

unique(), 72
 us, 220

V

View(), 214

W

which(), 77

X

xintercept, 142
 xlab(), 84, 145
 xmax, 144
 xmin, 144

Y

yintercept, 142
 ylab(), 84, 145
 ymax, 144
 ymd(), 162
 ymin, 144

A

Анализ тональности текста, 276

B

Векторизованная функция, 264
 Виртуальная машина для анализа данных, 277

Г

Географическая пространственная аналитика, 123
 Геокодирование, 251
 Геометрии, 48
 индивидуальные геометрии, 130
 коллективные геометрии, 130
 Горизонтальная столбчатая диаграмма, 69

Д

Датафрейм, 72, 163
 Диаграмма квадрантов, 134
 Доверительная вероятность, 149
 Доверительный интервал, 154

И

Именованный символьный вектор, 92, 181
 Именованный список, 198

Искусственный интеллект, 268

**К**

Квадратная диаграмма, 134
 Кортёж, 207

Л

Линейная регрессионная модель, 150
 Линейные модели, 149
 Линии тренда, 149
 Локальная регрессия, 154
 Локальный шлюз данных, 34, 294
 Лямбда-функция, 257

М

Метасимволы, 227
 Многослойная грамматика графиков, 42

О

Обобщенная линейная регрессия, 154
 Обобщенные аддитивные модели, 149, 154
 Обобщенные линейные модели, 149
 Обработка естественного языка, 233
 Окружение conda, 304
 Оператор конвейера, 75
 Опережающая проверка, 245

П

Поставщик (провайдер) данных, 212
 Предварительно обученная модель, 312
 Прогнозирование, 99
 Пространственная аналитика, 123
 Пузырьковая диаграмма, 89

Р

Рабочая директория, 160
 Регулярные выражения, 174, 224
 Робастные линейные модели, 154

С

Символьный вектор, 161
 Символьный тип, 161
 Скаляр, 199
 Словарь, 169, 207
 Стандартизация, 274
 Столбчатая диаграмма, 58

Т

Темы, 53
 Тепловая карта, 124

Ш

Шкалы, 64

Э

Экранирование, 161
 Эстетики, 48

Книги издательства «ДМК ПРЕСС»
можно купить оптом и в розницу
в книготорговой компании «Галактика»
(представляет интересы издательств
«ДМК ПРЕСС», «СОЛОН ПРЕСС», «КТК Галактика»).

Адрес: г. Москва, пр. Андропова, 38;
тел.: **(499) 782-38-89**, электронная почта: **books@aliants-kniga.ru**.

При оформлении заказа следует указать адрес (полностью),
по которому должны быть высланы книги;
фамилию, имя и отчество получателя.

Желательно также указать свой телефон и электронный адрес.

Эти книги вы можете заказать и в интернет-магазине: **www.a-planeta.ru**.



Райан Уэйд

Аналитика в Power BI с помощью R и Python

Главный редактор	<i>Мовчан Д. А.</i>
	<i>dmkpress@gmail.com</i>
Зам. главного редактора	<i>Сенченкова Е. А.</i>
Перевод	<i>Гинько А. Ю.</i>
Корректор	<i>Синяева Г. И.</i>
Верстка	<i>Чаннова А. А.</i>
Дизайн обложки	<i>Мовчан А. Г.</i>

Гарнитура PT Serif. Печать цифровая.
Усл. печ. л. 27,46. Тираж 200 экз.

Веб-сайт издательства: **www.dmkpress.com**

Аналитика в Power BI с помощью R и Python

Загрузка, преобразование и визуализация данных

Данная книга поможет вам научиться использовать языки программирования R и Python в аналитике совместно с Microsoft Power BI. Эксперт в области анализа данных и автор книги Райан Уэйд продемонстрирует на примерах, как можно легко и просто применить R и Python там, где стандартных средств Power BI просто недостаточно. Помимо прочего, вы научитесь анализировать данные в Power BI с применением пользовательских моделей машинного обучения и мощных моделей из состава службы Microsoft Cognitive Services.

Языки R и Python стоит рассматривать в качестве полезного дополнения к Power BI. С их помощью можно проводить углубленный анализ и преобразование исходных данных с использованием техник, недоступных для стандартных средств Power BI.

Если вы являетесь бизнес-аналитиком, специалистом в области науки о данных и хотите превратить Power BI из обычного инструмента в полноценную систему для всестороннего анализа данных, эта книга – для вас!

Из этой книги вы узнаете, как:

- создавать улучшенные визуализации в Power BI с использованием языка R и пакета ggplot2;
- преобразовывать и импортировать данные при помощи R и Python без ограничений, характерных для Power Query;
- применять модели машинного обучения к данным без необходимости приобретать платную версию Power BI Premium;
- внедрять модели искусственного интеллекта в Power BI с применением служб Microsoft Cognitive Services, IBM Watson Natural Language Understanding и предварительно обученных моделей из состава службы SQL Server Machine Learning Services;
- выполнять продвинутые манипуляции со строками в Power BI с использованием регулярных выражений при помощи R и Python.

Интернет-магазин:

www.dmkpress.com

Оптовая продажа:

КТК «Галактика»

e-mail: books@aliants-kniga.ru

Apress
ДМК
издательство
www.dmk.ru

ISBN 978-5-97060-923-1



9 785970 609231 >