O'REILLY®

Третье издание



Думай как аналитик

Статистика и данные с примерами на Python

SPRINT book

Аллен Дауни

THIRD EDITION

Think Stats

Exploratory Data Analysis

Allen B. Downey

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

ТРЕТЬЕ ИЗДАНИЕ

Думай как аналитик

Статистика и данные с примерами на Python

Аллен Дауни

SPRINT book 2026

Аллен Дауни

Думай как аналитик. Статистика и данные с примерами на Python

3-е издание

Перевел с английского Е. Суворкин

Научный редактор К. Быков

ББК 32.972.233-018.2
УДК 004.657

Дауни Аллен

Д21 Думай как аналитик. Статистика и данные с примерами на Python. 3-е изд. — Астана: «Спринт Бук», 2026. — 304 с.: ил.

ISBN 978-601-12-8917-7

Умение писать код — это все, что нужно, чтобы извлекать из данных ценные выводы и находить ответы на сложные вопросы. В этом полностью переработанном издании статистические концепции представлены не в виде громоздких математических формул, а как стройный вычислительный процесс на языке Python. На практических примерах с использованием реальных датасетов изучите весь цикл разведочного анализа данных — от первичной обработки данных и подсчета статистик до выявления закономерностей и проверки гипотез.

Являетесь ли вы специалистом data science, инженером-разработчиком или просто интересуетесь данными, вы быстро освоите такие популярные инструменты, как NumPy, SciPy и Pandas. Познакомьтесь с функциями распределения, статистическими зависимостями между переменными, методами визуализации и многими другими концепциями. Кроме того, все главы книги доступны в виде Jupyter-блокнотов, позволяющих читать текст, запускать код и выполнять упражнения, не переключаясь между инструментами.

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

ISBN 978-1098190255 англ.

Authorized Russian translation of the English edition of Think Stats, 3E
ISBN 9781098190255 © 2025 Allen B. Downey .
This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

ISBN 978-601-12-8917-7

© Перевод на русский язык ТОО «Спринт Бук», 2026
© Издание на русском языке, оформление ТОО «Спринт Бук», 2026

Права на издание получены по соглашению с O'Reilly. Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме без письменного разрешения владельцев авторских прав.

Информация, содержащаяся в данной книге, получена из источников, рассматриваемых издательством как надежные. Тем не менее, имея в виду возможные человеческие или технические ошибки, издательство не может гарантировать абсолютную точность и полноту приводимых сведений и не несет ответственности за возможные ошибки, связанные с использованием книги. Издательство не несет ответственности за доступность материалов, ссылки на которые вы можете найти в этой книге. На момент подготовки книги к изданию все ссылки на интернет-ресурсы были действующими. В книге возможны упоминания организаций, деятельность которых запрещена на территории Российской Федерации, таких как Meta Platforms Inc., Facebook, Instagram и др.

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

Дата изготовления: 07.2026. Наименование: книжная продукция. Срок годности: не ограничен.

Подписано в печать 04.06.26. Формат 70x100/16. Бумага офсетная. Усл. п. л. 24,510. Тираж 1000. Заказ 0000.

Оглавление

https://t.me/it_boooks/2

От издательства **10**
О научном редакторе русского издания 10

Предисловие **11**
Новое в этом издании 13
Использование исходного кода примеров 14
Условные обозначения 15

Благодарности **17**

Глава 1. Разведочный анализ данных **18**
Данные 18
Национальное исследование роста семьи США 20
Считывание данных 21
Проверка данных 24
Преобразование данных 27
Сводные статистические показатели 28
Интерпретация данных 30
Глоссарий 30
Упражнения 32

Глава 2. Распределение данных **33**
Таблицы частот 33
Распределения датасета NSFG 35
Выбросы 39
Первые дети 41
Размер эффекта 42
Представление результатов 44
Глоссарий 45
Упражнения 46

Глава 3. Функция вероятности **47**
Функция вероятности 47
Сводные характеристики функции вероятности 50

Парадокс размера группы ... 52
Данные NSFG ... 55
Дополнительная визуализация ... 56
Глоссарий ... 58
Упражнения ... 58

Глава 4. Интегральная функция распределения ... 60

Процентиль и процентильный ранг ... 60
Интегральная функция распределения ... 63
Сравнение ИФР ... 67
Процентильная статистика ... 69
Случайные числа ... 74
Глоссарий ... 75
Упражнения ... 76

Глава 5. Моделирование распределений ... 78

Биномиальное распределение ... 78
Распределение Пуассона ... 83
Экспоненциальное распределение ... 87
Нормальное распределение ... 91
Логнормальное распределение ... 94
Зачем нужны модели ... 97
Глоссарий ... 98
Упражнения ... 99

Глава 6. Функция плотности вероятности ... 101

Сравнение распределений ... 101
Плотность вероятности ... 104
Экспоненциальная функция плотности вероятности ... 107
Сравнение функции вероятности и функции плотности вероятности ... 109
Ядерная оценка плотности ... 111
Схема взаимосвязи распределений ... 115
Глоссарий ... 120
Упражнения ... 121

Глава 7. Связь между переменными ... 123

Диаграмма рассеяния ... 123
Децильная диаграмма ... 127
Корреляция ... 130

Сила корреляционной связи ... 134
Ранговая корреляция ... 135
Корреляция и причинно-следственная связь ... 139
Глоссарий ... 140
Упражнения ... 141

Глава 8. Оценка ... 144

Вес пингвинов ... 144
Робастность ... 148
Оценка дисперсии ... 150
Выборочное распределение ... 151
Стандартная ошибка ... 154
Доверительный интервал ... 155
Источники ошибок ... 156
Глоссарий ... 157
Упражнения ... 158

Глава 9. Проверка гипотез ... 163

Эксперимент с монетой ... 163
Проверка разницы средних ... 166
Прочие статистики критерия ... 169
Проверка значимости корреляции ... 170
Проверка пропорций ... 173
Глоссарий ... 176
Упражнения ... 177

Глава 10. Метод наименьших квадратов ... 179

Линейная регрессия методом наименьших квадратов ... 179
Коэффициент детерминации ... 183
Минимизация MSE ... 185
Оценка ... 186
Визуализация неопределенности ... 188
Преобразование переменных ... 190
Глоссарий ... 195
Упражнения ... 196

Глава 11. Множественная регрессия ... 198

Модуль StatsModels ... 198
Знакомство с множественной регрессией ... 202
Контрольная переменная ... 204

Нелинейная зависимость ... 208
Логистическая регрессия ... 211
Глоссарий ... 215
Упражнения ... 216

Глава 12. Анализ временных рядов ... 218

Данные об электрогенерации ... 218
Декомпозиция ... 220
Предсказание данных ... 226
Мультипликативная модель ... 229
Авторегрессия ... 234
Скользящее среднее ... 237
Ретроспективная оценка с авторегрессией ... 238
ARIMA ... 240
Предсказание с помощью ARIMA ... 242
Глоссарий ... 244
Упражнения ... 245

Глава 13. Анализ выживаемости ... 248

Функция выживания ... 248
Функция риска ... 250
Данные о браке ... 252
Взвешенный бутстрепинг ... 255
Оценка функции риска ... 257
Оценка функции выживания ... 260
Библиотека Lifelines ... 262
Доверительный интервал ... 264
Ожидаемое остаточное время жизни ... 266
Глоссарий ... 269
Упражнения ... 270

Глава 14. Аналитические методы ... 273

График нормальной вероятности ... 273
Нормальное распределение ... 277
Распределение выборочных средних ... 281
Распределение разностей ... 282
Центральная предельная теорема ... 285
Ограничения центральной предельной теоремы ... 287

Применение центральной предельной теоремы ... 289
Критерий корреляции ... 292
Критерий хи-квадрат ... 296
Вычислительные и аналитические методы ... 299
Глоссарий ... 300
Упражнения ... 301

Об авторе ... 303

Иллюстрация на обложке ... 304

От издательства

Мы выражаем огромную благодарность сотрудникам Центрального университета за помощь в работе над русскоязычным изданием книги и вклад в повышение качества переводной литературы.

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

Мы будем рады узнать ваше мнение!

О научном редакторе русского издания

Кирилл Быков — старший аналитик-разработчик во «ВКонтакте». Занимается поддержкой внутренней MapReduce-системы, разработкой платформы AB-тестирования и дата-инженерией. Преподаватель курсов по основам машинного обучения, глубокого обучения, программирования на Python и продуктовой аналитике в МФТИ, НИУ ВШЭ и Центральном университете. Развивает аналитические инструменты и ведет курсы по машинному обучению, вдохновляя студентов на освоение современных методов анализа данных и искусственного интеллекта.

Предисловие

С самых первых дней существования статистики взгляды на ее природу разделились. По одним представлениям, статистика — это раздел математики, цель которого — заложить теоретическую основу для теории вероятностей и статистического вывода. По другим — это набор инструментов и методов для работы с данными, поиска ответов на вопросы и принятия более взвешенных решений. Многие вводные курсы по статистике придерживаются первого подхода. Однако эта книга основывается на втором.

«Думай как аналитик» — это введение в практические методы исследования и визуализации данных, выявления взаимосвязей и тенденций, а также представления полученных результатов. Структура книги соответствует той процедуре, которой я следую, когда начинаю работать с новым датасетом.

Импортирование и очистка данных

В каком бы формате ни были представлены данные, обычно приходится уделить определенное время и приложить усилия, чтобы их считать, очистить и преобразовать, а также убедиться, что в процессе этих превращений ничего не потерялось.

Разведочный анализ одной переменной

Обычно я начинаю с изучения каждой переменной по отдельности, выясняя, что переменная означает, строя распределение ее значений и выбирая подходящие сводные статистические показатели.

Разведочный анализ пар переменных

Чтобы определить возможные взаимосвязи между переменными, я просматриваю таблицы и диаграммы рассеяния, а также вычисляю корреляции и подгоняю линию регрессии.

Многомерный анализ

Если между переменными есть очевидные взаимосвязи, я использую множественную регрессию, чтобы добавить контрольные переменные и исследовать более сложные соотношения.

Оценка и проверка гипотез

Перед тем как представлять результаты статистического анализа, важно ответить на следующие три вопроса. Насколько велик эффект? Какую изменчивость можно ожидать при повторных измерениях? Может ли быть так, что наблюдаемый эффект вызван случайностью?

Визуализация

Визуализация — важный инструмент для выявления возможных взаимосвязей и эффектов в ходе исследований. Далее, если наблюдаемый эффект успешно проходит тщательную проверку, визуализация становится эффективным способом представления полученных результатов.

В этой книге реализован вычислительный подход, который имеет ряд преимуществ перед более математическими методами.

- Большую часть концепций я представляю в виде кода на языке Python, а не с помощью математических формул. Прежде всего, код на Python более удобочитаем. Кроме того, поскольку код исполняем, читатель может запускать и изменять его, чтобы лучше в нем разобраться.
- Каждая глава завершается блоком упражнений, выполняя которые читатель может проверить и закрепить полученные знания. Когда вы пишете программу, то воплощаете свое понимание в коде, а во время отладки программы проверяете, что ваше понимание верно.
- Некоторые упражнения предполагают эксперименты для проверки статистических характеристик. Например, вы можете изучать центральную предельную теорему (ЦПТ) в действии, генерируя случайные выборки и вычисляя их суммы. Полученные графики показывают, когда ЦПТ работает, а когда нет.
- Некоторые идеи, которые трудно усвоить на языке математики, легко понять в процессе моделирования. Например, получая аппроксимацию *p*-значения с помощью случайного моделирования, мы закрепляем понимание концепции проверки гипотез.
- Поскольку в книге использован язык программирования общего назначения (Python), данные можно импортировать практически из любого источника и не ограничиваться датасетами, очищенными и отформатированными под конкретный статистический инструмент.

Для демонстрации своего подхода к статистическому анализу в примерах и упражнениях я использую данные из различных источников, среди которых:

- Национальное исследование роста семьи (National Survey of Family Growth, NSFG) (https://oreil.ly/6V82p), проведенное Центрами по контролю и профилактике заболеваний (Centers for Disease Control and Prevention, CDC) США с целью сбора «информации о семейной жизни, вступлении в брак и расторжении брака, беременности, бесплодии, использовании средств контрацепции, а также здоровье мужчин и женщин».
- Исследование в рамках Системы наблюдения за поведенческими факторами риска (Behavioral Risk Factor Surveillance System, BRFSS) (http://cdc.gov/BRFSS), проводимое Национальным центром профилактики хронических

заболеваний и укрепления здоровья (National Center for Chronic Disease Prevention and Health Promotion) США для «наблюдения за состоянием здоровья и рискованным поведением в США».

- Датасет «пингвины станции Палмер» (The Palmer Penguins, https://oreil.ly/kl2BD), который включает в себя результаты измерений выборки пингвинов в районе антарктической станции Палмер.
- Данные Управления энергетической информации США (Energy Information Administration, EIA) о производстве электроэнергии из возобновляемых источников в США.

Я выражаю благодарность отдельным людям и организациям, которые собрали эти данные и выложили их в свободный доступ. Я также надеюсь, что работа с реальными данными из самых разных областей усилит читательский интерес.

Новое в этом издании

Подготовку третьего издания я начал с перевода книги в формат Jupyter-блокнотов, поскольку это дало очевидное преимущество: теперь можно читать текст, запускать код и работать над упражнениями в одной среде. Кроме того, блокноты предназначены для работы в Google Colab, поэтому, чтобы приступить к делу, не потребуется ничего устанавливать.

Переход на формат Jupyter-блокнотов имеет еще одно преимущество — код становится более удобочитаемым. В первых двух изданиях часть кода была в самой книге, а часть — в дополнительных файлах, доступных для скачивания в интернете. По прошествии времени стало ясно, что такое деление материала было не лучшим решением и неоправданно усложняло код. В третьем издании мне удалось упростить код и сделать его понятнее.

За время, прошедшее с выхода предыдущего издания, мне удалось усовершенствовать библиотеку `empiricaldist`, в которой определены классы, представляющие статистические распределения. Библиотека стала более зрелой, поэтому обновленный код позволяет использовать ее более эффективно.

Когда я начинал этот проект, библиотеки NumPy и SciPy еще не получили широкого распространения, а библиотека Pandas была известна еще меньше. Поэтому в оригинальном коде использовались такие структуры данных языка Python, как списки и словари. В этом издании активно используются NumPy-массивы, структуры Pandas, а также функции, реализованные в этих библиотеках.

В третьем издании рассматриваются те же темы и почти в том же порядке, что и в первом издании, но текст существенно переработан. Некоторые примеры появились впервые, другие дополнены новыми данными. Я также добавил новые упражнения, некоторые из старых переработал, а некоторые удалил. Обновленные

упражнения, по моему мнению, лучше согласуются с примерами и стали интереснее.

В основу этой книги, начиная с первого ее издания, был положен тезис о том, что многие идеи, которые трудно объяснимы с помощью математики, проще понять в коде. Будучи верен этому принципу, в этом издании я изложил материал так, что математических обозначений в тексте почти не осталось.

В целом я считаю, что все эти изменения сделали книгу «Думай как аналитик» лучше. Надеюсь, она вам понравится!

Использование исходного кода примеров

Вспомогательные материалы (код, данные и т. д.) доступны для загрузки из Git-репозитория на GitHub по адресу https://oreil.ly/ThinkStatsCode. Git — это система контроля версий, которая помогает следить за изменениями в файлах проекта. Коллекция файлов, находящихся под управлением Git, называется **репозиторием**. GitHub предоставляет услуги хостинга, в частности хранилище для Git-репозиториев и удобный веб-интерфейс.

Для каждой главы этой книги в репозитории имеется свой Jupyter-блокнот — документ, содержащий текст, код и результаты исполнения кода. Вы можете использовать файлы блокнотов, чтобы исполнять код и работать над упражнениями.

Существует два способа запустить код из Jupyter-блокнота. Самый простой — использовать Colab, сервис Google, позволяющий исполнять код в веб-браузере без необходимости устанавливать что-либо на компьютер. На домашней странице книги (https://allendowney.github.io/ThinkStats) вы найдете ссылки на Jupyter-блокноты, в том числе на вводный блокнот, который знакомит с сервисом Colab и Jupyter-блокнотами.

Если вы не хотите использовать Colab, то можете загрузить файлы блокнотов и исполнять их на своем компьютере, но в этом случае вам придется установить Python, Jupyter и библиотеки, используемые в книге, в частности NumPy, SciPy и StatsModels. Если у вас есть опыт установки программного обеспечения, настройка среды для запуска Jupyter-блокнотов не составит большого труда. Но если такого опыта нет, все это может показаться вам сложным и изнурительным и помешать получить максимальную пользу от этой книги. Если хотите научиться разведочному анализу данных на Python, не тратьте свое время и силы на установку программ! Я настоятельно рекомендую вам прочитать хотя бы первые несколько глав в среде Colab. Затем, если захотите настроить среду на собственном компьютере, то можете сделать это, не прерывая изучение книги. И последний совет: если у вас возникают какие-либо проблемы с установкой программ, воспользуйтесь такими инструментами, как ChatGPT, — они обычно хорошо помогают в этих вопросах.

При написании этой книги я исходил из того, что вы знакомы с основами языка Python, в том числе объектно-ориентированным программированием. Знакомство с библиотеками NumPy и Pandas будет полезно, но необязательно: я дам все необходимые пояснения. Я также предполагаю, что вы знакомы с основными понятиями математики, например логарифмами и суммированием. От вас не требуется знание линейной алгебры или математического анализа. Один раз я буду говорить о производных и интегралах, но если вы не знакомы с этими понятиями, ничего страшного. И наконец, знаний статистики для чтения книги не требуется.

Если у вас возникнут вопросы технического характера по использованию примеров кода, направляйте их по электронной почте на адрес support@oreilly.com.

В общем случае все примеры кода из книги вы можете использовать в своих программах и в документации. Вам не нужно обращаться в издательство за разрешением, если вы не собираетесь воспроизводить существенные части программного кода. Если вы разрабатываете программу и используете в ней несколько фрагментов кода из книги, вам не нужно обращаться за разрешением. Но для продажи или распространения примеров из книги вам потребуется разрешение от издательства O'Reilly. Вы можете отвечать на вопросы, цитируя данную книгу или примеры из нее, но для включения существенных объемов программного кода из книги в документацию вашего продукта потребуется разрешение.

Мы рекомендуем, но не требуем добавлять ссылку на первоисточник при цитировании. Под ссылкой на первоисточник мы подразумеваем указание названия книги, автора, издательства и ISBN.

За разрешением на использование значительных объемов программного кода из книги обращайтесь по адресу permissions@oreilly.com.

Условные обозначения

В этой книге используются следующие условные обозначения:

Жирный шрифт

Показывает новые термины, каждому из которых дано определение в глоссарии.

Курсив

Курсивом выделены важные понятия.

`Моноширинный шрифт`

Используется для листингов программ, а также внутри абзацев для обозначения таких элементов, как переменные и функции, базы данных, типы данных, переменные среды, операторы и ключевые слова.

`Жирный моноширинный шрифт`

Показывает команды или другой текст, который пользователь должен ввести самостоятельно.

`Курсивный моноширинный шрифт`

Показывает текст, который должен быть заменен значениями, введенными пользователем, или значениями, определяемыми контекстом.

Шрифт без засечек

Используется для обозначения URL и адресов электронной почты.

Благодарности

Благодарю читателей, которые прислали исправления и предложения к предыдущим изданиям этой книги, а также студентов Олин-колледжа (Olin College), которые терпеливо вычитывали черновые варианты книги.

Большое спасибо научным редакторам этого издания — Питеру Брюсу (Peter Bruce), Закари дель Росарио (Zachary del Rosario), Уолтеру Р. Пачковски (Walter R. Paczkowski) и Томасу Нилду (Thomas Nield).

И спасибо всем в издательстве O'Reilly Media, особенно редакторам Саре Хантер (Sara Hunter) и Аарону Блэку (Aaron Black).

ГЛАВА 1

Разведочный анализ данных

https://t.me/it_boooks/2

Основной постулат этой книги гласит: данные можно использовать для ответа на вопросы, разрешения споров и принятия более эффективных решений.

Эта глава познакомит вас с процедурой, которой мы будем для этого придерживаться: загрузкой, проверкой и разведкой данных, а также выбором статистических показателей, измеряющих то, что нас интересует. В качестве примера воспользуемся данными Национального исследования роста семьи (National Survey of Family Growth, NSFG) США для ответа на вопрос, с которым я столкнулся, когда мы с женой ожидали нашего первенца: правда ли, что первые дети появляются на свет позже?

Данные

Возможно, вы слышали, что первые дети чаще рождаются позже срока. Поиск в интернете выдаст множество дискуссий на эту тему. Одни утверждают, что это правда, другие — что это миф, а некоторые заявляют ровно обратное: первые дети появляются на свет раньше срока.

Часто в подобных спорах оппоненты в подтверждение собственных слов ссылаются на данные. Вот несколько примеров такой аргументации:

> «У обеих моих подруг, недавно родивших первенцев, роды проходили почти на две недели позже срока — только тогда начались схватки или было проведено стимулирование родов».
>
> «Мой первый появился с опозданием на две недели, и поэтому я думаю, что второй появится на две недели раньше!»
>
> «Я не верю этому, потому что моя сестра, которая была первым ребенком у моей матери, родилась рано — как и многие мои двоюродные братья и сестры».

Подобные сообщения называют **бытующим, или неподтвержденным, мнением**, поскольку они основаны на неопубликованных и, как правило, личных данных. Нет ничего предосудительного, когда подобные факты из жизни приводятся в бытовых разговорах, и поэтому у меня нет намерения придираться к людям, которых я только что процитировал.

Однако нам понадобятся более убедительные доказательства и ответ, заслуживающий большего доверия. Чтобы их получить, «личного опыта» обычно бывает недостаточно по следующим причинам.

Малое количество наблюдений

Если беременность у родящих впервые длится дольше, то разница, вероятно, невелика по сравнению с естественным разбросом. В этом случае нам, возможно, придется сравнить большое количество беременностей, чтобы понять, имеет ли место различие.

Предвзятость в отборе данных

Вступающие в подобные дискуссии люди могут являться заинтересованными сторонами, если их первые дети родились поздно. В этом случае процесс отбора данных может повлиять на результаты.

Предвзятость подтверждения

Люди, верящие в некоторое утверждение, чаще будут приводить примеры, подтверждающие его. И наоборот, сомневающиеся в нем с большей вероятностью приведут контрпримеры.

Неточность

Бытующее мнение часто основано на личном опыте, который может быть неверно припомнен, искажен, неточно воспроизведен и т. д.

Чтобы устранить эти недостатки, мы будем использовать статистические инструменты, среди которых можно упомянуть следующие:

Сбор данных

Мы задействуем данные крупного национального опроса, который был специально разработан с целью получения статистически обоснованных выводов о населении США.

Описательная статистика

Мы вычислим статистические показатели, кратко описывающие данные, и сравним различные способы визуализации данных.

Разведочный анализ данных

Мы будем искать закономерности, различия и другие особенности, которые отвечают на интересующие нас вопросы. Помимо этого, мы будем проверять наличие несоответствий и выявлять ограничения.

Оценка

Мы будем использовать данные некоторой выборки для оценки характеристик генеральной совокупности.

Проверка гипотез

В тех случаях, когда наблюдается некоторый эффект, например разница между двумя группами, мы будем оценивать вероятность того, что этот эффект появился случайно.

Придерживаясь этой последовательности шагов и стараясь избегать ошибок, мы сможем прийти к выводам, которые будут более обоснованными и с большей вероятностью окажутся верными.

Национальное исследование роста семьи США

С 1973 года Центры по контролю и профилактике заболеваний (Centers for Disease Control and Prevention, CDC) США проводят Национальное исследование роста семьи (NSFG), целью которого является сбор «информации о семейной жизни, браке и разводах, беременности, бесплодии, использовании средств контрацепции, а также о здоровье мужчин и женщин. Результаты обследования используются... для планирования медицинских услуг и программ санитарного просвещения, а также для проведения статистических исследований семей, рождаемости и здоровья».

Мы воспользуемся собранными в этом исследовании данными, чтобы понять, рождаются ли первенцы позже, и ответить на другие вопросы. Чтобы эффективно использовать эти данные, нужно понимать дизайн статистического исследования.

Как правило, цель статистического исследования — сделать выводы о **генеральной совокупности** (популяции). В NSFG изучаемой популяцией являются жители Соединенных Штатов в возрасте от 15 до 44 лет.

В идеальном случае исследование должно содержать данные о каждом представителе популяции, но такое редко бывает возможно. Вместо этого собирают данные о некотором подмножестве генеральной совокупности, называемом **выборкой**. Люди, участвующие в опросе, зовутся **респондентами**.

NSFG представляет собой **поперечное, или одномоментное, исследование** (cross-sectional study); это означает, что оно фиксирует моментальный снимок популяции в определенный момент времени. Опрос в рамках NSFG проходил уже несколько раз; каждое его проведение называется **циклом**. Мы будем использовать данные 6-го цикла, который проходил с января 2002 по март 2003 года.

Обычно считается, что поперечные исследования являются **репрезентативными**; это означает, что выборка похожа на целевую популяцию во всех отношениях, которые важны для целей исследования. Такого идеала трудно достичь на практике, но специалисты, проводящие опросы, стараются приблизиться к нему настолько, насколько это возможно.

Исследование NSFG является не репрезентативным, а **стратифицированным**; это означает, что в нем для некоторых групп населения намеренно используется **избыточная выборка**. Разработчики исследования для трех групп населения — испаноязычных, афроамериканцев и подростков — увеличивали долю респондентов по сравнению с тем, как они представлены в населении США. Число респондентов в каждой группе, таким образом, становилось достаточным, чтобы можно было делать обоснованные выводы. Недостатком избыточной выборки является то,

что на основе ее статистических показателей делать выводы о генеральной совокупности становится труднее. Позже мы вернемся к этому вопросу.

При работе с такого рода данными важно ознакомиться с **кодбуком** (codebook), в котором документированы структура исследования, вопросы анкеты, а также коды ответов.

Считывание данных

Перед загрузкой данных NSFG требуется согласие с условиями использования.

Любая преднамеренная идентификация или раскрытие информации о физическом или юридическом лице нарушает гарантии конфиденциальности, предоставленные поставщикам информации. Таким образом, пользователи обязуются:

- Использовать данные из этого датасета исключительно для статистической отчетности и анализа.
- Не пытаться установить личность физических или идентичность юридических лиц, представленных в этих данных.
- Не связывать этот датасет с данными, позволяющими проводить идентификацию, как из других датасетов Национального центра статистики здравоохранения (National Center for Health Statistics, NCHS) США, так и ему не принадлежащих.
- Не предпринимать никаких попыток оценить методы раскрытия информации, применяемые для защиты физических и юридических лиц, а также не проводить никаких исследований по методам повторной идентификации физических и юридических лиц.

Если вы согласны соблюдать эти условия, то инструкции по загрузке данных можно найти в файле Jupyter-блокнота для этой главы.

Данные хранятся в двух файлах — «словаре» с описанием формата данных и файле данных:

```
dct_file = "2002FemPreg.dct"
dat_file = "2002FemPreg.dat.gz"
```

В Jupyter-блокноте для главы 1 определена функция, считывающая эти файлы. Она называется `read_stata`, поскольку формат этих данных совместим со статистическим программным пакетом Stata.

Вот пример ее использования:

```
preg = read_stata(dct_file, dat_file)
```

Функция возвращает объект типа `DataFrame`, который является структурой данных библиотеки Pandas и представляет табличные данные в виде строк и столбцов. В этой таблице каждая строка соответствует беременности, о которой сообщила респондентка, а столбец — одной из **переменных**. Каждая переменная может содержать ответы на вопросы анкеты или значения, рассчитанные на основе ответов на один или несколько вопросов.

Помимо данных, `DataFrame` содержит имена и типы переменных, а также предоставляет методы для доступа к данным и их модификации. Объект `DataFrame` имеет атрибут `shape`, содержащий количество строк и столбцов:

```
preg.shape
```

```
(13593, 243)
```

Этот датасет содержит 243 переменные с информацией о 13 593 беременностях. Объект `DataFrame` предоставляет метод `head`, который выводит на экран первые несколько строк:

```
preg.head()
```

	caseid	pregordr	howpreg_n	howpreg_p	moscurrp	nowprgdk	pregend1	pregend2	nbrnaliv	…
0	1	1	NaN	NaN	NaN	NaN	6.0	NaN	1.0	…
1	1	2	NaN	NaN	NaN	NaN	6.0	NaN	1.0	…
2	2	1	NaN	NaN	NaN	NaN	5.0	NaN	3.0	…
3	2	2	NaN	NaN	NaN	NaN	6.0	NaN	1.0	…
4	2	3	NaN	NaN	NaN	NaN	6.0	NaN	1.0	…

В левом столбце содержится индекс объекта `DataFrame`, который состоит из меток для каждой строки. В данном случае в качестве меток выступают целые числа, начинающиеся с 0, но они также могут быть строками или другими типами.

Также `DataFrame` имеет атрибут `columns`, содержащий имена переменных (столбцов):

```
preg.columns
```

```
Index(['caseid', 'pregordr', 'howpreg_n', 'howpreg_p', 'moscurrp', 'nowprgdk',
       'pregend1', 'pregend2', 'nbrnaliv', 'multbrth',
       …
       'poverty_i', 'laborfor_i', 'religion_i', 'metro_i', 'basewgt',
       'adj_mod_basewgt', 'finalwgt', 'secu_p', 'sest', 'cmintvw'],
      dtype='object', length=243)
```

Имена столбцов содержатся в объекте типа `Index` — еще одной структуре данных Pandas. Чтобы получить доступ к столбцу объекта `DataFrame`, можно воспользоваться именем столбца в качестве ключа:

```
pregordr = preg["pregordr"]
type(pregordr)
```

```
pandas.core.series.Series
```

Результатом является объект типа `Series` библиотеки Pandas, представляющий собой последовательность значений. У `Series` тоже имеется метод `head`, отображающий первые несколько значений и их метки:

```
pregordr.head()
```

```
0    1
1    2
2    1
3    2
4    3
Name: pregordr, dtype: int64
```

Последняя строка содержит название объекта `Series` и `dtype`, являющийся типом значений. В данном примере `int64` указывает, что значения являются 64-разрядными целыми числами.

Датасет NSFG содержит в общей сложности 243 переменные. Вот некоторые из тех, что мы будем использовать для разведочного анализа в этой книге:

`caseid`
: Целочисленный идентификатор (ID) респондентки.

`pregordr`
: Порядковый номер беременности: для первой беременности респондентки код равняется 1, для второй беременности — 2 и т. д.

`prglngth`
: Продолжительность беременности в неделях (целое число).

`outcome`
: Целочисленный код, обозначающий исход беременности. Код 1 указывает на рождение живого ребенка.

`birthord`
: Порядковый номер рождения живого ребенка: для первого ребенка респондентки код равен 1 и т. д. Для других исходов это поле остается пустым.

`birthwgt_lb` *и* `birthwgt_oz`
: Содержат вес ребенка при рождении в фунтах и унциях.

`agepreg`
: Возраст матери на момент окончания беременности.

`finalwgt`
: Статистический вес, связанный с данной респонденткой. Это число с плавающей точкой, указывающее на количество женщин в популяции США, которых респондентка представляет.

Если вы внимательно ознакомитесь с кодбуком, то обнаружите, что многие переменные являются **перекодированными**. Это означает, что они не принадлежат к **исходным данным**, собранным в ходе опроса, а являются результатом расчета с использованием исходных данных.

Например, значение `prglngth` в случае живорождения равно исходной переменной `wksgest` (недели беременности) при ее наличии; в противном случае оно оценивается по формуле `mosgest * 4.33` (месяцы беременности, умноженные на среднее количество недель в месяце).

Перекодирование переменных часто подразумевает проверку согласованности и точности данных. Поэтому обычно рекомендуется пользоваться перекодированными переменными, когда они доступны, если только нет веских причин для самостоятельной обработки исходных данных.

Проверка данных

При экспорте данных из одной программной среды и последующем импорте в другую могут возникать ошибки. Когда вы только знакомитесь с новым датасетом, то можете неправильно декодировать данные или неверно интерпретировать их значение. Если вы уделите время проверке данных, то сможете впоследствии его сэкономить и предотвратить появление ошибок.

Один из способов проверки корректности данных — вычисление основных статистических показателей и их сравнение с опубликованными результатами. Например, кодбук NSFG содержит сводные таблицы для всех переменных. Вот таблица для переменной `outcome`, содержащей коды исходов всех беременностей:

Value (Значение)	Label (Метка)	Total (Всего)
1	LIVE BIRTH (ЖИВОРОЖДЕНИЕ)	9148
2	INDUCED ABORTION (ИСКУССТВЕННЫЙ АБОРТ)	1862
3	STILLBIRTH (МЕРТВОРОЖДЕНИЕ)	120

Value (Значение)	Label (Метка)	Total (Всего)
4	MISCARRIAGE (ВЫКИДЫШ)	1921
5	ECTOPIC PREGNANCY (ВНЕМАТОЧНАЯ БЕРЕМЕННОСТЬ)	190
6	CURRENT PREGNANCY (ТЕКУЩАЯ БЕРЕМЕННОСТЬ)	352
Total (Итого)		13593

В столбце «Всего» указано количество беременностей для каждого исхода. Чтобы проверить эти значения, воспользуемся методом `value_counts`, который подсчитывает количество появлений каждого значения, и методом `sort_index`, сортирующим результаты в соответствии со значениями колонки `Index` (слева):

```
preg["outcome"].value_counts().sort_index()
```

```
outcome
1    9148
2    1862
3     120
4    1921
5     190
6     352
Name: count, dtype: int64
```

Сравнивая наши результаты с опубликованной таблицей, мы убеждаемся, что значения в столбце `outcome` верны. Вот аналогичная официальная таблица для `birthwgt_lb`:

Value (Значение)	Label (Метка)	Total (Всего)
.	inapplicable (неприменимо)	4449
0–5	UNDER 6 POUNDS (МЕНЕЕ 6 ФУНТОВ)	1125
6	6 POUNDS (6 ФУНТОВ)	2223
7	7 POUNDS (7 ФУНТОВ)	3049
8	8 POUNDS (8 ФУНТОВ)	1889
9–95	9 POUNDS OR MORE (9 ФУНТОВ И БОЛЕЕ)	799
97	Not ascertained (Не установлено)	1
98	REFUSED (ОТКАЗ)	1
99	DON'T KNOW (НЕИЗВЕСТНО)	57
Total (Итого)		13593

Вес при рождении указан только для беременностей, окончившихся рождением живого ребенка. В таблице указано, что в 4449 случаях эта переменная неприменима. Кроме того, есть один случай, когда вопрос не был задан, один случай, когда респондентка не ответила, и 57 случаев, когда вес был неизвестен опрашиваемым.

Здесь, чтобы сравнить суммарные значения из датасета с опубликованными в кодбуке, тоже можно использовать метод `value_counts`:

```
counts = preg["birthwgt_lb"].value_counts(dropna=False).sort_index()
counts
```

```
birthwgt_lb
0.0        8
1.0       40
2.0       53
3.0       98
4.0      229
5.0      697
6.0     2223
7.0     3049
8.0     1889
9.0      623
10.0     132
11.0      26
12.0      10
13.0       3
14.0       3
15.0       1
51.0       1
97.0       1
98.0       1
99.0      57
NaN     4449
Name:  count, dtype: int64
```

Аргумент `dropna=False` указывает `value_counts` не игнорировать значения «неприменимо» («Not applicable», или «NA»). Они помечены в результатах как `NaN`, что означает «не число» (Not a Number), а количество таких случаев соответствует числу значений «неприменимо» в кодбуке.

Число записей для 6, 7 и 8 фунтов соответствует кодбуку. Чтобы проверить подсчет для диапазона веса от 0 до 5 фунтов, можно воспользоваться атрибутом `loc` (сокращение от location — «местоположение») и индексом среза для выбора подмножества значений:

```
counts.loc[0:5]
```

```
birthwgt_lb
0.0      8
1.0     40
2.0     53
3.0     98
```

```
4.0    229
5.0    697
Name: count, dtype: int64
```

И можно задействовать метод `sum`, чтобы их суммировать:

```
counts.loc[0:5].sum()
```

```
1125
```

Итоговая сумма соответствует данным кодбука.

Значения 97, 98 и 99 соответствуют случаям, когда вес при рождении неизвестен. Существуют различные способы работы с отсутствующими данными. Простой вариант — заменить эти значения на `NaN`. Так же мы поступим и с очевидно неверным значением в 51 фунт[1].

Мы можем использовать метод `replace` («заменить») следующим образом:

```
preg["birthwgt_lb"] = preg["birthwgt_lb"].replace([51, 97, 98, 99], np.nan)
```

Первый аргумент метода — это список значений, подлежащих замене. Второй аргумент, `np.nan`, представляет собой значение `NaN` из библиотеки NumPy.

Когда вы таким образом считываете данные, то часто их приходится проверять на наличие ошибок и работать со специальными значениями. Подобные операции называются **очисткой данных**.

Преобразование данных

В рамках очистки данных иногда требуется изменить формат данных или провести с ними другие расчеты.

Например, переменная `agepreg` содержит возраст матери на момент окончания беременности. Он, согласно кодбуку, представляет собой целое число сотых долей года (centiyears), в чем можно убедиться, если применить метод `mean` для вычисления среднего значения:

```
preg["agepreg"].mean()
```

```
2468.8151197039497
```

Чтобы получить значение в годах, можно поделить весь столбец на 100:

```
preg["agepreg"] /= 100.0
preg["agepreg"].mean()
```

```
24.6881511970395
```

[1] Примерно соответствует 23 кг. — *Примеч. пер.*

Теперь среднее значение выглядит более правдоподобным.

Еще один пример: вес при рождении представлен в виде двух столбцов, `birthwgt_lb` и `birthwgt_oz`, содержащих фунты и унции соответственно. Будет удобнее объединить их в один столбец, содержащий вес в фунтах и долях фунта.

Сначала проведем очистку `birthwgt_oz`, как мы это делали в случае с `birthwgt_lb`:

```
preg["birthwgt_oz"] = preg["birthwgt_oz"].replace([97, 98, 99], np.nan)
```

Теперь, используя очищенные значения, создадим новый столбец, объединяющий фунты и унции в единую величину:

```
preg["totalwgt_lb"] = preg["birthwgt_lb"] + preg["birthwgt_oz"] / 16.0
preg["totalwgt_lb"].mean()
```

```
7.265628457623368
```

Среднее выглядит правдоподобным[1].

Сводные статистические показатели

Статистический показатель — это число, полученное из датасета, обычно предназначенное для количественной оценки некоторого параметра данных. В качестве примера можно назвать количество значений (count), среднее (mean), дисперсию (variance) и стандартное отклонение (standard deviation).

Объект `Series` имеет метод `count`, который возвращает количество значений, отличных от `nan`:

```
weights = preg["totalwgt_lb"]
n = weights.count()
n
```

```
9038
```

Он также предоставляет метод `sum`, возвращающий сумму всех значений; его можно использовать для вычисления среднего следующим образом:

```
mean = weights.sum() / n
mean
```

```
7.265628457623368
```

Но, как вы уже знаете, существует также метод `mean`, дающий тот же результат:

```
weights.mean()
```

```
7.265628457623368
```

[1] Примерно 3,3 кг. — *Примеч. пер.*

В нашем датасете средний вес при рождении составляет около 7.3 фунта.

Дисперсия — это статистический показатель, оценивающий разброс множества значений. Это среднее квадратов отклонений (squared deviations), равных расстоянию каждой точки от среднего значения:

```
squared_deviations = (weights - mean) ** 2
```

Среднее квадратов отклонений можно получить следующим образом:

```
var = squared_deviations.sum() / n
var
```

```
1.983070989750022
```

Объект `Series` ожидаемо предоставляет метод `var`, который делает *почти* то же самое:

```
weights.var()
```

```
1.9832904288326545
```

Результат слегка отличается от нашего, потому что в методе `var` при вычислении среднего квадратов отклонений происходит деление на `n-1`, а не на `n`. Это связано с тем, что существует два способа вычисления дисперсии выборки, в зависимости от того, что вы хотите сделать. Я объясню эту разницу в разделе «Оценка дисперсии» главы 8 на с. 150, но в реальной жизни это обычно не так важно. Если вы предпочитаете вариант с `n` в знаменателе, то можете получить его, передав методу `var` в качестве именованного аргумента `ddof=0`:

```
weights.var(ddof=0)
```

```
1.983070989750022
```

У наших данных дисперсия веса при рождении составляет около 1.98, но это значение трудно интерпретировать — прежде всего потому, что оно выражено в квадратных фунтах. Дисперсия полезна для некоторых вычислений, но не является хорошим способом описания данных. Более удачный показатель — **стандартное отклонение** (standard deviation), которое представляет собой квадратный корень из дисперсии. Его можно вычислить следующим образом:

```
std = np.sqrt(var)
std
```

```
1.40821553384062
```

Или же воспользоваться методом `std`:

```
weights.std(ddof=0)
```

```
1.40821553384062
```

В нашем датасете стандартное отклонение веса при рождении составляет около 1.4 фунта. Говоря просто, значения, лежащие в пределах одного или двух стандартных отклонений от среднего, являются распространенными, а более удаленные — редкими.

Интерпретация данных

Чтобы эффективно работать с данными, нужно обращать внимание не только на статистические показатели, но и на контекст. В качестве примера выберем в таблице с данными о беременности те строки, где переменная `caseid` равна 10229. Метод `query` принимает на вход строку, которая, помимо прочего, может содержать имена столбцов, операторы сравнения и числа:

```
subset = preg.query("caseid == 10229")
subset.shape
```

```
(7, 244)
```

Результатом является объект `DataFrame`, содержащий только те строки, для которых запрос принимает значение `True` (истина). Эта респондентка сообщила о семи беременностях. Вот их исходы, записанные в хронологическом порядке:

```
subset["outcome"].values
```

```
array([4, 4, 4, 4, 4, 4, 1])
```

Код исхода `1` обозначает рождение живого ребенка. Код `4` указывает на выкидыш, то есть потеря ребенка, как правило, по неизвестной физиологической причине.

Статистически случай этой респондентки не является чем-то необычным. Потеря ребенка — нередкое явление, к тому же есть другие респондентки, которые сообщили о таком же количестве случаев. Но эти данные рассказывают историю женщины, которая была беременна шесть раз, и каждый из этих случаев заканчивался выкидышем. Ее седьмая и последняя беременность завершилась рождением живого ребенка. Если мы отнесемся к этой информации с сочувствием, то, конечно, будем тронуты историей, которую она рассказывает.

За каждой строкой в датасете NSFG стоит человек, честно ответивший на множество сложных вопросов личного характера. Мы можем использовать эти данные для ответа на статистические вопросы о семейной жизни, репродукции и здоровье. При этом мы обязаны считаться с людьми, которых эти данные описывают, и проявлять к ним уважение и благодарность.

Глоссарий

В конце каждой главы книги приводится глоссарий терминов, которые в ней были определены.

Бытующее мнение (anecdotal evidence)

Данные, собранные неофициально на основе небольшого числа отдельных случаев и часто без применения систематической выборки.

Поперечное исследование (cross-sectional study)

Исследование, в ходе которого собираются данные о репрезентативной выборке из популяции за единичный момент или интервал времени.

Цикл (cycle)

Один интервал сбора данных в исследовании, где данные собираются за несколько интервалов времени.

Генеральная совокупность, или популяция (population)

Вся группа лиц или предметов, которые являются объектом исследования.

Выборка (sample)

Подмножество генеральной совокупности, часто отбираемое случайным образом.

Респонденты (respondents)

Люди, которые участвуют в опросе и отвечают на вопросы.

Репрезентативный (representative)

Выборка является репрезентативной, если она сходна с генеральной совокупностью по тем параметрам, которые важны для целей исследования.

Стратифицированный (stratified)

Выборка является стратифицированной, если в нее намеренно избыточно включены представители некоторых групп. Это делается, как правило, для того, чтобы набрать количество членов, достаточное для получения достоверных выводов.

Избыточная выборка (oversampled)

Группа является избыточной выборкой, если ее члены имеют повышенную вероятность попадания в выборку.

Переменная (variable)

В данных опроса переменная представляет собой совокупность ответов на вопросы или вычисленных на их основе значений.

Кодбук (codebook)

Документ, описывающий переменные датасета и содержащий иную информацию о данных.

Перекодированная переменная (recode)

Переменная, вычисленная на основе других переменных в датасете.

Исходные, или сырые, данные (raw data)

Данные, не прошедшие обработку после сбора.

Очистка данных (data cleaning)

Процесс выявления и исправления ошибок в датасете, работы с пропущенными значениями и вычисления перекодированных переменных.

Статистический показатель (statistic)

Значение, которое описывает или обобщает некоторое свойство выборки.

Стандартное отклонение (standard deviation)

Статистический показатель, количественно оценивающий разброс данных вокруг среднего значения.

Упражнения

Упражнения этой главы требуют использования файла NSFG о беременностях.

Упражнение 1.1

В таблице `preg` выберите столбец `birthord`, выведите на экран количество значений и сравните с результатами, опубликованными в кодбуке для 6-го цикла опроса NSFG по беременностям (https://oreil.ly/M2hDe).

Упражнение 1.2

Создайте новый столбец с именем `totalwgt_kg`, который содержит вес при рождении в килограммах (килограмм — это примерно 2.2 фунта). Для нового столбца вычислите среднее значение и стандартное отклонение.

Упражнение 1.3

Каковы продолжительности беременностей у респондентки с `caseid` 2298?

Каков был вес при рождении первого ребенка респондентки с `caseid` 5013? Подсказка: в запросе можно использовать оператор `and` для проверки нескольких условий.

ГЛАВА 2

Распределение данных

В этой главе вы познакомитесь с одной из самых фундаментальных идей статистики — распределением. Начнем с таблиц частот, которые содержат уникальные значения из датасета и количество повторений каждого из них. Мы используем такие таблицы для анализа данных Национального исследования роста семьи (NSFG). Также попрактикуемся в поиске экстремальных или ошибочных значений, называемых выбросами, и рассмотрим способы работы с ними.

Таблицы частот

Одним из способов описания переменной является **таблица частот**, которая содержит значения переменной и их **частоты**, то есть количество появления каждого значения. Такое описание называется **распределением** переменной.

Для представления распределений мы будем использовать библиотеку `empiricaldist`[1]. В данном контексте «эмпирический» означает, что распределения основаны на данных, а не на математических моделях. Библиотека `empiricaldist` предоставляет класс `FreqTab`, который можно использовать для вычисления и графического отображения таблиц частот. Импортируем его следующим образом:

```
from empiricaldist import FreqTab
```

Чтобы продемонстрировать, как этот класс работает, начнем с небольшого списка значений:

```
t = [1.0, 2.0, 2.0, 3.0, 5.0]
```

Класс `FreqTab` предоставляет метод `from_seq`, который принимает на вход последовательность и создает объект класса `FreqTab`:

```
ftab = FreqTab.from_seq(t)
ftab
```

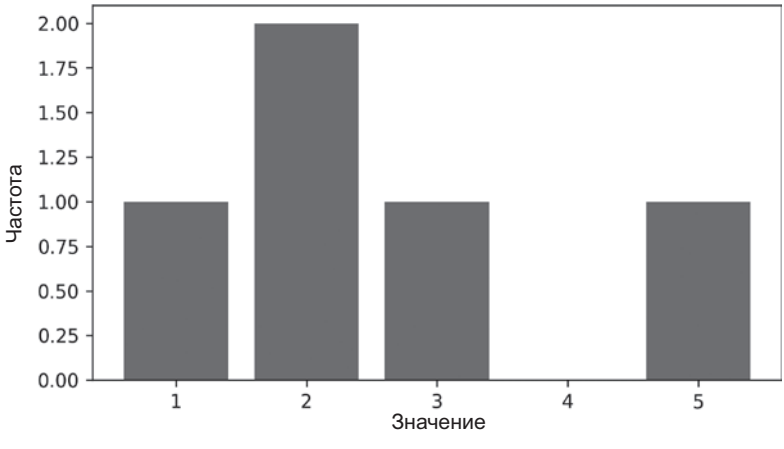
[1] От англ. empirical distributions — «эмпирические распределения». — *Примеч. пер.*

	частота
1.0	1
2.0	2
3.0	1
5.0	1

Объект `FreqTab` — это разновидность `Series` из Pandas, в котором содержатся значения и их частоты. В этом примере значение `1.0` соответствует частоте 1, значение `2.0` — частоте 2 и т. д.

`FreqTab` также предоставляет метод `bar`, отображающий таблицу частот в виде столбчатой диаграммы:

```
ftab.bar()
decorate(xlabel="Значение", ylabel="Частота")
```



Поскольку класс `FreqTab` является производным от `Series` из Pandas, можно воспользоваться оператором квадратных скобок для поиска значения и получения его частоты:

```
ftab[2.0]
```

```
2
```

Но, в отличие от `Series`, для поиска значений объект `FreqTab` также можно вызывать как функцию:

```
ftab(2.0)
```

```
2
```

Если попытаться найти значение, которого нет в объекте `FreqTab`, то вызов функции вернет `0`:

```
ftab(4.0)
```

```
0
```

Объект `FreqTab` имеет атрибут `qs`, который содержит массив значений. `qs` здесь означает «величины» (quantities), хотя, строго говоря, не все значения являются количественными величинами:

```
ftab.qs
```

```
array([1., 2., 3., 5.])
```

Во `FreqTab` также есть атрибут `fs`, который содержит массив частот (frequencies):

```
ftab.fs
```

```
array([1, 2, 1, 1])
```

Класс `FreqTab` предоставляет метод `items`, который можно использовать для циклического прохождения пар «величина — частота»:

```
for x, freq in ftab.items():
    print(x, freq)
```

```
1.0 1
2.0 2
3.0 1
5.0 1
```

В дальнейшем вы познакомитесь и с другими методами класса `FreqTab`.

Распределения датасета NSFG

Когда вы приступаете к работе с новым датасетом, я советую исследовать переменные, которые вы планируете использовать по одной за раз; и хорошая практика — начать с просмотра таблиц частот.

В качестве примера рассмотрим данные из датасета NSFG. В предыдущей главе мы загрузили его, считали в объект Pandas `DataFrame` и очистили некоторые переменные. Код, который мы использовали для загрузки и очистки данных, находится в модуле под названием `nsfg.py`. Инструкции по его установке приведены в Jupyter-блокноте к этой главе.

Чтобы импортировать этот модуль и считать данные о беременностях, выполним следующую команду:

```
from nsfg import read_fem_preg

preg = read_fem_preg()
```

В примерах этой главы мы сосредоточимся на беременностях, которые закончились рождением живого ребенка.

Чтобы выбрать строки, в которых переменная `outcome` равна 1, можно использовать метод `query`:

```
live = preg.query("outcome == 1")
```

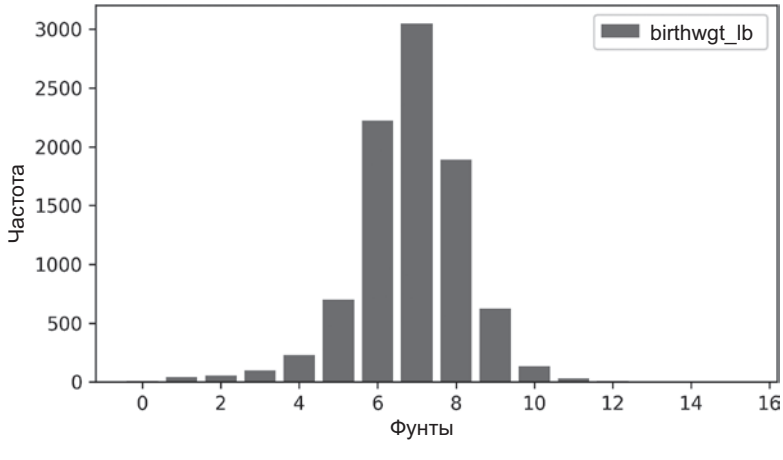
В строке, передаваемой в `query`, имена переменных, такие как `outcome`, ссылаются на имена столбцов в объекте `DataFrame`. Она также может содержать операторы, подобные `==`, и операнды, такие как `1`.

Теперь, чтобы подсчитать, сколько раз каждая величина появляется в переменной `birthwgt_lb`, являющейся частью веса при рождении в фунтах, можно воспользоваться методом `FreqTab.from_seq`. Аргумент `name` присваивает объекту `FreqTab` имя, которое используется в качестве метки при построении графиков:

```
ftab_lb = FreqTab.from_seq(live["birthwgt_lb"], name="birthwgt_lb")
```

Вот как выглядит это распределение:

```
ftab_lb.bar()
decorate(xlabel="Фунты", ylabel="Частота")
```



Глядя на него, первое, что бросается в глаза, — это форма, которая напоминает знаменитую колоколообразную кривую, формально называемую нормальным распределением или распределением Гаусса. Другой примечательной особенностью этого распределения является мода (mode) — значение, которое встречается чаще всего. Чтобы найти моду, можно использовать метод `idxmax`, который находит величину, соответствующую самой высокой частоте:

```
ftab_lb.idxmax()
```

```
7.0
```

Класс `FreqTab` имеет метод `mode`, который дает тот же результат:

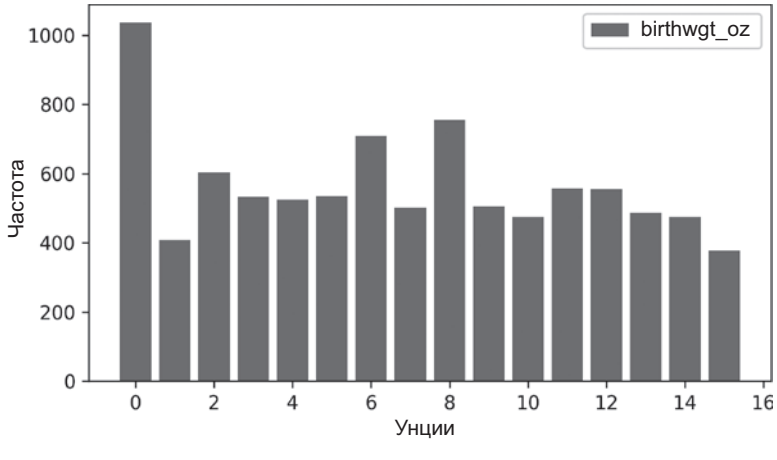
```
ftab_lb.mode()
```

```
7.0
```

В данном распределении мода составляет 7 фунтов.

В качестве еще одного примера создадим таблицу частот переменной `birthwgt_oz`, которая является частью веса при рождении в унциях:

```
ftab_oz = FreqTab.from_seq(live["birthwgt_oz"], name="birthwgt_oz")
ftab_oz.bar()
decorate(xlabel="Унции", ylabel="Частота")
```



Поскольку природа не осведомлена о фунтах и унциях, то можно было ожидать, что все значения `birthwgt_oz` будут равновероятными — то есть полученное распределение должно быть **равномерным** (uniform). Но оказывается, что `0` встречается чаще других величин, а `1` и `15` — реже; это говорит о том, что респонденты

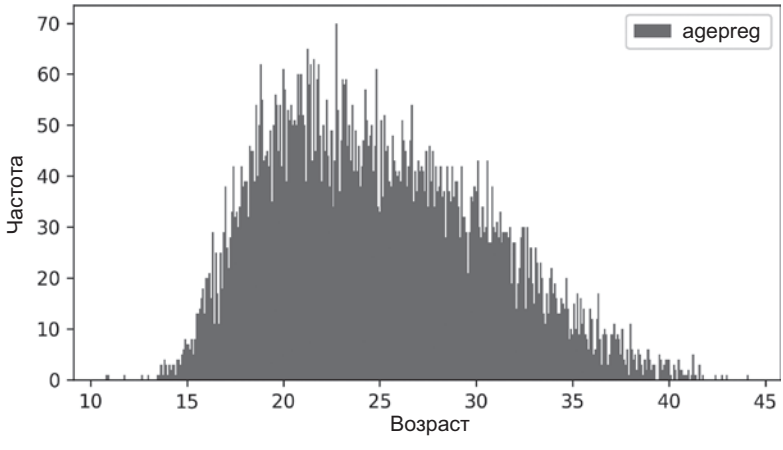
регулярно округляют те значения веса при рождении, которые близки целому числу фунтов.

Еще один пример: рассмотрим таблицу частот переменной `agepreg`, содержащей возраст матери в конце беременности:

```
ftab_age = FreqTab.from_seq(live["agepreg"], name="agepreg")
```

В датасете NSFG возраст указывается в годах и месяцах, поэтому здесь больше уникальных значений, чем в других рассмотренных нами распределениях. Поэтому, чтобы столбцы не слишком сильно перекрывали друг друга, передадим методу `bar` именованный аргумент `width=0.1`, регулирующий их ширину:

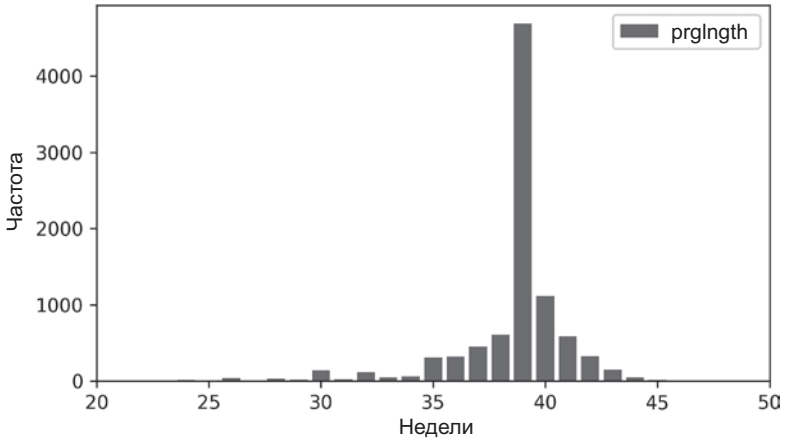
```
ftab_age.bar(width=0.1)
decorate(xlabel="Возраст", ylabel="Частота")
```



Распределение только отдаленно напоминает колокол и при этом **скошено** вправо, то есть его хвост тянется вправо дальше, чем влево.

В заключение взглянем на таблицу частот переменной `prglngth`, представляющей собой продолжительность беременности в неделях. Аргумент `xlim` устанавливает пределы по оси x в диапазоне от 20 до 50 недель: значений за пределами этого диапазона не так много, и, вероятно, это ошибки.

```
ftab_length = FreqTab.from_seq(live["prglngth"], name="prglngth")
ftab_length.bar()
decorate(xlabel="Недели", ylabel="Частота", xlim=[20, 50])
```



Срок 39 недель встречается заметно чаще других. Также левый хвост распределения длиннее правого: ранние роды случаются регулярно, но беременность редко длится дольше 43 недель, и в таких случаях врачи часто принимают меры.

Выбросы

Глядя на таблицы частот, легко определить форму распределения и наиболее часто встречающиеся величины, но редкие величины не всегда видны. Прежде чем двигаться дальше, стоит проверить наличие **выбросов** — экстремальных значений, которые могут быть либо ошибкой измерений или записи, либо точным отражением редких событий.

Для выявления выбросов функция ниже принимает объект `FreqTab` и целое число `n`, чтобы с использованием индекса среза выбрать `n` наименьших (smallest) величин и их частот:

```
def smallest(ftab, n=10):
    return ftab[:n]
```

Вот 10 наименьших значений в таблице частот переменной `prglngth`:

```
smallest(ftab_length)
prglngth
```

```
0     1
4     1
9     1
13    1
17    2
18    1
19    1
20    1
```

```
21    2
22    7
Name: prglngth, dtype: int64
```

Поскольку мы отобрали случаи рождения живых детей, то продолжительность беременности менее 10 недель, безусловно, является ошибочной. Скорее всего, причина в том, что результат был указан неверно. Продолжительность беременности более 30 недель, судя по всему, является корректной. О значениях между 10 и 30 неделями трудно сказать определенно: какие-то из них, скорее всего, являются ошибочными, но другие корректно отражают случаи преждевременных родов.

Следующая функция выделяет самые большие (largest) значения из объекта `FreqTab`:

```
def largest(ftab, n=10):
    return ftab[-n:]
```

Вот самые продолжительные сроки беременности в датасете:

```
largest(ftab_length)
prglngth
40    1116
41     587
42     328
43     148
44      46
45      10
46       1
47       1
48       7
50       2
Name: prglngth, dtype: int64
```

Опять же, некоторые из этих значений, возможно, ошибочны. Большинство врачей рекомендуют стимуляцию родов, если срок беременности превышает 41 неделю. Поэтому маловероятно, что срок в 50 недель достоверен. Но тем не менее не существует четкой границы между очевидно ошибочными значениями и правильными значениями для редких случаев.

Наилучший способ обработки выбросов диктуется знанием предметной области, то есть информацией о том, откуда берутся данные и что они означают. Также он зависит от того, какой анализ вы планируете выполнить.

В данном примере нас интересует вопрос, появляются первенцы на свет раньше или позже других детей. Так что мы будем использовать статистические показатели, которые не сильно искажаются присутствием малого количества некорректных значений.

Первые дети

Сравним теперь распределение продолжительности беременности для женщин, рожающих впервые, и всех остальных. Для выбора строк, представляющих первых (firsts) и других (others) детей, можно использовать метод `query`:

```
firsts = live.query("birthord == 1")
others = live.query("birthord != 1")
```

Создадим также объекты `FreqTab` для продолжительности беременности женщин из каждой группы:

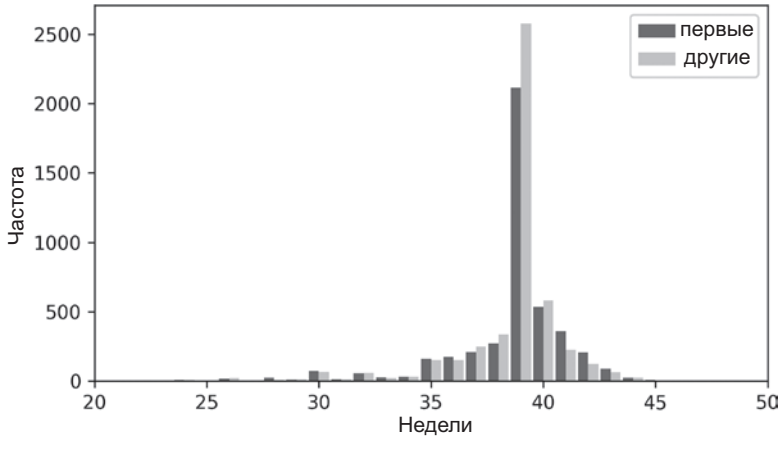
```
ftab_first = FreqTab.from_seq(firsts["prglngth"], name="первые")
ftab_other = FreqTab.from_seq(others["prglngth"], name="другие")
```

Следующая функция отображает две частотные диаграммы на одном графике:

```
def two_bar_plots(ftab1, ftab2, width=0.45):
    ftab1.bar(align="edge", width=-width)
    ftab2.bar(align="edge", width=width, alpha=0.5)
```

Вот как они выглядят:

```
two_bar_plots(ftab_first, ftab_other)
decorate(xlabel="Недели", ylabel="Частота", xlim=[20, 50])
```



Нет очевидной разницы в форме распределений или в наличии выбросов. Похоже, что на 39-й неделе рождается больше вторых и последующих детей. Но в нашем датасете таких детей больше, поэтому не следует эти количественные показатели сравнивать непосредственно:

```
firsts["prglngth"].count(), others["prglngth"].count()
```

```
(4413, 4735)
```

Если сравнить средние значения обоих распределений, то кажется, что первые дети в среднем рождаются немного позже:

```
first_mean = firsts["prglngth"].mean()
other_mean = others["prglngth"].mean()
first_mean, other_mean
```

```
(38.60095173351461, 38.52291446673706)
```

Но разница составляет всего 0.078 недели, что равняется приблизительно 13 часам:

```
diff = first_mean - other_mean
diff, diff * 7 * 24
```

```
(0.07803726677754952, 13.11026081862832)
```

Существует несколько возможных причин этого очевидного различия:

- Возможно, существует реальная разница в средней продолжительности беременности при рождении первых и других детей.
- Явная разница, наблюдаемая в этом датасете, может быть результатом предвзятости в процессе составления выборки, то есть при отборе респондентов для опроса.
- Видимая разница может быть результатом ошибки измерения. Например, продолжительность беременности, сообщаемая респондентками, может быть точнее для первых или последующих детей.
- Очевидная разница может быть результатом случайных вариаций в процессе составления выборки.

В последующих главах мы рассмотрим эти возможные объяснения более подробно, но пока будем считать, что наш результат верен: в этом датасете у двух групп наблюдается небольшая разница в продолжительности беременности.

Размер эффекта

Разницу, которую мы описали выше, иногда называют *эффектом*. Существует несколько способов оценить величину эффекта количественно. Самый простой — указать разницу в абсолютных величинах: в данном примере разница составляет 0.078 недели.

Другой способ — указать разницу в относительных величинах. Например, мы могли бы сказать, что первая беременность в среднем длится на 0.2 % дольше других:

```
diff / live["prglngth"].mean() * 100
```

```
0.20237586646738304
```

Еще один вариант — сообщить **стандартизированный** размер эффекта, представляющий собой статистический показатель, предназначенный для количественной оценки размера эффекта таким способом, чтобы можно было сравнивать между собой различные величины и группы.

Стандартизация означает, что мы выражаем разницу как величину, кратную стандартному отклонению. Поэтому может возникнуть соблазн написать, например, так:

```
diff / live["prglngth"].std()
```

```
0.028877623375210403
```

Но обратите внимание, что для вычисления стандартного отклонения мы использовали обе группы. Если группы существенно различаются между собой, то при их объединении стандартное отклонение становится больше, чем у любой из групп по отдельности. В таком случае может показаться, что размер эффекта мал.

Альтернативой является использование стандартного отклонения только одной из групп, но неясно, какой именно. Еще мы могли бы взять среднее значение двух стандартных отклонений, но если группы различаются по размеру, то это придало бы слишком большой вес одной группе и недостаточный — другой.

Распространенное решение — использование **объединенного стандартного отклонения** (pooled standard deviation), которое представляет собой квадратный корень из объединенной дисперсии. Последняя равна взвешенной сумме дисперсий в отдельных группах. Чтобы вычислить этот показатель, начнем с дисперсий:

```
group1, group2 = firsts["prglngth"], others["prglngth"]

v1, v2 = group1.var(), group2.var()
```

Вот взвешенная сумма с размерами групп в качестве весов:

```
n1, n2 = group1.count(), group2.count()
pooled_var = (n1 * v1 + n2 * v2) / (n1 + n2)
```

Наконец, вот объединенное стандартное отклонение:

```
np.sqrt(pooled_var)
```

```
2.7022108144953862
```

Объединенное стандартное отклонение находится между стандартными отклонениями двух групп:

```
firsts["prglngth"].std(), others["prglngth"].std()
```

```
(2.7919014146687204, 2.6158523504392375)
```

Стандартизированный размер эффекта, в котором используется объединенное стандартное отклонение, называется **размером эффекта Коэна**. Вот функция, которая его вычисляет:

```
def cohen_effect_size(group1, group2):
    diff = group1.mean() - group2.mean()

    v1, v2 = group1.var(), group2.var()
    n1, n2 = group1.count(), group2.count()
    pooled_var = (n1 * v1 + n2 * v2) / (n1 + n2)

    return diff / np.sqrt(pooled_var)
```

А вот размер эффекта для разницы в средней продолжительности беременности:

```
cohen_effect_size(firsts["prglngth"], others["prglngth"])
```

```
0.028879044654449834
```

В данном примере разница составляет 0.029 стандартного отклонения, это мало. Для сравнения: разница в росте между мужчинами и женщинами составляет около 1.7 стандартного отклонения.

Представление результатов

Мы рассмотрели несколько способов описания разницы в продолжительности беременности (если таковая имеется) при рождении первых и последующих детей. Как представить полученные результаты?

Ответ зависит от того, кто задает вопрос. Ученого может интересовать любой (реальный) эффект, каким бы незначительным он ни был. Врача могут интересовать только те эффекты, которые **практически значимы**, то есть различия, существенные для практики. Беременную женщину могут заинтересовать результаты, имеющие к ней отношение, например вероятность преждевременных или поздних родов.

Способ представления результатов также зависит от ваших целей. Если вы хотите продемонстрировать важность эффекта, можете выбрать сводную статистику, подчеркивающую различия. Если хотите успокоить пациента — статистические показатели, которые отражают различия в соответствующем контексте.

В своих решениях вы также, несомненно, должны руководствоваться профессиональной этикой. Быть убедительным — это нормально: статистические отчеты и иллюстрации *следует* создавать так, чтобы они ясно описывали ситуацию. Но вы также должны прилагать максимум усилий, чтобы ваши отчеты были честными и признавали наличие неопределенности и ограничений.

Глоссарий

Распределение (distribution)

Набор значений и частота появления каждого значения в датасете.

Таблица частот (frequency table)

Сопоставление значений и частот.

Частота (frequency)

Количество появлений значения в выборке.

Скошенный (skewed)

Распределение называется скошенным, если оно асимметрично. При этом экстремальные величины распространяются дальше в одном направлении, чем в другом.

Мода (mode)

Наиболее часто встречающаяся в выборке величина или одна из наиболее часто встречающихся величин.

Равномерное распределение (uniform distribution)

Распределение, в котором все величины имеют одинаковую частоту.

Выброс (outlier)

Экстремальная величина в распределении.

Стандартизированный (standardized)

Статистический показатель считается стандартизированным, если он выражен в единицах, сопоставимых между различными датасетами и предметными областями.

Объединенное стандартное отклонение (pooled standard deviation)

Статистический показатель, для вычисления общего стандартного отклонения объединяющий данные из двух или более групп.

Размер эффекта Коэна (Cohen's effect size)

Стандартизированный статистический показатель, количественно выражающий разницу между средними значениями двух групп.

Практически значимый (practically significant)

Эффект является практически значимым, если он достаточно велик, чтобы его имело смысл учитывать на практике.

Упражнения

Для выполнения упражнений этой главы загрузите файл данных NSFG с ответами женщин-респондентов, по одной строке на каждую респондентку. Инструкции по загрузке данных и кодбука приведены в Jupyter-блокноте для этой главы.

Модуль `nsfg.py` содержит функцию, которая считывает файл ответов респонденток, очищает некоторые переменные и возвращает объект `DataFrame`:

```
from nsfg import read_fem_resp

resp = read_fem_resp()
resp.shape
```

```
(7643, 3092)
```

Данный `DataFrame` содержит 3092 столбца, но мы будем использовать лишь некоторые из них.

Упражнение 2.1

Начнем с переменной `totincr`, которая содержит общий доход семьи респондентки, закодированный значением от 1 до 14. Обратитесь к кодбуку для файла респондентов, чтобы узнать, какой уровень дохода соответствует каждому значению.

Создайте объект `FreqTab` для распределения этой переменной и отобразите его при помощи столбчатой диаграммы.

Упражнение 2.2

Составьте таблицу частот колонки `parity`, содержащей количество детей, рожденных каждой респонденткой. Как бы вы описали форму этого распределения?

Используйте функцию `largest`, чтобы найти наибольшие значения переменной `parity`. Существуют ли значения, которые вам кажутся ошибочными?

Упражнение 2.3

Выясним, рожают ли женщины с более высоким доходом больше детей. Используйте метод `query`, чтобы выбрать респонденток с самым высоким доходом (уровень 14). Постройте диаграмму частот переменной `parity` только для респонденток с высоким доходом.

Сравните среднее значение переменной `parity` для респонденток с высоким уровнем дохода и всех остальных респонденток.

Рассчитайте размер эффекта Коэна для этого различия. Как он соотносится с разницей в продолжительности беременности для первых и других детей?

Означают ли эти результаты, что у людей с более высоким доходом больше детей, или вы можете предложить другое объяснение имеющемуся различию?

ГЛАВА 3

Функция вероятности

В предыдущей главе мы работали с распределениями с использованием объектов класса `FreqTab`, хранящего множество значений и их частоту, то есть сколько раз каждое значение встречается в выборке. В этой главе вы познакомитесь с другим способом описания распределения — функцией вероятности (ФВ).

Для представления ФВ мы будем использовать объекты класса `Pmf`, содержащего набор значений и их вероятности. Объекты `Pmf` окажутся полезными для вычисления среднего значения и дисперсии распределения, а также коэффициента асимметрии, показывающего, скошено распределение влево или вправо. И наконец, мы рассмотрим «парадокс проверки» (inspection paradox), проявляющийся в том, что выборка может дать необъективную оценку распределения.

Функция вероятности

Объект класса `Pmf` похож на `FreqTab`, только содержит вероятности, а не частоты. Поэтому один из способов создать `Pmf` — это задействовать объект `FreqTab`. Например, рассмотрим `FreqTab`, представляющий распределение значений в короткой последовательности:

```
from empiricaldist import FreqTab

ftab = FreqTab.from_seq([1, 2, 2, 3, 5])
ftab
```

	частота
1	1
2	2
3	1
5	1

Сумма частот равна размеру исходного ряда:

```
n = ftab.sum()
n
```

```
5
```

Если мы разделим частоту на n, то полученное значение будет представлять собой пропорцию, а не количество:

```
pmf = ftab / n
pmf
```

	вероятность
1	0.2
2	0.4
3	0.2
5	0.2

Результат указывает на то, что 20 % значений в последовательности равны 1, 40 % — 2 и т. д.

Эти пропорции также можно считать вероятностями в следующем смысле. Если из исходной последовательности выбрать случайное значение, то вероятность получить значение 1 равна 0.2, значение 2 — 0.4 и т. д.

Поскольку мы делили все частоты на n, то сумма вероятностей равна 1, это означает, что полученное распределение **нормированное** (normalized):

```
pmf.sum()
```

```
1.0
```

Нормированный объект `FreqTab` представляет собой **функцию вероятности**[1], названную так потому, что вероятности, связанные с дискретными значениями, также называют «вероятностными мерами»[2].

В библиотеке `empiricaldist` реализован класс `Pmf`, представляющий функцию вероятности. Поэтому вместо создания объекта `FreqTab` и последующей его нормировки можно сразу создать объект `Pmf`:

```
from empiricaldist import Pmf

pmf = Pmf.from_seq([1, 2, 2, 3, 5])
pmf
```

[1] Или «функция распределения масс (вероятностей)», от англ. probability mass function, PMF. — *Примеч. пер.*

[2] Дословно «вероятностная масса», от англ. probability mass. — *Примеч. пер.*

	вероятность
1	0.2
2	0.4
3	0.2
5	0.2

Объект `Pmf` нормированный, поэтому суммарная вероятность равна 1:

```
pmf.sum()
```

```
1.0
```

Объекты `Pmf` и `FreqTab` во многом похожи. Чтобы найти вероятность, связанную с каким-либо значением, можно использовать оператор квадратных скобок:

```
pmf[2]
```

```
0.4
```

Или можно использовать круглые скобки для вызова `Pmf` как функции:

```
pmf(2)
```

```
0.4
```

Чтобы присвоить значению вероятность, используйте оператор квадратных скобок:

```
pmf[2] = 0.2
pmf(2)
```

```
0.2
```

Вы можете изменить объект `Pmf`, увеличив вероятность, связанную с некоторым значением:

```
pmf[2] += 0.3
pmf[2]
```

```
0.5
```

Или можете умножить вероятность на коэффициент:

```
pmf[2] *= 0.5
pmf[2]
```

```
0.25
```

После изменения результирующий объект `Pmf` может оказаться ненормированным, то есть вероятности в сумме могут больше не равняться 1:

```
pmf.sum()
```

```
0.8500000000000001
```

Метод `normalize` перенормирует `Pmf` путем деления на сумму вероятностей и возвращает эту сумму:

```
pmf.normalize()
```

```
0.8500000000000001
```

Объекты `Pmf` имеют метод копирования `copy`, поэтому можно создавать и изменять копию, не меняя оригинальный объект:

```
pmf.copy()
```

	вероятность
1	0.235294
2	0.294118
3	0.235294
5	0.235294

Как и `FreqTab`, объект `Pmf` имеет атрибут `qs` для доступа к величинам и атрибут `ps` для доступа к вероятностям.

В нем также есть метод `bar`, отображающий `Pmf` в виде столбчатой диаграммы, и метод `plot`, изображающий распределение в виде линейного графика.

Сводные характеристики функции вероятности

В разделе «Сводные статистические показатели» на с. 28 мы вычисляли среднее значение выборки, складывая все элементы и деля сумму на количество элементов. Вот простой пример:

```
seq = [1, 2, 2, 3, 5]

n = len(seq)
mean = np.sum(seq) / n
mean
```

```
2.6
```

Теперь предположим, что у нас есть функция вероятности значений в последовательности:

```
pmf = Pmf.from_seq(seq)
```

Имея объект `Pmf`, мы по-прежнему можем вычислить среднее значение, но процедура будет другой: нужно перемножить вероятности и соответствующие величины, а потом сложить полученные результаты:

```
mean = np.sum(pmf.ps * pmf.qs)
mean
```

```
2.6
```

Обратите внимание, что здесь нам *не* нужно делить на `n`, потому что мы уже сделали это, когда нормировали `Pmf`. У объектов `Pmf` есть метод `mean`, который делает то же самое:

```
pmf.mean()
```

```
2.6
```

Имея под рукой `Pmf`, мы можем вычислить дисперсию, посчитав отклонение (deviation) каждой величины от среднего значения:

```
deviations = pmf.qs - mean
```

Затем умножаем квадраты отклонений на вероятности и складываем полученные результаты:

```
var = np.sum(pmf.ps * deviations**2)
var
```

```
1.84
```

Метод `var` дает тот же результат:

```
pmf.var()
```

```
1.84
```

Зная дисперсию, можно вычислить стандартное отклонение обычным способом:

```
np.sqrt(var)
```

```
1.3564659966250536
```

Или воспользоваться методом `std` с тем же результатом:

```
pmf.std()
```

```
1.3564659966250536
```

Класс `Pmf` также предоставляет метод `mode`, находящий значение, связанное с наибольшей вероятностью:

```
pmf.mode()
```

```
2
```

В дальнейшем вы познакомитесь и с другими методами, но для начала этих будет достаточно.

Парадокс размера группы

В качестве примера того, для чего могут пригодиться объекты `Pmf`, рассмотрим феномен, который я называю «парадоксом размера группы» (class size paradox).

Во многих американских колледжах и университетах соотношение студентов и преподавателей равняется примерно 10:1. Но студентов часто удивляет то, что во многих группах обучается больше 10 человек, иногда заметно больше. Такое несоответствие объясняется следующими двумя причинами:

- Студенты обычно выбирают четыре или пять курсов в семестр, но преподаватели зачастую ведут только один или два курса.
- Количество студентов в группе малочисленного курса невелико, а в группе многочисленного — велико.

Первый факт очевиден — по крайней мере, когда на него обращают внимание; второй требует пояснений. Рассмотрим пример. Предположим, что колледж предлагает 65 курсов в течение одного семестра, и нам задано количество курсов в каждом из следующих диапазонов размера группы:

```
ranges = pd.interval_range(start=5, end=50, freq=5, closed="left")
ranges.name = "class size"

data = pd.DataFrame(index=ranges)
data["count"] = [8, 8, 14, 4, 6, 12, 8, 3, 2]
data
```

class size (размер группы)	count (количество)
[5, 10)	8
[10, 15)	8
[15, 20)	14
[20, 25)	4
[25, 30)	6

class size (размер группы)	count (количество)
[30, 35)	12
[35, 40)	8
[40, 45)	3
[45, 50)	2

Функция Pandas `interval_range` создает объект `Index`, где каждая метка представляет диапазон значений. Запись `[5, 10)` означает, что `5` включено в интервал, а `10` — нет. Поскольку мы не знаем размеры групп в каждом интервале, то предположим, что каждая группа находится в середине своего диапазона:

```
sizes = ranges.left + 2
sizes
```

```
Index([7, 12, 17, 22, 27, 32, 37, 42, 47], dtype='int64')
```

Теперь создадим объект `Pmf`, представляющий распределение размера группы. Поскольку мы знаем размеры и их частоту, то можем создать `Pmf` напрямую, передав в качестве аргументов количество (counts), размеры (sizes) и название (name). Результатом нормирования нового объекта `Pmf` будет сумма значений:

```
counts = data["count"]
actual_pmf = Pmf(counts, sizes, name="фактическое")
actual_pmf.normalize()
```

```
65
```

Если вы спросите сотрудников колледжа о среднем размере группы, то получите в ответ среднее значение этого распределения, которое составляет 23.7:

```
actual_pmf.mean()
```

```
23.692307692307693
```

Но если вы проведете опрос среди студентов, интересуясь численностью их групп, и вычислите среднее значение, то оно окажется больше. Посмотрим, насколько больше.

Следующая функция принимает на входе объект `Pmf` фактического размера групп и создает новый объект `Pmf`, соответствующий размеру групп в представлении студентов. Численности групп в обоих распределениях одинаковы, но вероятности во втором распределении умножаются на размер, потому что в группе размера `x` присутствует `x` студентов-наблюдателей. Таким образом, вероятность наблюдения группы пропорциональна ее размеру:

```
def bias(pmf, name):
    # умножить каждую вероятность на размер группы
    ps = pmf.ps * pmf.qs

    # создать новый объект Pmf и нормировать его
    new_pmf = Pmf(ps, pmf.qs, name=name)
    new_pmf.normalize()
    return new_pmf
```

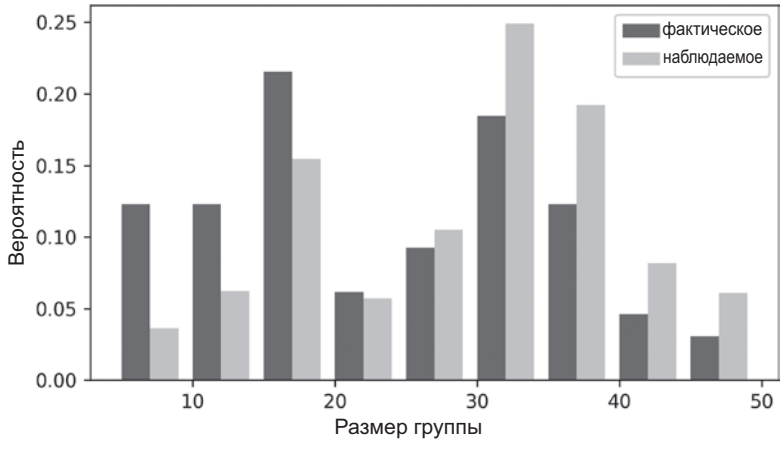
Теперь можно посчитать наблюдаемое студентами смещенное (biased) распределение `Pmf`:

```
observed_pmf = bias(actual_pmf, name="наблюдаемое")
```

Вот как выглядят эти два распределения:

```
from thinkstats import two_bar_plots

two_bar_plots(actual_pmf, observed_pmf, width=2)
decorate(xlabel="Размер группы", ylabel="Вероятность")
```



В наблюдаемом распределении малочисленных групп меньше, а многочисленных — больше. Смещенное (biased) среднее значение равно 29.1, что почти на 25 % выше фактического среднего:

```
observed_pmf.mean()
```

```
29.123376623376622
```

Также можно выполнить обратную операцию. Предположим, вы хотите найти распределение по размеру группы в колледже, но не можете получить надежные

данные. Один из вариантов — взять случайную выборку студентов и спросить их, сколько студентов посещает их группы.

Результат будет смещенным по причинам, которые мы только что рассмотрели, но его можно использовать для оценки фактического распределения. Вот функция, которая устраняет смещение `Pmf` путем деления вероятности на размер группы:

```
def unbias(pmf, name):
    # разделить каждую вероятность на размер группы
    ps = pmf.ps / pmf.qs

    new_pmf = Pmf(ps, pmf.qs, name=name)
    new_pmf.normalize()
    return new_pmf
```

Вот что мы получаем:

```
debiased_pmf = unbias(observed_pmf, "debiased")
debiased_pmf.mean()
```

```
23.692307692307693
```

Среднее значение `Pmf` после устранения смещения совпадает со средним значением фактического распределения, с которого мы начинали.

Если вас заинтересовал этот пример, то вам также может понравиться глава 2 из моей книги *Probably Overthinking It* (University of Chicago Press, 2023), в которой приводится этот и несколько других примеров так называемого парадокса проверки.

Данные NSFG

В предыдущей главе мы строили частотные диаграммы продолжительности беременности при рождении первых и последующих детей. Но размеры этих групп различаются, поэтому мы не можем сравнивать такие таблицы частот непосредственно. Но, поскольку функции вероятности (ФВ) нормированные, их можно подвергать сравнению. Поэтому снова загрузим данные исследования NSFG и, чтобы построить распределения продолжительности беременности, создадим объекты `Pmf`.

В модуле `nsfg` реализована функция `read_nsfg_groups`, которая считывает данные, выбирает строки, представляющие живорождение (live), а затем разбивает их на соответствующие первенцам (firsts) и последующим (others) детям строки. Функция возвращает три `DataFrame`-объекта:

```
from nsfg import get_nsfg_groups

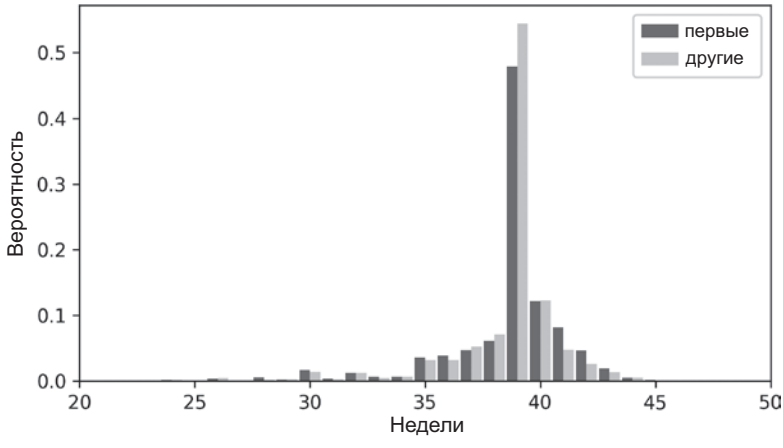
live, firsts, others = get_nsfg_groups()
```

Теперь можно воспользоваться переменными `firsts` и `others`, чтобы создать объекты `Pmf` для ФВ продолжительности беременности в каждой группе:

```
first_pmf = Pmf.from_seq(firsts["prglngth"], name="первые")
other_pmf = Pmf.from_seq(others["prglngth"], name="последующие")
```

Вот столбчатые диаграммы ФВ для первых и последующих детей:

```
two_bar_plots(first_pmf, other_pmf)
decorate(xlabel="Недели", ylabel="Вероятность", xlim=[20, 50])
```



Построив диаграммы ФВ вместо диаграмм частот, мы можем сравнить два распределения, не будучи введенными в заблуждение разницей в размерах выборок. Если посмотреть на рисунок, может показаться, что первые дети с меньшей вероятностью, чем последующие, появляются на свет вовремя (39-я неделя) и с большей вероятностью — позже срока (41-я и 42-я недели).

Дополнительная визуализация

Таблицы частот и функции вероятности полезны на этапе разведки данных для выявления закономерностей и взаимосвязей. Когда у вас сформируется понимание происходящего, создайте такую визуализацию, которая представит выявленные вами закономерности наиболее отчетливо.

Наибольшие различия в распределениях данных NSFG наблюдаются вблизи моды. Поэтому имеет смысл увеличить масштаб этой части графика и выбрать данные за 35–46-ю недели.

Когда объект `Pmf` вызывается как функция, можно задать на входе последовательность величин и получить на выходе последовательность вероятностей:

```
weeks = range(35, 46)
first_pmf(weeks)
```

```
array([0.03602991, 0.03897575, 0.04713347, 0.06163608, 0.4790392,
       0.12145932, 0.08157716, 0.04645366, 0.01971448, 0.00521187,
       0.00135962])
```

```
other_pmf(weeks)
```

```
array([0.03210137, 0.03146779, 0.05216473, 0.07074974, 0.54466737,
       0.12249208, 0.04794087, 0.02597677, 0.01288279, 0.00485744,
       0.00084477])
```

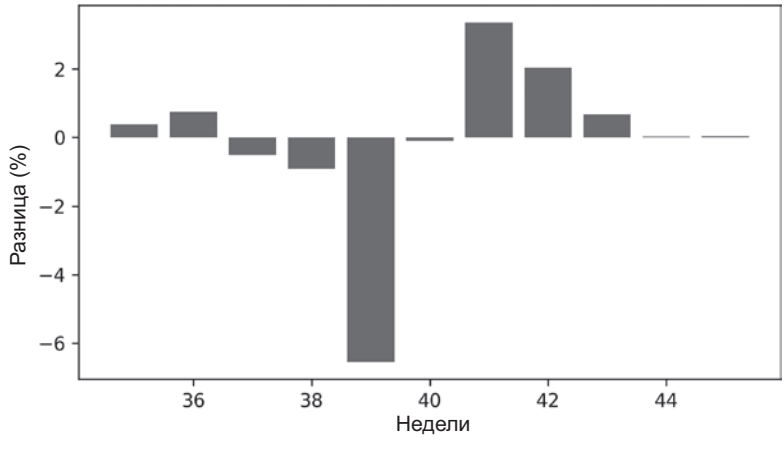
Таким образом, разницу в вероятностях можно вычислить следующим образом:

```
diffs = first_pmf(weeks) - other_pmf(weeks)
diffs
```

```
array([0.00392854, 0.00750796, -0.00503126, -0.00911366, -0.06562817,
       -0.00103276, 0.03363629, 0.02047689, 0.00683169, 0.00035443,
       0.00051485])
```

Вот как они выглядят после умножения на 100, чтобы выразить разницу в процентах:

```
plt.bar(weeks, diffs * 100)
decorate(xlabel="Недели", ylabel="Разница (%)")
```



Этот график делает картину более ясной: первые дети реже рождаются на 39-й неделе и несколько чаще — на 41-й и 42-й.

Наблюдая подобную закономерность в некоторой выборке, мы, однако, не можем с уверенностью утверждать, что она присутствует и в генеральной совокупности, — мы даже не знаем, увидим ли ее в другой выборке из той же популяции. Вернемся к этому вопросу позже, в главе 9.

Глоссарий

В этой главе не так много новых терминов, как в предыдущих главах:

Нормированный (normalized)
: Множество вероятностей нормированное, если эти вероятности в сумме дают 1.

Функция вероятности (probability mass function, PMF)
: Функция, описывающая распределение вероятностей путем установления соответствия каждой величины с ее вероятностью.

Упражнения

Для упражнений в этой главе используйте файл респондентов NSFG, в котором каждая строка соответствует одной респондентке. Инструкции по загрузке данных приведены в Jupyter-блокноте для этой главы.

В модуле `nsfg.py` реализована функция, которая считывает файл респондентов и возвращает объект `DataFrame`:

```
from nsfg import read_fem_resp

resp = read_fem_resp()
resp.shape
```

```
(7643, 3092)
```

Этот `DataFrame` содержит 7643 строки и 3092 столбца.

Упражнение 3.1

Выберите столбец `numbabes`, в котором записано «количество младенцев, родившихся живыми» у каждой респондентки. Создайте объект `FreqTab` и постройте частотную диаграмму значений в этом столбце. Убедитесь, что они соответствуют частотам, указанным в кодбуке. Есть ли какие-то специальные значения, которые следует заменить на `NaN`?

Теперь создайте объект `Pmf` и отобразите его в виде столбчатой диаграммы. Является ли распределение симметричным, скошенным влево или вправо?

Упражнение 3.2

Подобно тому как среднее значение определяет центральную точку в распределении, а дисперсия описывает количественно его разброс, существует еще один статистический показатель, называемый **асимметрией** (skewness), который указывает, является распределение скошенным влево или вправо.

Для имеющейся выборки асимметрию можно вычислить, сложив возведенные в куб отклонения и разделив полученную сумму на стандартное отклонение в кубе. Например, вот как рассчитать асимметрию переменной `numbabes`:

```
numbabes = resp["numbabes"].replace(97, np.nan)
```

```
deviations = numbabes - numbabes.mean()
skewness = np.mean(deviations**3) / numbabes.std(ddof=0) ** 3
skewness
```

```
1.7018914266755958
```

Положительное значение указывает на скошенность распределения вправо, а отрицательное — на скошенность влево.

Если у вас вместо последовательности значений есть объект `Pmf`, то асимметрию можно вычислить следующим образом:

1. Вычислите отклонение каждой величины в `Pmf` от среднего значения.
2. Возведите отклонения в куб, умножьте на вероятности, записанные в `Pmf`, и сложите полученные результаты.
3. Разделите полученную сумму на стандартное отклонение в третьей степени.

Напишите функцию `pmf_skewness`, которая принимает на входе объект `Pmf` и возвращает его асимметрию. Используйте свою функцию и объект `Pmf` переменной `numbabes` для вычисления асимметрии. Убедитесь, что полученный результат соответствует значению, вычисленному выше.

Упражнение 3.3

Явление, подобное парадоксу размера группы, будет наблюдаться, если вы проведете опрос среди детской аудитории и поинтересуетесь количеством детей в их семьях. Многодетные семьи с большей вероятностью будут представлены в вашей выборке, а бездетные семьи вообще не имеют шансов туда попасть.

В датафрейме `resp` выберите столбец `numkdhh`, содержащий количество детей в возрасте до 18 лет в домохозяйстве каждой респондентки. На основе значений этого столбца создайте объект `Pmf`.

Используйте функцию `bias` для вычисления распределения, которое бы мы увидели, если бы у детей респонденток спросили, сколько детей младше 18 лет (включая самих опрошенных) проживает в их семьях.

Постройте диаграммы фактического и смещенного распределений, вычислите их средние значения.

ГЛАВА 4

Интегральная функция распределения

https://t.me/it_boooks/2

Таблица частот и функция вероятности — наиболее распространенные способы представления распределений, но, как вы увидите в этой главе, у них есть ограничения. Альтернативой им является интегральная функция распределения (ИФР), которая пригодна для вычисления процентилей и особенно полезна для сравнения распределений.

Также в этой главе мы займемся вычислением статистических показателей на основе процентиля, чтобы количественно оценить положение, разброс и асимметрию распределения.

Процентиль и процентильный ранг

Если вы когда-либо проходили стандартизированный тест, то, вероятно, получали результаты в виде первичного балла (raw score) и **процентильного ранга**. В данном контексте процентильный ранг — это процент тестируемых, которые получили тот же балл, что и вы, или ниже. То есть нахождение «в 90-м процентиле» означает, что вы сдали экзамен так же или лучше, чем 90 % испытуемых.

Чтобы разобраться в процентилях и процентильных рангах, рассмотрим пример со скоростью бега. Несколько лет назад я участвовал в забеге James Joyce Ramble — состязании в беге на 10 километров, проходящем в Массачусетсе. После окончания забега я скачал его результаты, чтобы посмотреть на свое время в сравнении с временем других бегунов.

Инструкции по загрузке этих данных можно найти в Jupyter-блокноте для главы 4. В модуле `relay.py` реализована функция, которая считывает результаты и возвращает Pandas-объект `DataFrame`:

```
from relay import read_results

results = read_results()
results.head()
```

	Place	Div/Tot	Division	Guntime	Nettime	Min/Mile	MPH
0	1	1/362	M2039	30:43	30:42	4:57	12.121212
1	2	2/362	M2039	31:36	31:36	5:06	11.764706
2	3	3/362	M2039	31:42	31:42	5:07	11.726384
3	4	4/362	M2039	32:28	32:27	5:14	11.464968
4	5	5/362	M2039	32:52	32:52	5:18	11.320755

Датафрейм `results` содержит по одной строке на каждого из 1633 финишировавших бегунов. Для количественной оценки результатов мы будем использовать столбец `MPH`, в котором указана средняя скорость каждого бегуна в милях в час. Выберем этот столбец и воспользуемся атрибутом `values` для извлечения значений скорости в виде NumPy-массива:

```
speeds = results["MPH"].values
```

Я финишировал с чистым временем (net time) 42:44, поэтому мою строку можно найти так:

```
my_result = results.query("Nettime == '42:44'")
my_result
```

	Place	Div/Tot	Division	Guntime	Nettime	Min/Mile	MPH
96	97	26/256	M4049	42:48	42:44	6:53	8.716707

Индекс моей строки равен 96, так что мою скорость можно получить так:

```
my_speed = speeds[96]
```

Для подсчета количества бегунов, прошедших дистанцию с моей скоростью или медленнее, можно применить метод `sum`:

```
(speeds <= my_speed).sum()
```

```
1537
```

Также для вычисления процента бегунов, прошедших дистанцию с моей скоростью или медленнее, можно использовать метод `mean`:

```
(speeds <= my_speed).mean() * 100
```

```
94.12124923453766
```

Результат, составивший около 94 %, — это мой процентильный ранг в забеге.

Если обобщить, то следующая функция вычисляет процентильный ранг определенного значения в некоторой последовательности значений:

```
def percentile_rank(x, seq):
    """Процентильный ранг x.

    x: значение
    seq: последовательность значений

    returns: процентильный ранг 0-100
    """
    return (seq <= x).mean() * 100
```

В столбце `Division` датафрейма `results` указана категория каждого участника, определяемая по полу и возрасту. Например, моя категория была `M4049`, составленная из бегунов-мужчин в возрасте от 40 до 49 лет. При помощи метода `query` выберем строки бегунов моей категории и извлечем из них значения скорости:

```
my_division = results.query("Division == 'M4049'")
my_division_speeds = my_division["MPH"].values
```

Теперь можно использовать функцию `percentile_rank` для вычисления моего процентильного ранга в моей категории участников:

```
percentile_rank(my_speed, my_division_speeds)
```

```
90.234375
```

Если нужно проделать обратную операцию, то следующая функция по процентильному рангу находит соответствующее значение в последовательности:

```
def percentile(p, seq):
    n = len(seq)
    i = (1 - p / 100) * (n + 1)
    return seq[round(i)]
```

Здесь `n` — количество элементов в последовательности, а `i` — индекс элемента с заданным процентильным рангом. Когда мы указываем процентильный ранг, то соответствующее значение в последовательности называется **процентилем**:

```
percentile(90, my_division_speeds)
```

```
8.591885441527447
```

В моей категории 90-й процентиль составлял около 8.6 мили в час.

Прошло несколько лет с момента моего участия в том забеге, и я нахожусь в категории `M5059`. Поэтому посмотрим, с какой скоростью я должен был бы бежать,

чтобы иметь тот же процентильный ранг в моей новой категории. Мы можем ответить на этот вопрос, переведя мой процентильный ранг в категории `M4049`, равный примерно 90.2 %, в значение скорости в категории `M5059`:

```
next_division = results.query("Division == 'M5059'")
next_division_speeds = next_division["MPH"].values

percentile(90.2, next_division_speeds)
```

```
8.017817371937639
```

В категории `M5059` участник с таким же процентильным рангом, как у меня, бежал чуть быстрее 8 миль в час. Чтобы найти его, воспользуемся методом `query`:

```
next_division.query("MPH > 8.01").tail(1)
```

	Place	Div/Tot	Division	Guntime	Nettime	Min/Mile	MPH
222	223	18/171	M5059	46:30	46:25	7:29	8.017817

Этот бегун финишировал со временем 46:25 и занял 18-е место из 171 участника своей категории.

После знакомства с понятиями процентильного ранга и процентиля мы готовы к работе с интегральной функцией распределения.

Интегральная функция распределения

Интегральная функция распределения, ИФР (cumulative distribution function, CDF), — это еще один способ, описывающий распределение набора значений, помимо таблицы частот и функции вероятности. При заданном значении `x` ИФР показывает долю значений, меньших или равных `x`. Для начала посмотрим на пример с короткой последовательностью:

```
t = [1, 2, 2, 3, 5]
```

Один из способов вычислить ИФР — начать с функции вероятности. Вот объект `Pmf`, представляющий собой распределение значений в `t`:

```
from empiricaldist import Pmf

pmf = Pmf.from_seq(t)
pmf
```

	вероятность
1	0.2
2	0.4
3	0.2
5	0.2

Как вы узнали из предыдущей главы, для поиска в объекте `Pmf` какого-либо значения можно использовать оператор квадратных скобок:

```
pmf[2]
```

```
0.4
```

Результатом является доля значений исходной последовательности, равных указанному значению. В данном примере два из пяти значений равны `2`, поэтому результат равен 0.4. Эту пропорцию также можно рассматривать как вероятность того, что выбранное из последовательности случайным образом значение равно `2`.

У `Pmf` есть метод `make_cdf`, который вычисляет накопленную сумму вероятностей:

```
cdf = pmf.make_cdf()
cdf
```

	вероятность
1	0.2
2	0.6
3	0.8
5	1.0

Результатом является объект `Cdf`, производный от `Series` библиотеки Pandas. Поэтому для поиска значения можно использовать оператор квадратных скобок:

```
cdf[2]
```

```
0.6000000000000001
```

Результатом является доля значений в последовательности, меньших или равных заданному. В нашем примере три из пяти значений в последовательности меньше или равны `2`, поэтому результат равен 0.6. Эту пропорцию также можно рассматривать как вероятность того, что выбранное из последовательности случайным образом значение будет меньше или равно `2`.

Объект `Cdf` можно вызывать как функцию, используя круглые скобки:

```
cdf(3)
```

```
array(0.8)
```

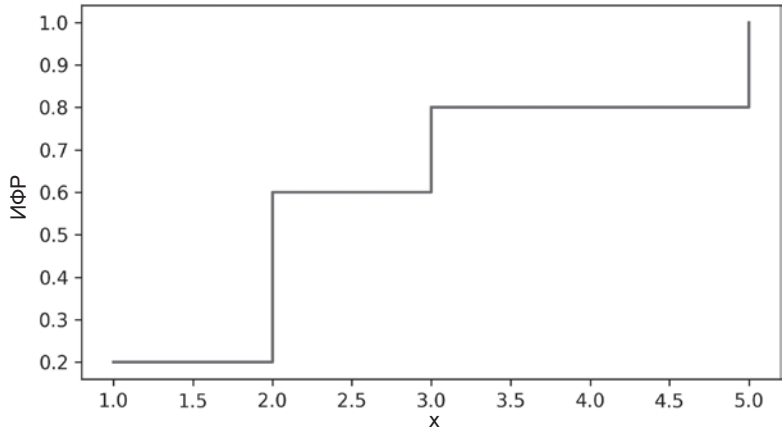
Интегральная функция распределения определена для всех чисел, а не только для тех, которые присутствуют в последовательности:

```
cdf(4)
```

```
array(0.8)
```

Чтобы визуализировать объект `Cdf`, можно использовать метод `step`, который отображает `Cdf` в виде ступенчатой функции:

```
cdf.step()
decorate(xlabel="x", ylabel="ИФР")
```



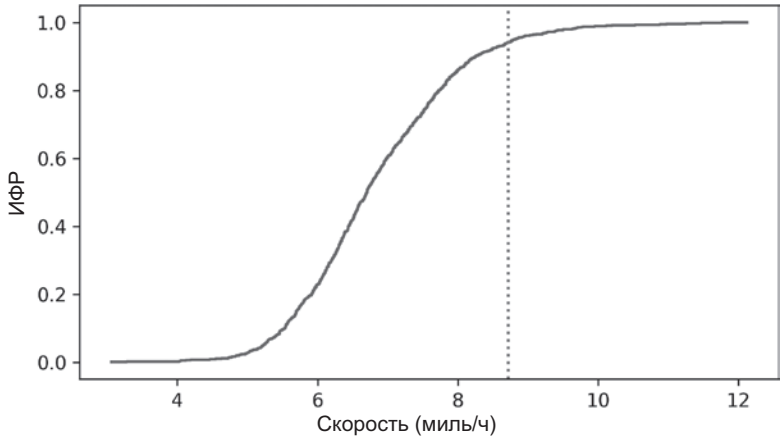
В качестве второго примера создадим объект `Cdf`, который будет представлять распределение скоростей бега из предыдущего раздела. Класс `Cdf` предоставляет функцию `from_seq`, которую можно использовать, чтобы создать объект `Cdf` на основе последовательности:

```
from empiricaldist import Cdf

cdf_speeds = Cdf.from_seq(speeds)
```

А вот как выглядит распределение (вертикальная линия обозначает мою скорость):

```
cdf_speeds.step()
plt.axvline(my_speed, ls=":", color="gray")
decorate(xlabel="Скорость (миль/ч)", ylabel="ИФР")
```



Если задать в качестве аргумента мою скорость, то на выходе будет доля участников, пробежавших с моей скоростью или медленнее. Если дополнительно умножить результат на 100, получится мой процентильный ранг:

```
cdf_speeds(my_speed) * 100
```

```
94.12124923453766
```

Таким образом, `Cdf` можно считать объектом, который на основе заданного значения вычисляет процентильный ранг, только возвращая долю значений в диапазоне от 0 до 1, а не процент от 0 до 100.

В классе `Cdf` имеется также метод `inverse`, обратный интегральной функции распределения: принимая долю от 0 до 1, он находит соответствующее значение.

Например, если кто-то утверждает, что он пробежал так же быстро или быстрее, чем 50 % участников, можно определить его скорость следующим образом:

```
cdf_speeds.inverse(0.5)
```

```
array(6.70391061)
```

Если у вас есть доля значений и вы используете обратный метод для нахождения соответствующего значения, в результате вы получите то, что называется **квантилем**. Поэтому функцию, обратную ИФР, иногда называют квантильной функцией.

Однако если у вас есть квантиль и вы используете ИФР для нахождения соответствующей доли значений, то результат не будет иметь отдельного названия.

По аналогии с процентилем и процентильным рангом его можно было бы назвать «квантильным рангом», но, насколько мне известно, никто так не говорит. Чаще всего эту величину называют просто интегральной вероятностью.

Сравнение ИФР

ИФР особенно эффективны для сравнения распределений. В качестве примера сравним распределение веса при рождении для первенцев и всех последующих детей. Мы снова загрузим датасет NSFG и создадим на его базе три объекта `DataFrame`: все дети, родившиеся живыми (live), первые (firsts) и последующие (others) дети:

```
from nsfg import get_nsfg_groups

live, firsts, others = get_nsfg_groups()
```

Из датафреймов `firsts` и `others` выберем полный вес при рождении в фунтах и воспользуемся методом `dropna` для удаления `nan`-значений:

```
first_weights = firsts["totalwgt_lb"].dropna()
first_weights.mean()
```

```
7.201094430437772
```

```
other_weights = others["totalwgt_lb"].dropna()
other_weights.mean()
```

```
7.325855614973262
```

Кажется, что первенцы в среднем весят немного меньше. Но подобное отличие может возникать по разным причинам. Например, небольшое количество первых детей могут весить необычно мало, или небольшое количество последующих детей — очень много. В этих ситуациях распределения будут иметь разную форму. В другом случае распределения могут совпадать по форме, но по-разному располагаться на оси значений.

Чтобы сравнить распределения, можно построить графики функции вероятности:

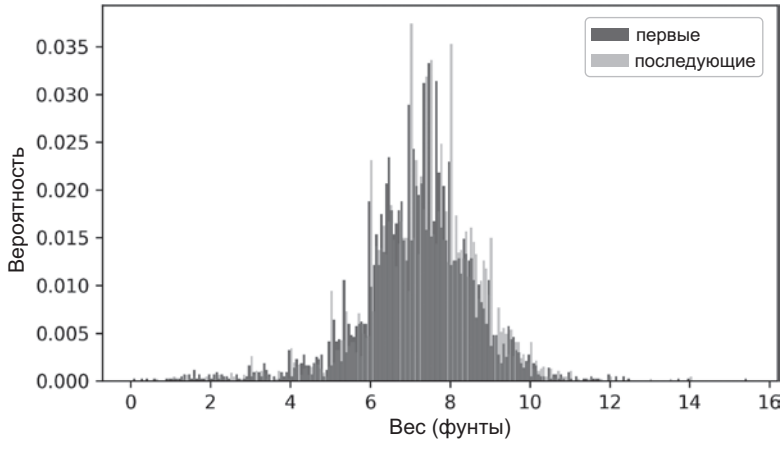
```
from empiricaldist import Pmf

first_pmf = Pmf.from_seq(first_weights, name="первые")
other_pmf = Pmf.from_seq(other_weights, name="последующие")
```

Но, как видим, здесь они не очень наглядны:

```
from thinkstats import two_bar_plots

two_bar_plots(first_pmf, other_pmf, width=0.06)
decorate(xlabel="Вес (фунты)", ylabel="Вероятность")
```



Я подобрал ширину и прозрачность столбцов так, чтобы распределения выглядели как можно четче, но сравнивать их все равно трудно. Видно множество пиков и впадин, а также некоторые очевидные различия, но трудно понять, какие из этих характеристик имеют значение. Помимо этого, трудно увидеть общие закономерности: например, сложно оценить визуально, какое из распределений имеет более высокое среднее значение.

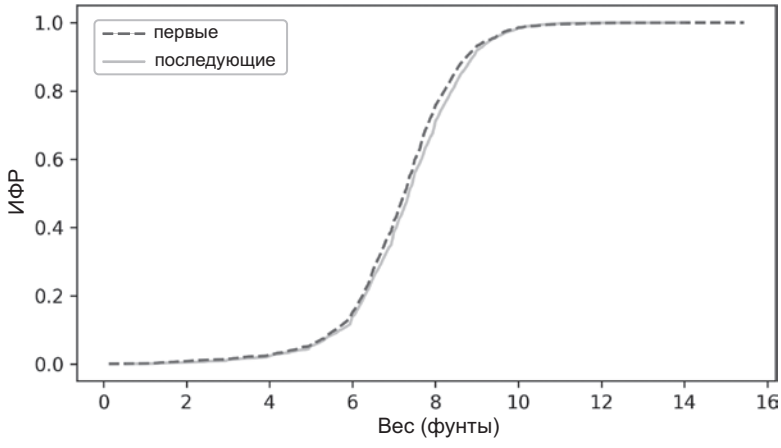
Эти проблемы можно частично решить, сгруппировав данные, то есть разделив диапазон величин на непересекающиеся интервалы и подсчитав количество величин в каждом интервале. Подобная группировка (binning) может иметь смысл, но сложно правильно подобрать ширину интервала. При достаточно широком интервале может сглаживаться не только шум, но и полезная информация.

Хорошей альтернативой такому подходу станет график ИФР:

```
first_cdf = first_pmf.make_cdf()
other_cdf = other_pmf.make_cdf()
```

Вот как он выглядит:

```
first_cdf.plot(ls="--")
other_cdf.plot(alpha=0.5)
decorate(xlabel="Вес (фунты)", ylabel="ИФР")
```



На этом рисунке форма распределений и различия между ними выглядят заметно яснее. Кривая для первых детей находится неизменно слева от кривой для последующих детей, указывая на то, что первенцы весят немного меньше во всем диапазоне весов, причем различие более заметно в районе среднего значения.

Процентильная статистика

В главе 3 мы вычисляли среднее арифметическое, устанавливающее центральную точку в распределении, а также стандартное отклонение, которое количественно измеряет степень разброса распределения. В одном из предыдущих упражнений мы также вычисляли асимметрию, которая указывает, является распределение скошенным влево или вправо. Одним из недостатков всех этих показателей является то, что они чувствительны к выбросам. Одно экстремальное значение в датасете может серьезно повлиять на среднее значение, стандартное отклонение или асимметрию.

Альтернатива — использование статистических показателей, основанных на процентилях распределения, которые, как правило, более **робастны**. Это означает, что они менее чувствительны к выбросам. Чтобы убедиться в этом, снова загрузим данные NSFG и не будем проводить их очистку:

```
from nsfg import read_stata

dct_file = "2002FemPreg.dct"
dat_file = "2002FemPreg.dat.gz"

preg = read_stata(dct_file, dat_file)
```

Напомним, что вес при рождении записывается в двух отдельных колонках — для фунтов и для унций:

```
birthwgt_lb = preg["birthwgt_lb"]
birthwgt_oz = preg["birthwgt_oz"]
```

Если мы создадим объект `Hist`[1] со значениями из `birthwgt_oz`, то увидим, что они содержат специальные значения 97, 98 и 99, указывающие на недостающие данные:

```
from empiricaldist import Hist

Hist.from_seq(birthwgt_oz).tail(5)
```

birthwgt_oz	freqs
14.0	475
15.0	378
97.0	1
98.0	1
99.0	46

Столбец `birthwgt_lb` содержит те же специальные значения, но в нем также присутствует значение 51, которое, по всей видимости, является ошибкой[2]:

```
Hist.from_seq(birthwgt_lb).tail(5)
```

birthwgt_lb	freqs
15.0	1
51.0	1
97.0	1
98.0	1
99.0	57

Теперь представьте два сценария. В первом мы очищаем эти переменные, заменяя пропущенные и недопустимые значения на `nan`, а затем вычисляем общий вес в фунтах. Делением объекта `birthwgt_oz_clean` на 16 мы переводим вес из унций в фунты в десятичной системе счисления:

[1] Hist — алиас для FreqTab. В примечаниях к коду на GitHub автор указывает, что в предыдущих версиях FreqTab назывался Hist, но ревьюеры порекомендовали заменить имя; однако Hist он решил оставить для обратной совместимости. — *Примеч. науч. ред.*

[2] 51 фунт примерно равен 23 кг. — *Примеч. пер.*

```
birthwgt_lb_clean = birthwgt_lb.replace([51, 97, 98, 99], np.nan)
birthwgt_oz_clean = birthwgt_oz.replace([97, 98, 99], np.nan)

total_weight_clean = birthwgt_lb_clean + birthwgt_oz_clean / 16
```

Во втором сценарии мы пренебрегаем очисткой данных и непреднамеренно вычисляем общий вес с учетом перечисленных фиктивных (bogus) значений:

```
total_weight_bogus = birthwgt_lb + birthwgt_oz / 16
```

Такой «фиктивный» датасет содержит только 49 некорректных значений, что составляет около 0.5 % от общего объема данных:

```
count1, count2 = total_weight_bogus.count(), total_weight_clean.count()
diff = count1 - count2

diff, diff / count2 * 100
```

```
(49, 0.5421553441026776)
```

Теперь вычислим среднее значение в обоих сценариях:

```
mean1, mean2 = total_weight_bogus.mean(), total_weight_clean.mean()
mean1, mean2
```

```
(7.319680587652691, 7.265628457623368)
```

Фиктивные значения оказывают умеренное влияние на среднее. Если мы примем среднюю величину очищенных данных за верную, то среднее фиктивных данных будет отличаться менее чем на 1 %:

```
(mean1 - mean2) / mean2 * 100
```

```
0.74394294099376
```

Подобная ошибка может остаться незамеченной, но посмотрим, что происходит со стандартными отклонениями:

```
std1, std2 = total_weight_bogus.std(), total_weight_clean.std()
std1, std2
```

```
(2.096001779161835, 1.4082934455690173)
```

```
(std1 - std2) / std2 * 100
```

```
48.83274403900607
```

Стандартное отклонение фиктивных данных отличается почти на 50 %, что намного заметнее. Наконец, вот асимметрия двух датасетов:

```
def skewness(seq):
    deviations = seq - seq.mean()
    return np.mean(deviations**3) / seq.std(ddof=0) ** 3
```

```
skew1, skew2 = skewness(total_weight_bogus), skewness(total_weight_clean)
skew1, skew2
```

```
(22.251846195422484, -0.5895062687577697)
```

Асимметрия фиктивного датасета искажена почти в 40 раз и имеет неверный знак! При добавлении выбросов к данным распределение сильно скашивается вправо, на что указывает большая положительная асимметрия. Но распределение достоверных данных слегка скошено влево, о чем свидетельствует малая отрицательная асимметрия.

Эти результаты показывают, что небольшое количество выбросов оказывает умеренное влияние на среднее значение, сильное — на стандартное отклонение и катастрофическое — на асимметрию.

Другой способ — использование показателей, основанных на процентилях. К ним относятся:

- Медиана, представляющая собой 50-й процентиль, подобно среднему значению указывает на центр распределения.
- Межквартильный размах, равный разнице между 25-м и 75-м процентилями, аналогично стандартному отклонению измеряет разброс распределения.
- Квартильная асимметрия использует квартили распределения (25-й, 50-й и 75-й процентили) для количественной оценки асимметрии.

Объект `Cdf` удобен для вычисления таких процентильных показателей. В демонстрационных целях создадим объекты `Cdf` из фиктивного (bogus) и очищенного (clean) датасетов:

```
cdf_total_weight_bogus = Cdf.from_seq(total_weight_bogus)
cdf_total_weight_clean = Cdf.from_seq(total_weight_clean)
```

Следующая функция принимает на вход объект `Cdf` и использует его метод `inverse` для вычисления 50-го процентиля, который является медианой (во всяком случае, это один из способов определения медианы датасета):

```
def median(cdf):
    m = cdf.inverse(0.5)
    return m
```

Теперь вычислим медиану обоих датасетов:

```
median(cdf_total_weight_bogus), median(cdf_total_weight_clean)
```

```
(array(7.375), array(7.375))
```

Результаты одинаковы, так что в данном случае выбросы вообще не повлияли на медиану. И в целом выбросы оказывают меньшее влияние на медиану, чем на среднее значение.

Межквартильный размах, МКР (interquartile range, IQR), — это разница между 75-м и 25-м процентилями. Следующая функция принимает объект `Cdf` и возвращает значение МКР:

```
def iqr(cdf):
    low, high = cdf.inverse([0.25, 0.75])
    return high - low
```

А вот значения межквартильного размаха для двух датасетов:

```
iqr(cdf_total_weight_bogus), iqr(cdf_total_weight_clean)
```

```
(1.625, 1.625)
```

Как правило, выбросы оказывают меньшее влияние на МКР, чем на стандартное отклонение (в данном случае вообще не оказывают).

Наконец, вот функция, которая вычисляет квартильную асимметрию, которая зависит от следующих трех статистических показателей:

- медианы (median);
- средней точки (midpoint) между 25-м и 75-м процентилями;
- половины МКР (semi-IQR).

```
def quartile_skewness(cdf):
    low, median, high = cdf.inverse([0.25, 0.5, 0.75])
    midpoint = (high + low) / 2
    semi_iqr = (high - low) / 2
    return (midpoint - median) / semi_iqr
```

Вот чему равна квартильная асимметрия для двух датасетов:

```
qskew1 = quartile_skewness(cdf_total_weight_bogus)
qskew2 = quartile_skewness(cdf_total_weight_clean)
qskew1, qskew2
```

```
(-0.07692307692307693, -0.07692307692307693)
```

Небольшое количество выбросов в этом примере не оказывает эффекта на квартильную асимметрию. В целом из этих примеров видно, что статистические показатели, основанные на процентилях, менее чувствительны к выбросам и ошибкам в данных.

Случайные числа

Объекты `Cdf` обеспечивают эффективный способ генерации случайных чисел из распределений. Сначала мы генерируем случайные числа из равномерного распределения от 0 до 1. Затем в полученных точках вычисляем обратную ИФР. Следующая функция реализует этот алгоритм:

```
def sample_from_cdf(cdf, n):
    ps = np.random.random(size=n)
    return cdf.inverse(ps)
```

Для примера сгенерируем случайную выборку (sample) скоростей бега:

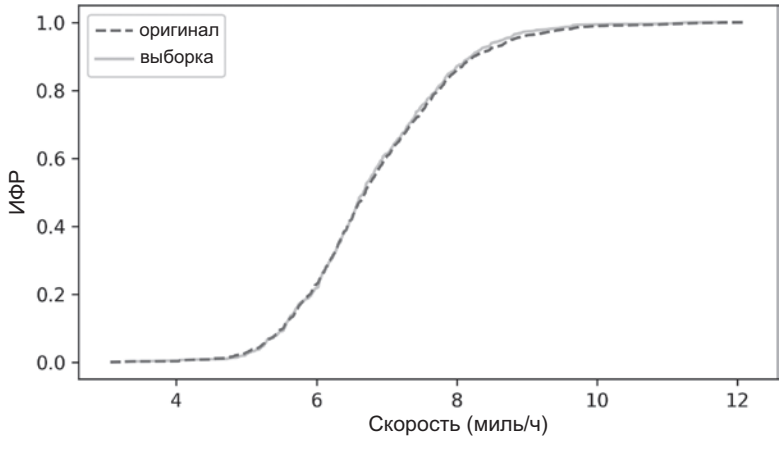
```
sample = sample_from_cdf(cdf_speeds, 1001)
```

Чтобы убедиться, что все верно, можно сравнить ИФР полученной выборки и исходного датасета:

```
cdf_sample = Cdf.from_seq(sample)

cdf_speeds.plot(label="оригинал", ls="--")
cdf_sample.plot(label="выборка", alpha=0.5)

decorate(xlabel="Скорость (миль/ч)", ylabel="ИФР")
```



Распределение, построенное на основе выборки, соответствует распределению исходных данных. Чтобы понять, как работает этот алгоритм, обсудим следующий вопрос. Допустим, мы генерируем случайную выборку из совокупности скоростей бега и находим процентильный ранг скоростей в выборке. Теперь предположим, что мы вычисляем ИФР процентильных рангов. Как вы думаете, как она будет выглядеть?

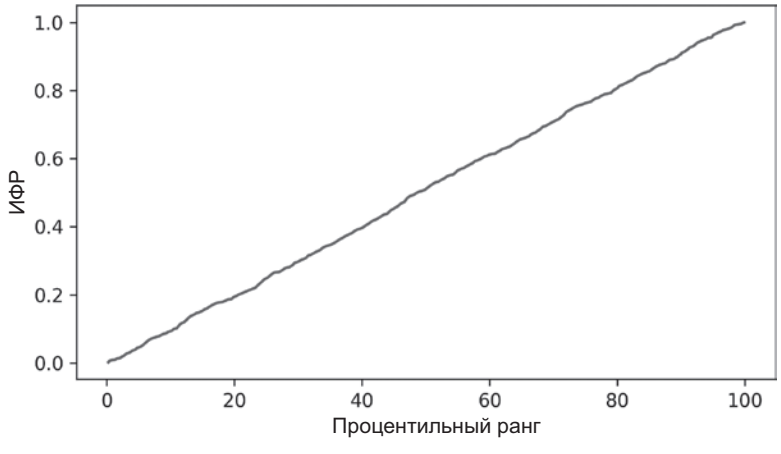
Давайте узнаем. Вот процентильные ранги для сформированной нами выборки:

```
percentile_ranks = cdf_speeds(sample) * 100
```

А вот ИФР процентильных рангов:

```
cdf_percentile_rank = Cdf.from_seq(percentile_ranks)
cdf_percentile_rank.plot()

decorate(xlabel="Процентильный ранг", ylabel="ИФР")
```



ИФР процентильных рангов близка к прямой линии, проведенной между 0 и 1. И это логично: ведь в любом распределении доля значений с процентильным рангом менее 50 % равна 0.5, с рангом менее 90 % — 0.9 и т. д.

В классе `Cdf` имеется метод `sample`, использующий этот алгоритм. Поэтому выборку можно было сгенерировать так:

```
sample = cdf_speeds.sample(1001)
```

Глоссарий

Процентильный ранг (percentile rank)

Процент значений в распределении, которые меньше или равны заданной величине.

Процентиль (percentile)

Значение в распределении, связанное с заданным процентильным рангом.

Интегральная функция распределения, ИФР (cumulative distribution function, CDF)
Функция, которая сопоставляет некоторое значение с долей значений распределения, меньших или равных этому значению.

Квантиль (quantile)
Значение в распределении, которое больше или равно указанной доли значений.

Робастный (robust)
Статистический показатель является робастным, если он устойчив к влиянию экстремальных значений или выбросов.

Межквартильный разброс, МКР (interquartile range, IQR)
Разница между 75-м и 25-м процентилями, используемая для оценки разброса распределения.

Упражнения

Упражнение 4.1

Сколько вы весили при рождении? Если вы не знаете, позвоните своей маме или кому-то еще, кто знает. А если такого человека нет, то для этого упражнения вы можете использовать мой вес при рождении — 8.5 фунта.

Возьмите данные NSFG (все дети, рожденные живыми), рассчитайте распределение веса при рождении и используйте эту информацию для определения своего процентильного ранга. Если вы были первым ребенком, найдите свой процентильный ранг в распределении для первенцев. В противном случае возьмите распределение для вторых и последующих детей. Если вы находитесь в 90-м процентиле или выше, перезвоните своей маме и извинитесь перед ней:

```
from nsfg import get_nsfg_groups

live, firsts, others = get_nsfg_groups()
```

Упражнение 4.2

В датасете NSFG столбец `babysex` указывает пол рожденных живыми детей — мужской или женский. Воспользуемся методом `query` для выбора строк для младенцев мужского (male) и женского (female) пола:

```
male = live.query("babysex == 1")
female = live.query("babysex == 2")
len(male), len(female)
```

```
(4641, 4500)
```

Создайте объекты `Cdf`, представляющие распределение веса при рождении для младенцев мужского и женского пола. Постройте график этих двух ИФР. Каковы различия в форме и положении распределений?

Если ребенок мужского пола весит 8.5 фунта, каков его процентильный ранг? Каков вес ребенка женского пола, имеющего такой же процентильный ранг?

Упражнение 4.3

В данных о беременности NSFG выберите столбец `agepreg` и создайте объект `Cdf`, представляющий распределение возраста на момент зачатия для каждой беременности. Используйте эту ИФР для вычисления процентной доли женщин в возрасте до 20 лет включительно, а также процентной доли женщин в возрасте до 30 лет включительно. Используйте эти результаты для вычисления процентной доли возраста от 20 до 30:

```
from nsfg import read_fem_preg

preg = read_fem_preg()
```

Упражнение 4.4

Вот данные о скорости бега людей, которые финишировали в забеге James Joyce Ramble, упомянутом ранее в этой главе:

```
speeds = results["MPH"].values
```

Создайте объект `Cdf`, представляющий распределение этих скоростей, и используйте его для вычисления медианы, МКР и квартильной асимметрии. Является распределение скошенным влево или вправо?

Упражнение 4.5

Предполагается, что числа, сгенерированные при помощи функции `np.random.random`, равномерно распределены в диапазоне от 0 до 1, это означает, что ИФР выборки должна быть прямой линией. Посмотрим, так ли это. Вот выборка из 1001 числа. Постройте для нее график ИФР. Представляет ли он прямую линию?

```
t = np.random.random(1001)
```

ГЛАВА 5

Моделирование распределений

Распределения, с которыми мы работали до сих пор, называются эмпирическими, поскольку они основаны на эмпирических наблюдениях, или, другими словами, на данных. Многие датасеты, с которыми мы встречаемся на практике, можно точно аппроксимировать при помощи теоретических распределений, обычно задаваемых простой математической формулой. В этой главе вы познакомитесь с некоторыми из таких теоретических распределений и датасетами, которые с их помощью можно моделировать.

Вы на примерах убедитесь в следующем:

- Количество попаданий и промахов в соревнованиях по стендовой стрельбе хорошо моделируется с помощью биномиального распределения.
- В таких спортивных играх, как хоккей и футбол, количество голов за игру описывается распределением Пуассона, а время между голами — экспоненциальным распределением.
- Вес новорожденных соответствует нормальному распределению, также называемому гауссовым, а вес взрослого человека описывается логнормальным распределением.

На случай если вы не знакомы с этими распределениями или с каким-то из видов спорта, я дам все необходимые разъяснения. Каждый из примеров мы начинаем с моделирования простой схемы и далее проверяем, что результаты моделирования соответствуют теоретическому распределению, а затем — насколько точно с моделью согласуются реальные данные.

Биномиальное распределение

В качестве первого примера рассмотрим такой вид спорта, как стендовая стрельба, в котором участники используют гладкоствольные ружья для стрельбы по глиняным тарелочкам, подбрасываемым в воздух. На международных соревнованиях, в том числе Олимпийских играх, проводится пять раундов по 25 мишеней в каждом. При необходимости для определения победителя проводятся дополнительные раунды.

В основу модели таких соревнований положим допущение, что каждый участник с одинаковой вероятностью, `p`, поражает каждую из мишеней. Конечно, это будет упрощением: на самом деле у одних участников она выше, чем у других, и даже у одного конкретного участника вероятность может меняться от попытки

к попытке. Но даже при таких допущениях эта модель, как вы увидите, делает удивительно точные предсказания.

Для моделирования воспользуемся следующей функцией, которая принимает в качестве аргументов количество целей (`n`) и вероятность попадания в каждую из них (`p`), а возвращает последовательность из единиц и нулей, обозначающих попадания и промахи соответственно:

```
def flip(n, p):
    choices = [1, 0]
    probs = [p, 1 - p]
    return np.random.choice(choices, n, p=probs)
```

Вот пример моделирования раунда из 25 мишеней, в котором вероятность попадания в каждую из них равна 90 %:

```
flip(25, 0.9)
```

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 1])
```

Если сгенерировать более длинную последовательность и построить объект `Pmf` для полученных результатов, то можно убедиться, что доли единиц и нулей в распределении верны, по крайней мере приблизительно:

```
from empiricaldist import Pmf

seq = flip(1000, 0.9)
pmf = Pmf.from_seq(seq)
pmf
```

	вероятность
0	0.101
1	0.899

Теперь для моделирования (simulate) раунда стендовой стрельбы реализуем новую функцию, вызывающую функцию `flip` и возвращающую количество попаданий:

```
def simulate_round(n, p):
    seq = flip(n, p)
    return seq.sum()
```

Предположим, что в крупном соревновании 200 участников стреляют по 5 раундов каждый с одинаковой вероятностью попадания в цель, `p=0.9`. Можем смоделировать подобное соревнование, вызвав `simulate_round` 1000 раз:

```
n = 25
p = 0.9
results_sim = [simulate_round(n, p) for i in range(1000)]
```

Средний балл близок к `22.5` — произведению `n` на `p`:

```
np.mean(results_sim), n * p
```

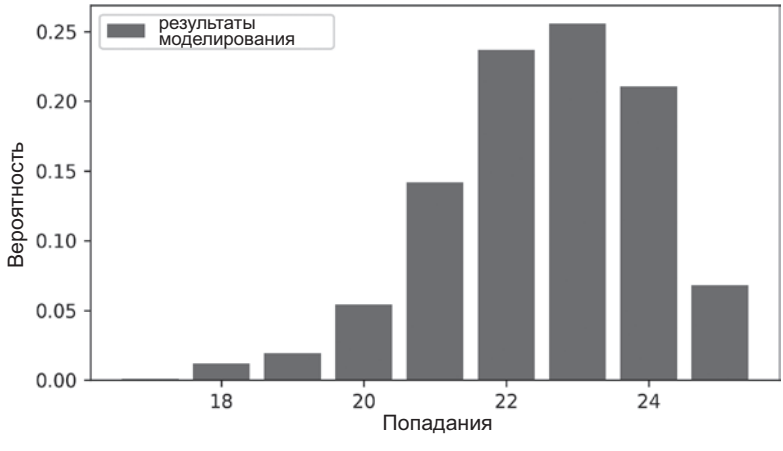
```
(22.522, 22.5)
```

Вот как выглядит распределение результатов:

```
from empiricaldist import Pmf

pmf_sim = Pmf.from_seq(results_sim, name="результаты моделирования")

pmf_sim.bar()
decorate(xlabel="Попадания", ylabel="Вероятность")
```



Максимум распределения близок к среднему значению, а само распределение скошено влево.

Вместо того чтобы проводить моделирование, мы могли бы предсказать это распределение. Математически распределение этих исходов соответствует **биномиальному распределению** с функцией вероятности, которую легко рассчитать:

```
from scipy.special import comb

def binomial_pmf(k, n, p):
    return comb(n, k) * (p**k) * ((1 - p) ** (n - k))
```

В библиотеке SciPy реализована функция `comb`, которая вычисляет количество сочетаний (combinations) из `n` элементов, взятых по `k` за раз, что часто произносится как «из n по k».

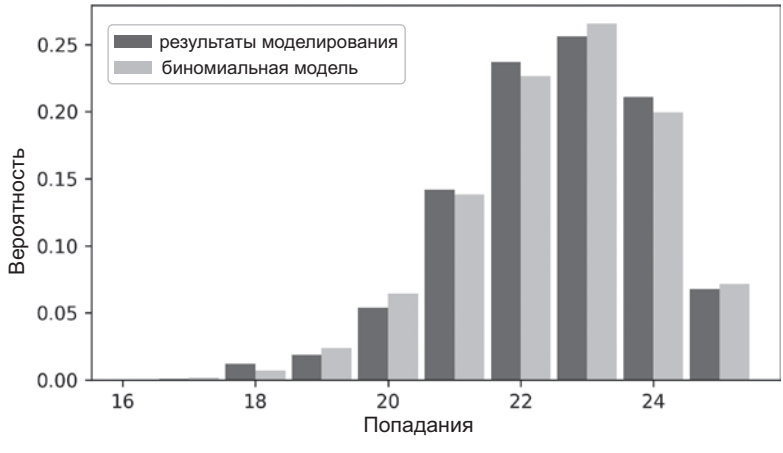
Функция `binomial_pmf` вычисляет вероятность при заданном `p`, попасть `k` раз, сделав `n` попыток. Если вызвать эту функцию, подав на вход массив из `k` значений, то можно создать объект `Pmf`, представляющий собой распределение вероятностей исходов:

```
ks = np.arange(16, n + 1)
ps = binomial_pmf(ks, n, p)
pmf_binom = Pmf(ps, ks, name="биномиальная модель")
```

И вот как оно выглядит в сравнении с результатами моделирования:

```
from thinkstats import two_bar_plots

two_bar_plots(pmf_sim, pmf_binom)
decorate(xlabel="Попадания", ylabel="Вероятность")
```



Распределения сходны, а небольшие различия вызваны случайными вариациями в результатах моделирования. Это соответствие не должно вызывать удивления, поскольку и численное моделирование, и биномиальная модель опираются на одни и те же предположения, в частности на то, что каждая попытка имеет одинаковую вероятность успеха. Более строгой проверкой модели будет ее сопоставление с реальными данными.

На странице википедии, посвященной соревнованиям по стендовой стрельбе среди мужчин на летних Олимпийских играх 2020 года, можно найти таблицу с результатами квалификационных раундов. Инструкции по загрузке данных приведены в Jupyter-блокноте для этой главы.

```
filename = "Shooting_at_the_2020_Summer_Olympics_Mens_skeet"
```

```
tables = pd.read_html(filename)
table = tables[6]
table.head()
```

	Rank (Место)	Athlete (Спортсмен)	Country (Страна)	1	2	3	4	5	Total[3] (Итого)	Shoot-off (Перестрелка)	Notes (Примечания)
0	1	Éric Delaunay	France	25	25	25	24	25	124	+6	Q, OR
1	2	Tammaro Cassandro	Italy	24	25	25	25	25	124	+5	Q, OR
2	3	Eetu Kallioinen	Finland	25	25	24	25	24	123	NaN	Q
3	4	Vincent Hancock	United States	25	25	25	25	22	122	+8	Q
4	5	Abdullah Al-Rashidi	Kuwait	25	25	24	25	23	122	+7	Q

В таблице содержится по одной строке для каждого участника и по одному столбцу для каждого из пяти раундов. Выберем столбцы с результатами раундов и с помощью NumPy-функции `flatten` преобразуем выбранные данные в одномерный массив:

```
columns = ["1", "2", "3", "4", "5"]
results = table[columns].values.flatten()
```

Для 30 участников мы получили результаты 150 раундов, по 25 выстрелов в каждом — всего 3574 попаданий (hits) при 3750 попытках (shots):

```
total_shots = 25 * len(results)
total_hits = results.sum()
n, total_shots, total_hits
```

```
(25, 3750, 3575)
```

Таким образом, доля успешных попыток составляет 95,3 %:

```
p = total_hits / total_shots
p
```

```
0.9533333333333334
```

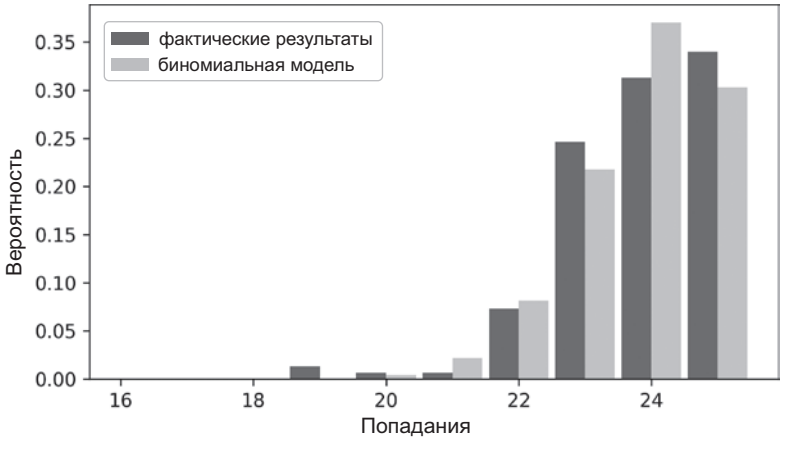
Теперь создадим объект `Pmf` для биномиального распределения с `n = 25` и только что вычисленным значением `p`:

```
ps = binomial_pmf(ks, n, p)
pmf_binom = Pmf(ps, ks, name="биномиальная модель")
```

Полученное распределение можно сравнить с Pmf фактических результатов:

```
pmf_results = Pmf.from_seq(results, name="фактические результаты")

two_bar_plots(pmf_results, pmf_binom)
decorate(xlabel="Попадания", ylabel="Вероятность")
```



В данном примере биномиальное распределение является хорошей аппроксимацией данных, несмотря на нереалистичное предположение, что возможности всех спортсменов одинаковы и постоянны.

Распределение Пуассона

Другим примером спортивных результатов, распределение которых легко предсказать, служит количество шайб, забитых в хоккейных матчах.

Начнем с моделирования игры продолжительностью 60 минут (3600 секунд), исходя из предположения, что команды забивают в среднем по 6 шайб за игру и что вероятность заброшенной шайбы в секунду, p, постоянна во времени:

```
n = 3600
m = 6
p = m / 3600
p
```

```
0.0016666666666666668
```

Тогда для моделирования n секунд игры и подсчета количества заброшенных за это время шайб можно использовать такую функцию:

```
def simulate_goals(n, p):
    return flip(n, p).sum()
```

Если провести моделирование большого набора игр, то результат подтвердит, что среднее количество шайб за игру близко к 6:

```
goals = [simulate_goals(n, p) for i in range(1001)]
np.mean(goals)
6.021978021978022
```

Для описания этих результатов можно было бы использовать биномиальное распределение, но, когда `n` велико, а `p` мало, результаты также хорошо аппроксимируются **распределением Пуассона**. Оно определяется параметром, обычно обозначаемым греческой буквой λ («лямбда»). В коде ниже этому параметру соответствует переменная `lam` (`lambda` нельзя использовать в качестве имени переменной в Python, поскольку это зарезервированное слово). `lam` в данном примере представляет собой показатель результативности, составляющий 6 голов за игру.

ФВ распределения Пуассона легко вычислить. Имея значение `lam`, для вычисления вероятности того, что в игре будет забито `k` голов, можно использовать следующую функцию:

```
from scipy.special import factorial

def poisson_pmf(k, lam):
    return (lam**k) * np.exp(-lam) / factorial(k)
```

В библиотеке SciPy есть функция `factorial`, которая вычисляет произведение целых чисел от `1` до `k`.

Если вызвать `poisson_pmf`, указав массив `k` значений, то можно построить объект `Pmf`, представляющий распределение исходов:

```
lam = 6
ks = np.arange(20)
ps = poisson_pmf(ks, lam)
pmf_poisson = Pmf(ps, ks, name="пуассоновская модель")
```

А также легко подтвердить, что среднее значение распределения близко к 6:

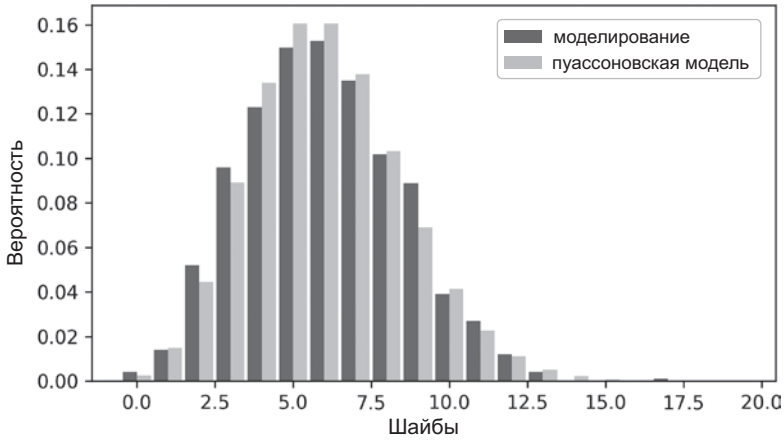
```
pmf_poisson.normalize()
pmf_poisson.mean()
```

```
5.999925498375129
```

Теперь сравним результаты моделирования с распределением Пуассона с тем же средним:

```
pmf_sim = Pmf.from_seq(goals, name="моделирование")

two_bar_plots(pmf_sim, pmf_poisson)
decorate(xlabel="Шайбы", ylabel="Вероятность")
```



Распределения похожи, если не считать небольших различий из-за случайной изменчивости. И этоне удивительно, ведь в основе случайного моделирования и распределения Пуассона лежит одно и то же предположение, что шайба с равной вероятностью забивается за любую секунду игры. Поэтому более строгим тестом будет проверка того, насколько точно модель соответствует реальным данным.

Сначала я скачал результаты всех игр регулярного чемпионата Национальной хоккейной лиги (НХЛ) 2023–2024 годов (без игр плей-офф) с сайта HockeyReference. Далее я извлек информацию о шайбах, забитых за 60 минут основного времени матча, без учета дополнительного времени и серии буллитов. Результаты записаны в файле формата HDF, в котором один ключ соответствует одной игре. По ключу можно получить список моментов времени, в секундах с начала игры, в которые были заброшены шайбы. Инструкции по загрузке данных приведены в Jupyter-блокноте для этой главы.

Вот как считывать ключи из файла:

```
filename = "nhl_2023_2024.hdf"

with pd.HDFStore(filename, "r") as store:
    keys = store.keys()

len(keys), keys[0]
```

```
(1312, '/202310100PIT')
```

В регулярном сезоне было проведено 1312 игр. Каждый ключ содержит дату игры и аббревиатуру из трех букв, обозначающую принимающую команду. Функция `read_hdf` по ключу выдает список моментов времени для забитых шайб:

```
times = pd.read_hdf(filename, key=keys[0])
times
```

```
0     424
1    1916
2    2137
3    3005
4    3329
5    3513
dtype: int64
```

В первой игре сезона было забито шесть шайб: первая — после 424 секунд игры, последняя — после 3513 секунд, всего за 87 секунд до конца игры.

Следующий цикл считывает результаты всех игр, подсчитывает количество шайб в каждой из них и сохраняет результаты в виде списка:

```
goals = []

for key in keys:
    times = pd.read_hdf(filename, key=key)
    n = len(times)
    goals.append(n)
```

Среднее количество шайб за игру немного превышает 6:

```
lam = np.mean(goals)
lam
```

```
6.0182926829268295
```

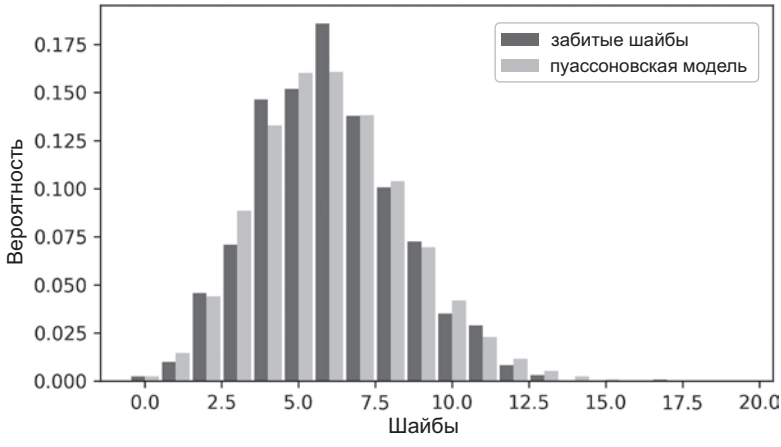
Воспользуемся функцией `poisson_pmf` для создания объекта `Pmf`, представляющего распределение Пуассона с тем же средним значением, что и данные:

```
ps = poisson_pmf(ks, lam)
pmf_poisson = Pmf(ps, ks, name="пуассоновская модель")
```

И вот как это выглядит в сравнении с ФВ данных:

```
pmf_goals = Pmf.from_seq(goals, name="забитые шайбы")

two_bar_plots(pmf_goals, pmf_poisson)
decorate(xlabel="Шайбы", ylabel="Вероятность")
```



Распределение Пуассона хорошо согласуется с данными, что означает, что эта модель точно описывает процесс забивания шайб в хоккее.

Экспоненциальное распределение

В предыдущем разделе мы использовали простую модель хоккейного матча, в которой шайба с одинаковой вероятностью может быть забита в любую секунду игры. Оказывается, что в рамках этой же модели время до первой шайбы описывается **экспоненциальным распределением**.

Чтобы доказать это, снова предположим, что команды забивают в среднем шесть шайб за игру, и вычислим вероятность забивания шайбы в секунду:

```
n = 3600
m = 6
p = m / 3600
p
```

```
0.0016666666666666668
```

Следующая функция моделирует `n` секунд матча и с помощью метода `argmax` определяет время первой шайбы:

```
def simulate_first_goal(n, p):
    return flip(n, p).argmax()
```

Мы получаем верный результат, поскольку функция `flip` возвращает последовательность из единиц и нулей, а значит, максимальное значение почти всегда равно 1. Если в последовательности имеется одна единица (забитая шайба) или больше, то `argmax` возвращает индекс первой из них. В противном случае возвращается 0, но такое бывает редко, так что мы проигнорируем этот вариант.

При помощи функции `simulate_first_goal` смоделируем 1001 игру и составим список моментов времени первых забитых шайб:

```
first_goal_times = [simulate_first_goal(n, p) for i in range(1001)]
mean = np.mean(first_goal_times)
mean
```

```
597.7902097902098
```

Среднее время до первой шайбы составляет около 600 секунд, или 10 минут. И это логично: если мы ожидаем 6 шайб за 60 минут игры, то в среднем можно ожидать одну забитую шайбу каждые 10 минут.

Когда `n` велико, а `p` мало, можно математически доказать, что время ожидания первого гола следует экспоненциальному распределению.

Поскольку при моделировании генерируется множество уникальных значений времени, для сравнения распределений мы будем использовать ИФР, а не ФВ. Кроме того, ИФР экспоненциального распределения легко вычислить:

```
def exponential_cdf(x, lam):
    return 1 - np.exp(-lam * x)
```

Параметр `lam` — это среднее количество событий в единицу времени, в данном примере — забитых шайб в секунду. Мы можем использовать среднее значение результатов моделирования для вычисления `lam`:

```
lam = 1 / mean
lam
```

```
0.0016728276636563566
```

Если подать на вход этой функции массив значений времени, можно получить аппроксимацию распределения времени первой забитой шайбы. NumPy-функция `linspace` создает массив равноотстоящих значений; в данном примере она вычисляет 201 значение от 0 до 3600, включая границы диапазона:

```
from empiricaldist import Cdf

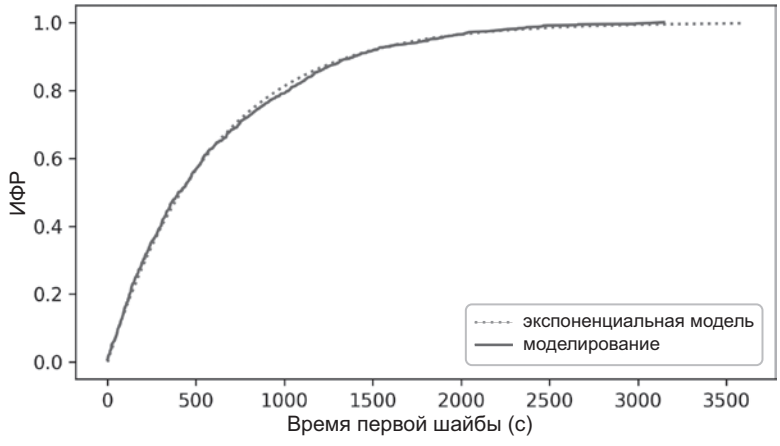
ts = np.linspace(0, 3600, 201)
ps = exponential_cdf(ts, lam)
cdf_expo = Cdf(ps, ts, name="экспоненциальная модель")
```

Теперь можно сравнить результаты моделирования с только что вычисленным экспоненциальным распределением:

```
cdf_sim = Cdf.from_seq(first_goal_times, name="моделирование")

cdf_expo.plot(ls=":", color="gray")
cdf_sim.plot()
```

```
decorate(xlabel="Время первой шайбы (с)", ylabel="ИФР")
```



Экспоненциальная модель прекрасно согласуется с результатами моделирования, но более надежным показателем будет то, как она соотносится с реальными данными.

В следующем цикле для каждой игры считываются результаты, извлекается время первой забитой шайбы и сохраняется в списке. Если шайб в игре забито не было, в список добавляется `nan`:

```
firsts = []

for key in keys:
    times = pd.read_hdf(filename, key=key)
    if len(times) > 0:
        firsts.append(times[0])
    else:
        firsts.append(np.nan)
```

Чтобы оценить голевой темп, можно воспользоваться методом `nanmean`, который вычисляет среднее значение времени, игнорируя значения `nan`:

```
lam = 1 / np.nanmean(firsts)
lam
```

```
0.0015121567467720825
```

Теперь можно построить ИФР экспоненциального распределения с тем же голевым темпом, что и в данных:

```
ps = exponential_cdf(ts, lam)
cdf_expo = Cdf(ps, ts, name="экспоненциальная модель")
```

Чтобы посчитать ИФР данных, применим аргумент `dropna=False`, что приводит к добавлению `nan`-значений на конце временного диапазона:

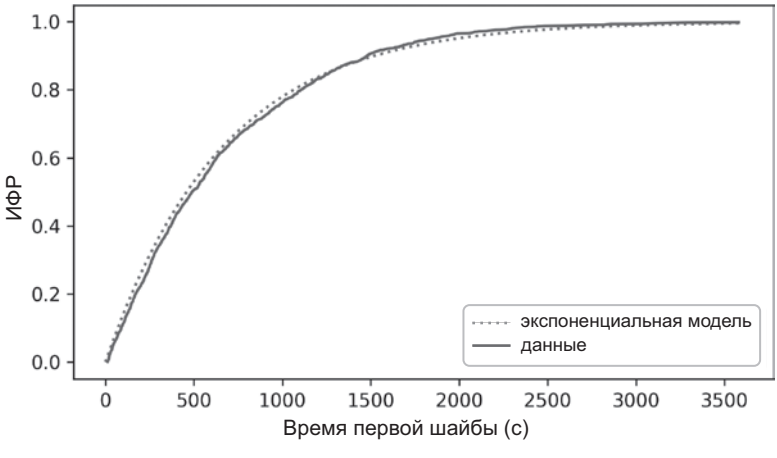
```
cdf_firsts = Cdf.from_seq(firsts, name="данные", dropna=False)
cdf_firsts.tail()
```

	вероятность
3286.0	0.996951
3581.0	0.997713
NaN	1.000000

На следующем рисунке можно сравнить экспоненциальное распределение с распределением реальных данных:

```
cdf_expo.plot(ls=":", color="gray")
cdf_firsts.plot()

decorate(xlabel="Время первой шайбы (с)", ylabel="ИФР")
```



Данные на отдельных отрезках расходятся с моделью: по графику видно, что за первые 1000 секунд забито меньше шайб, чем предсказывает модель. Но тем не менее, модель хорошо соответствует данным.

Базовое предположение обеих моделей — пуассоновской модели голов и экспоненциальной модели времени — заключается в том, что гол одинаково вероятен в течение любой секунды игры. Если вы поинтересуетесь у хоккейного болельщика, так ли это, он ответит отрицательно и будет прав — объективная реальность во многом противоречит подобным допущениям. Тем не менее теоретические распределения часто в высшей степени хорошо согласуются с реальными данными.

Нормальное распределение

Многие показатели, которые мы измеряем в реальной жизни, подчиняются **нормальному распределению**, также известному как гауссово распределение или колоколообразная кривая. Чтобы понять, чем объясняется это распределение, рассмотрим модель роста гигантской тыквы. Допустим, что каждый день тыква набирает один фунт, если погода плохая; два фунта, если погода удовлетворительная; и три фунта, если погода хорошая. Также предположим, что погода бывает плохой, удовлетворительной или хорошей с равной вероятностью.

Для имитации такой модели в течение `n` дней можно задействовать следующую функцию, возвращающую итоговое значение прироста (gains) веса:

```
def simulate_growth(n):
    choices = [1, 2, 3]
    gains = np.random.choice(choices, n)
    return gains.sum()
```

В модуле `random` библиотеки NumPy имеется функция `choice`, которая генерирует массив `n` результатов случайного выбора из последовательности значений — в данном примере из списка `choices`.

Теперь предположим, что 1001 человек, живущих в разных местах с разной погодой, выращивают гигантские тыквы. Если мы смоделируем процесс роста в течение 100 дней, то получим список из 1001 значения веса:

```
sim_weights = [simulate_growth(100) for i in range(1001)]
m, s = np.mean(sim_weights), np.std(sim_weights)
m, s
```

```
(199.37062937062936, 8.388630840376777)
```

Среднее значение близко к 200 фунтам, а стандартное отклонение составляет около 8 фунтов. Чтобы увидеть, соответствуют ли веса нормальному распределению, воспользуемся следующей функцией, которая принимает на вход выборку и создает объект `Cdf`, представляющий нормальное распределение с теми же значениями среднего и стандартного отклонения, что и выборка. Распределение рассчитывается в диапазоне от четырех стандартных отклонений ниже среднего (low) до четырех стандартных отклонений выше среднего (high):

```
from scipy.stats import norm

def make_normal_model(data):
    m, s = np.mean(data), np.std(data)
    low, high = m - 4 * s, m + 4 * s
    qs = np.linspace(low, high, 201)
    ps = norm.cdf(qs, m, s)
    return Cdf(ps, qs, name="нормальная модель")
```

Вот пример ее использования:

```
cdf_model = make_normal_model(sim_weights)
```

Теперь создадим объект `Cdf`, содержащий распределение результатов моделирования:

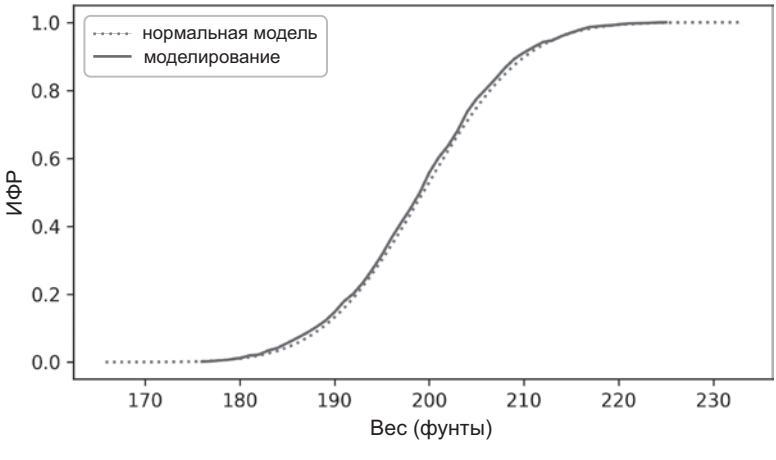
```
cdf_sim_weights = Cdf.from_seq(sim_weights, name="моделирование")
```

Напишем функцию для сравнения распределений. Аргументы `cdf_model` и `cdf_data` являются объектами `Cdf`. `xlabel` — это строка, а `options` — это словарь параметров отображения `cdf_data`:

```
def two_cdf_plots(cdf_model, cdf_data, xlabel="", **options):
    cdf_model.plot(ls=":", color="gray")
    cdf_data.plot(**options)
    decorate(xlabel=xlabel, ylabel="ИФР")
```

А вот результат ее работы:

```
two_cdf_plots(cdf_model, cdf_sim_weights, xlabel="Вес (фунты)")
```



Нормальная модель очень хорошо согласуется с распределением веса, полученным при моделировании. Вообще, когда сочетается влияние достаточно большого количества случайных факторов, результат стремится к нормальному распределению. Это следствие центральной предельной теоремы, к которой мы вернемся позже в главе 14.

Но сначала посмотрим, насколько хорошо нормальное распределение описывает реальные данные. Для примера рассмотрим распределение веса новорожденных по данным Национального исследования роста семьи (NSFG). Воспользуемся функцией `read_fem_preg` для чтения данных, а затем выберем столбец `totalwgt_lb`, в котором записан вес при рождении в фунтах:

```
import nsfg

preg = nsfg.read_fem_preg()
birth_weights = preg["totalwgt_lb"].dropna()
```

Средний вес новорожденного составляет около 7.27 фунта, а стандартное отклонение — 1.4 фунта, но, как вы уже знаете, в этом датасете есть выбросы, которые, скорее всего, являются ошибочными значениями:

```
m, s = np.mean(birth_weights), np.std(birth_weights)
m, s
```

```
(7.265628457623368, 1.40821553384062)
```

Чтобы уменьшить влияние выбросов на оценку среднего и стандартного отклонения, при помощи функции `trimboth` библиотеки SciPy удалим самые большие и самые маленькие значения:

```
from scipy.stats import trimboth

trimmed = trimboth(birth_weights, 0.01)
m, s = np.mean(trimmed), np.std(trimmed)
m, s
```

```
(7.280883100022579, 1.2430657948614345)
```

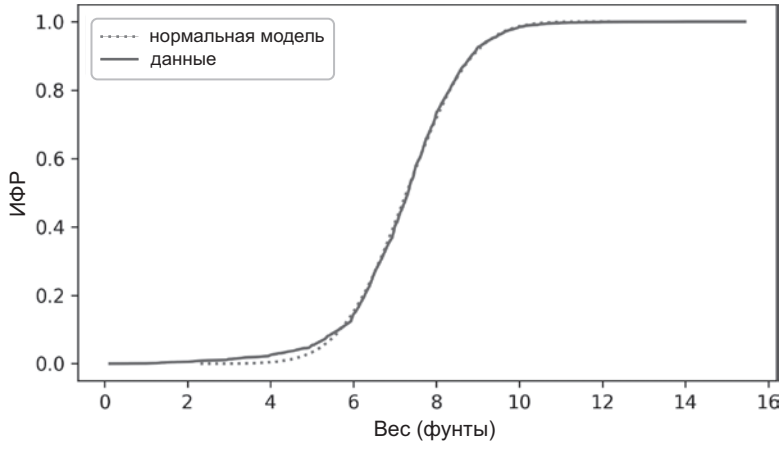
Среднее значение таких усеченных (trimmed) данных немного меньше, а стандартное отклонение существенно меньше. Воспользуемся этими данными для построения нормальной модели:

```
cdf_model = make_normal_model(trimmed)
```

И сравним ее с `Cdf` реальных данных:

```
cdf_birth_weight = Cdf.from_seq(birth_weights, name='данные')

two_cdf_plots(cdf_model, cdf_birth_weight, xlabel="Вес при рождении (фунты)")
```



Нормальная модель хорошо соответствует данным, за исключением диапазона ниже пяти фунтов, где распределение данных расположено левее модельной кривой. Это означает, что дети с самым маленьким весом в реальности легче, чем предсказывает нормальное распределение. Действительность обычно оказывается сложнее простых математических моделей.

Логнормальное распределение

В предыдущем разделе мы моделировали процесс роста тыквы, исходя из предположения, что, в зависимости от погоды, тыква прибавляет от одного до трех фунтов в день. Теперь предположим, что прирост веса тыквы пропорционален ее текущему весу, так что большие тыквы за день набирают больше, чем маленькие. Такая модель выглядит более правдоподобной.

Следующая функция моделирует процесс такого пропорционального роста, при котором тыква прибавляет в весе 3 % в плохую погоду, 5 % — в удовлетворительную и 7 % — в хорошую. Снова будем исходить из предположения, что погода в любой день с равной вероятностью будет плохой, удовлетворительной или хорошей:

```
def simulate_proportionate_growth(n):
    choices = [1.03, 1.05, 1.07]
    gains = np.random.choice(choices, n)
    return gains.prod()
```

Если тыква прибавляет в весе 3 %, то конечный вес будет равен произведению исходного веса на коэффициент 1.03. Таким образом, чтобы рассчитать вес через 100 дней, нужно выбрать случайные коэффициенты и перемножить их.

Чтобы смоделировать 1001 тыкву, вызовем эту функцию 1001 раз, сохраняя итоговый вес:

```
sim_weights = [simulate_proportionate_growth(100) for i in range(1001)]
np.mean(sim_weights), np.std(sim_weights)
```

```
(130.80183363824722, 20.956047434921466)
```

Средний вес составляет около 131 фунта, а стандартное отклонение — около 21 фунта. Таким образом, согласно этой модели, вес тыквы в среднем меньше, но более изменчив, чем в предыдущей модели.

Можно доказать математически, что вес в данном случае соответствует **логарифмически нормальному (логнормальному) распределению**. Это означает, что логарифмы веса соответствуют нормальному распределению. Для проверки вычислим логарифм веса, его среднее значение и стандартное отклонение. Можно было бы применить логарифм по любому основанию, но лучше использовать основание 10, поскольку в этом случае будет проще интерпретировать результаты:

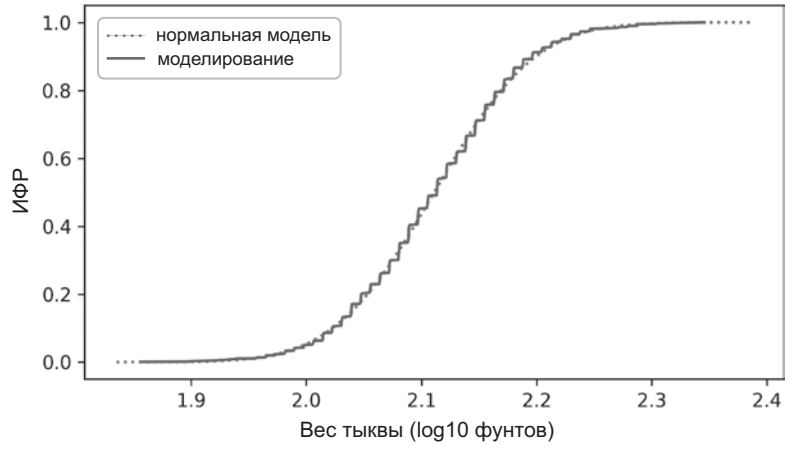
```
log_sim_weights = np.log10(sim_weights)
m, s = np.mean(log_sim_weights), np.std(log_sim_weights)
m, s
```

```
(2.1111299372609933, 0.06898607064749827)
```

Теперь сравним распределение логарифма веса с нормальным распределением с теми же средним значением и стандартным отклонением:

```
cdf_model = make_normal_model(log_sim_weights)
cdf_log_sim_weights = Cdf.from_seq(log_sim_weights, name="моделирование")

two_cdf_plots(
    cdf_model, cdf_log_sim_weights, xlabel="Вес тыквы (log10 фунтов)"
)
```



Модель, как и ожидалось, прекрасно согласуется с результатами моделирования.

Если у человека, так же как у тыквы, изменение веса от года к году пропорционально его текущему весу, то следует ожидать, что распределение веса взрослых людей будет соответствовать логнормальному распределению. Проверим это.

Национальный центр профилактики хронических заболеваний и укрепления здоровья (The National Center for Chronic Disease Prevention and Health Promotion) США проводит ежегодное исследование в рамках Системы наблюдения за поведенческими факторами риска (Behavioral Risk Factor Surveillance System, BRFSS). В 2008 году было опрошено 414 509 респондентов, которым были заданы вопросы об их демографических характеристиках, состоянии здоровья и факторах риска. Среди собранных ими данных — данные о весе в килограммах 398 484 респондентов. Инструкции по загрузке этих данных приведены в Jupyter-блокноте к этой главе.

В модуле `thinkstats` реализована функция, считывающая данные BRFSS и возвращающая Pandas-объект `DataFrame`:

```
from thinkstats import read_brfss

brfss = read_brfss()
```

Вес взрослых в килограммах записан в столбце `wtkg2`:

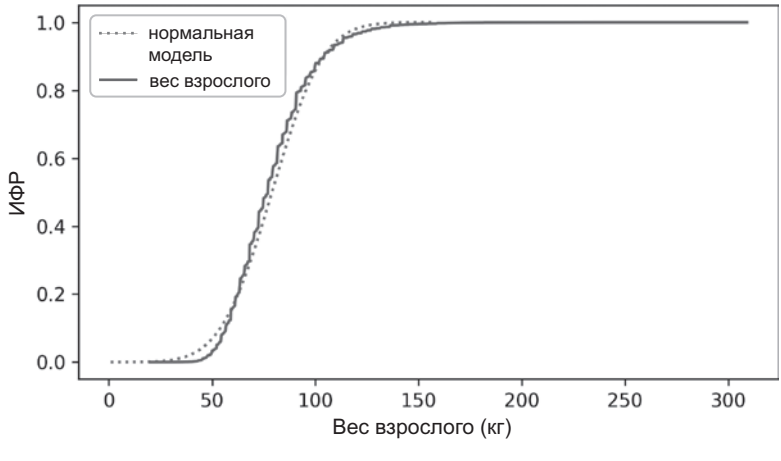
```
adult_weights = brfss["wtkg2"].dropna()
m, s = np.mean(adult_weights), np.std(adult_weights)
m, s
```

```
(78.9924529968581, 19.546132387397257)
```

Средний вес составляет около 79 кг. Прежде чем вычислять логарифм, посмотрим, насколько значение веса соответствует нормальному распределению:

```
cdf_model = make_normal_model(adult_weights)
cdf_adult_weights = Cdf.from_seq(adult_weights, name="вес взрослого")

two_cdf_plots(cdf_model, cdf_adult_weights, xlabel="Вес взрослого (кг)")
```



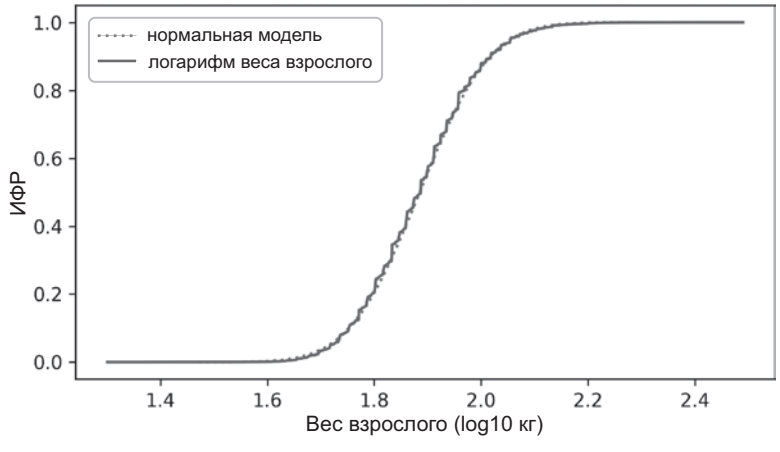
При определенных обстоятельствах нормальное распределение можно считать приемлемым для таких данных, но посмотрим, можно ли улучшить полученные результаты.

Вот распределение логарифма веса и нормальная модель с теми же средним значением и стандартным отклонением:

```
log_adult_weights = np.log10(adult_weights)
cdf_model = make_normal_model(log_adult_weights)

cdf_log_adult_weights = Cdf.from_seq(log_adult_weights, name="логарифм веса
взрослого")
```

```
two_cdf_plots(cdf_model, cdf_log_adult_weights, xlabel="Вес взрослого (log10 кг)")
```



Нормальная модель лучше согласуется с логарифмом веса, чем с самим весом. Это свидетельствует о том, что пропорциональный рост лучше подходит для моделирования набора веса, чем аддитивный.

Зачем нужны модели

В начале этой главы я упомянул, что многие явления окружающей действительности можно смоделировать при помощи теоретических распределений. Но, возможно, вас интересует вопрос, почему это так важно.

Как и все модели, теоретические распределения являются абстракциями. Это означает, что они не учитывают детали, которые считаются несущественными. Например, наблюдаемое распределение может содержать ошибки измерений или другие странности, специфичные для данной выборки; теоретические модели игнорируют подобное.

Теоретические модели также представляют собой своего рода сжатие данных. Когда модель хорошо аппроксимирует какой-либо датасет, несколько чисел могут эффективно справиться с обобщением большого объема данных.

Порой вызывает удивление, что данные о природном явлении соответствуют теоретическому распределению, но такие результаты наблюдений могут дать представление о сути физических систем. Иногда то, почему наблюдаемое распределение имеет ту или иную форму, можно объяснить. Например, в предыдущем разделе вы узнали, что вес взрослого человека хорошо моделируется логнормальным распределением, что позволяет предположить, что изменения веса год к году пропорциональны текущему весу.

Кроме того, теоретические распределения, как вы увидите в главе 14, поддаются математическому анализу.

Но важно помнить, что все модели несовершенны. Данные объективной реальности никогда не соответствуют теоретическому распределению полностью. Люди иногда рассуждают так, как будто данные сгенерированы моделями; например, они могут заявлять, что распределение человеческого роста является нормальным или распределение доходов является логнормальным. Но в строгом понимании это не так: между физической действительностью и математическими моделями всегда существуют различия.

Модели полезны, если они отражают существенные детали и пренебрегают неважными. Но определение того, что считать важным или неважным, зависит от того, для чего вы планируете использовать модель.

Глоссарий

Биномиальное распределение (binomial distribution)

Теоретическое распределение, часто используемое для моделирования количества успехов, или попаданий, в последовательности попаданий и промахов.

Распределение Пуассона (Poisson distribution)

Теоретическое распределение, часто используемое для моделирования количества событий, происходящих за определенный промежуток времени.

Экспоненциальное распределение (exponential distribution)

Теоретическое распределение, часто используемое для моделирования времени между событиями.

Нормальное распределение (normal distribution)

Теоретическое распределение, часто используемое для моделирования данных, описываемых симметричной колоколообразной кривой.

Логнормальное распределение (lognormal distribution)

Теоретическое распределение, часто используемое для моделирования данных, описываемых скошенной вправо колоколообразной кривой.

Упражнения

Упражнение 5.1

В файле респондента датасета NSFG в столбце `numfmhh` указано «количество членов семьи» в домохозяйстве каждой респондентки. Вот как можно задействовать функцию `read_fem_resp` для чтения файла и метод `query` для выбора респонденток, которым на момент опроса было 25 лет и старше:

```
from nsfg import read_fem_resp

resp = read_fem_resp()
```

```
older = resp.query("age >= 25")
num_family = older["numfmhh"]
```

Создайте объект `Pmf` переменной `numfmhh` для таких респонденток старшего возраста и сравните эту функцию вероятности с распределением Пуассона, обладающим тем же средним значением. Насколько хорошо модель Пуассона соответствует данным?

Упражнение 5.2

Ранее в этой главе вы узнали, что время до первой заброшенной шайбы в хоккейном матче описывается экспоненциальным распределением. Если наша модель голов верна, то вероятность забросить шайбу одинакова в любое время, независимо от того, сколько времени прошло с момента предыдущего заброса шайбы. И если это так, то следует ожидать, что время между голами также будет распределено по экспоненте.

Цикл ниже снова считывает хоккейные данные, вычисляет время между следующими одна за другой заброшенными шайбами, если в игре их больше одной, и помещает время между шайбами в список:

```
intervals = []

for key in keys:
    times = pd.read_hdf(filename, key=key)
    if len(times) > 1:
        intervals.extend(times.diff().dropna())
```

Используйте функцию `exponential_cdf` для построения ИФР экспоненциального распределения с тем же средним значением, что и наблюдаемые интервалы, и сравните эту модель с ИФР данных.

Упражнение 5.3

Распределение человеческого роста больше похоже на нормальное или логнормальное? Чтобы получить ответ на этот вопрос, выберем из BRFSS данные о росте следующим образом:

```
adult_heights = brfss["htm3"].dropna()
m, s = np.mean(adult_heights), np.std(adult_heights)
m, s
```

```
(168.82518961012298, 10.35264015645592)
```

Постройте ИФР этих значений и сравните ее с нормальным распределением с теми же средним значением и стандартным отклонением. Затем вычислите логарифмы роста и постройте их распределение в сравнении с нормальным распределением. По результатам визуального сравнения, какая модель лучше согласуется с данными?

ГЛАВА 6

Функция плотности вероятности

В предыдущей главе мы моделировали данные с помощью теоретических распределений, в том числе биномиального, пуассоновского, экспоненциального и нормального.

Биномиальное распределение и распределение Пуассона являются **дискретными**, это означает, что их исходы являются отдельными единицами, такими как целое число попаданий или промахов, количество забитых голов. В дискретном распределении каждому исходу соответствует значение вероятности.

Экспоненциальное и нормальное распределения являются **непрерывными**, это означает, что исходы могут находиться в любой точке диапазона возможных значений. В непрерывном распределении каждому исходу соответствует **плотность вероятности**. Плотность вероятности — абстрактное понятие, которое поначалу может оказаться трудным для понимания, но мы будем вводить его постепенно, шаг за шагом. Для начала еще раз обратимся к сравнению распределений.

Сравнение распределений

В предыдущей главе, когда мы сравнивали дискретные распределения, то использовали столбчатые диаграммы для отображения их функций вероятности (ФВ). А когда сравнивали непрерывные распределения, то строили линейный график для отображения их интегральных функций распределения (ИФР).

Для дискретных распределений мы также могли бы использовать ИФР. Например, рассмотрим ФВ распределения Пуассона с `lam=2.2`, хорошо моделирующего распределение размера домохозяйств в данных NSFG.

Для чтения файла данных респондента можно использовать функцию `read_fem_resp`:

```
from nsfg import read_fem_resp

resp = read_fem_resp()
```

Теперь выберем значения размера домохозяйства (num_family) для респондентов в возрасте (age) 25 лет и старше (older):

```
older = resp.query("age >= 25")
num_family = older["numfmhh"]
```

Создадим объект `Pmf`, представляющий собой распределение ответов:

```
from empiricaldist import Pmf

pmf_family = Pmf.from_seq(num_family, name="данные")
```

Также создадим объект `Pmf` для распределения Пуассона с тем же средним значением:

```
from thinkstats import poisson_pmf

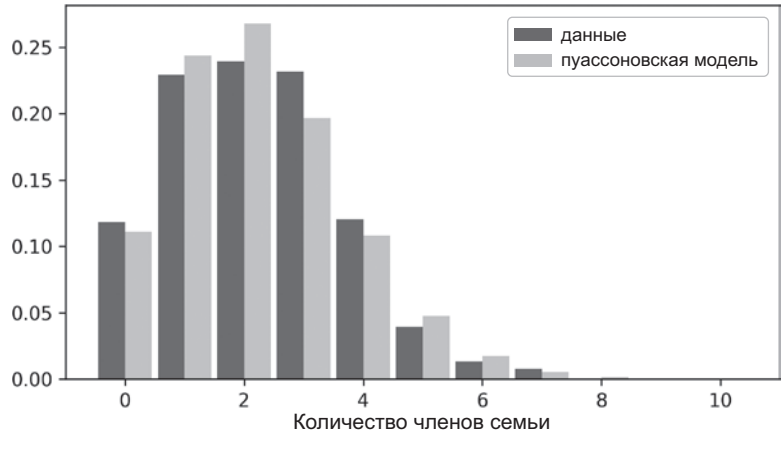
lam = 2.2
ks = np.arange(11)
ps = poisson_pmf(ks, lam)

pmf_poisson = Pmf(ps, ks, name="пуассоновская модель")
```

Посмотрим, как распределение данных соотносится с пуассоновской моделью:

```
from thinkstats import two_bar_plots

two_bar_plots(pmf_family, pmf_poisson)
decorate(xlabel="Количество членов семьи")
```



Сравнение двух функций вероятности показывает, что модель хорошо согласуется с данными, но есть некоторые расхождения.

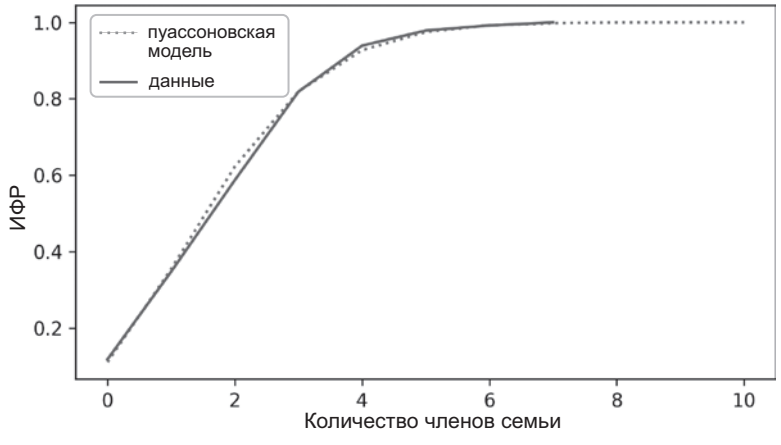
Чтобы понять, насколько существенны эти расхождения, полезно сравнить две ИФР. Для построения ИФР данных и модели можно воспользоваться методом `make_cdf`:

```
cdf_family = pmf_family.make_cdf()
cdf_poisson = pmf_poisson.make_cdf()
```

Вот как они выглядят:

```
from thinkstats import two_cdf_plots

two_cdf_plots(cdf_poisson, cdf_family)
decorate(xlabel="Количество членов семьи")
```



При сравнении двух ИФР расхождения менее заметны, но становится понятно, где и как они различаются. В то время как ФВ обычно подчеркивают небольшие различия, глядя на ИФР, можно понять общую картину.

ИФР также хорошо подходят для непрерывных данных. Для примера еще раз посмотрим на распределение веса новорожденных (birth_weights) из файла данных о беременности NSFG:

```
from nsfg import read_fem_preg

preg = read_fem_preg()
birth_weights = preg["totalwgt_lb"].dropna()
```

Вот код, который мы использовали в предыдущей главе, чтобы вписать нормальное распределение в данные:

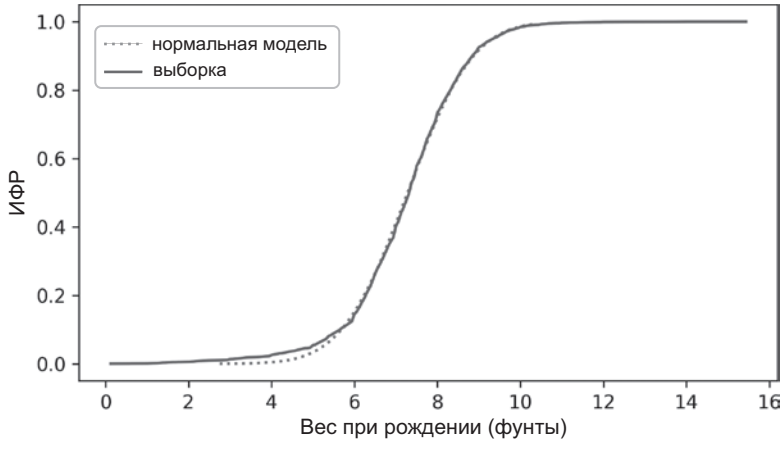
```
from scipy.stats import trimboth
from thinkstats import make_normal_model

trimmed = trimboth(birth_weights, 0.01)
cdf_model = make_normal_model(trimmed)
```

А вот распределение данных в сравнении с нормальной моделью:

```
from empiricaldist import Cdf

cdf_birth_weight = Cdf.from_seq(birth_weights, name="выборка")
two_cdf_plots(cdf_model, cdf_birth_weight, xlabel="Вес при рождении (фунты)")
```



Как вы уже видели в предыдущей главе, нормальная модель хорошо соответствует данным, за исключением диапазона самых маленьких весов.

На мой взгляд, ИФР обычно лучше всего подходят для сравнения данных с моделью. Но для тех, кто не знаком с ИФР, есть еще один вариант — функция плотности вероятности.

Плотность вероятности

Вначале рассмотрим **функцию плотности вероятности, ФПВ** (probability density function, PDF) нормального распределения, которая вычисляет плотность вероятности в точке `xs` при заданных параметрах `mu` и `sigma`:

```
def normal_pdf(xs, mu, sigma):
    z = (xs - mu) / sigma
    return np.exp(-(z**2) / 2) / sigma / np.sqrt(2 * np.pi)
```

В качестве `mu` и `sigma` используем среднее значение и стандартное отклонение усеченного датасета весов новорожденных:

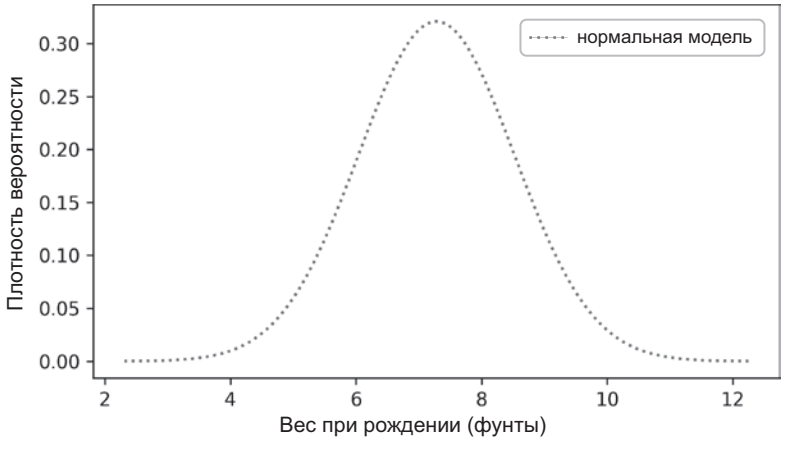
```
m, s = np.mean(trimmed), np.std(trimmed)
```

Теперь подадим массив значений веса (qs) на вход функции `normal_pdf`:

```
low = m - 4 * s
high = m + 4 * s
qs = np.linspace(low, high, 201)
ps = normal_pdf(qs, m, s)
```

Построим график полученного распределения:

```
plt.plot(qs, ps, label="нормальная модель", ls=":", color="gray")
decorate(xlabel="Вес при рождении (фунты)", ylabel="Плотность вероятности")
```



Результат выглядит как колоколообразная кривая, что характерно для нормального распределения.

Результатом вычисления значения функции `normal_pdf` является плотность вероятности. Например, вот величина функции плотности, вычисленная со средним значением в качестве аргумента, то есть там, где плотность наибольшая:

```
normal_pdf(m, m, s)
```

```
0.32093416297880123
```

Сама по себе плотность вероятности мало что значит, и самое главное — она *не* является вероятностью. Было бы неверно утверждать, что вероятность того, что случайно выбранный вес новорожденного равен `m`, составляет 32 %. На самом деле вероятность того, что вес ребенка при рождении действительно в точности равен `m` или любому другому конкретному значению, равна нулю.

Однако мы можем использовать плотность вероятности для вычисления вероятности того, что исход случайного события попадет в интервал между двумя значениями, вычислив площадь под кривой.

Это можно сделать с помощью функции `normal_pdf`, но удобнее использовать класс `NormalPdf`, который определен в модуле `thinkstats`. Вот как можно создать объект класса `NormalPdf` для распределения с теми же средним значением и стандартным отклонением, что и вес новорожденных в датасете NSFG:

```
from thinkstats import NormalPdf

pdf_model = NormalPdf(m, s, name="нормальная модель")
pdf_model
```

```
NormalPdf(7.280883100022579, 1.2430657948614345, name='нормальная модель')
```

Если вызвать этот объект как функцию, он вычислит значение ФПВ нормального распределения:

```
pdf_model(m)
```

```
0.32093416297880123
```

Теперь, чтобы вычислить площадь под графиком ФПВ, можно использовать следующую функцию, которая принимает на входе объект `NormalPdf` и границы интервала: нижнюю — `low` и верхнюю — `high`. Она вычисляет ФПВ в равномерно распределенных точках в интервале между `low` и `high` и использует функцию библиотеки SciPy `simpson` для оценки площади под кривой (название `simpson` объясняется использованием формулы Симпсона):

```
from scipy.integrate import simpson

def area_under(pdf, low, high):
    qs = np.linspace(low, high, 501)
    ps = pdf(qs)
    return simpson(y=ps, x=qs)
```

Если вычислить площадь под кривой во всем диапазоне оси абсцисс графика, то результат будет близок к 1:

```
area_under(pdf_model, 2, 12)
```

```
0.9999158086616793
```

Если мы расширим диапазон до всего множества вещественных чисел от минус бесконечности до плюс бесконечности, то общая площадь будет в точности равна 1.

Если в качестве нижней границы взять 0 (или любое значение, значительно меньше среднего), то можно вычислить долю новорожденных, вес которых меньше или равен 8.5 фунта:

```
area_under(pdf_model, 0, 8.5)
```

```
0.8366380335513807
```

Возможно, вы помните, что «доля значений, меньше или равных заданному» — это определение ИФР. Поэтому мы могли бы получить тот же результат, используя ИФР нормального распределения:

```
from scipy.stats import norm

norm.cdf(8.5, m, s)
```

```
0.8366380358092718
```

Точно так же можно использовать площадь под кривой плотности для вычисления доли новорожденных с весом от 6 до 8 фунтов:

```
area_under(pdf_model, 6, 8)
```

```
0.5671317752927691
```

Или получить тот же результат, используя ИФР для вычисления доли значений меньше 8 и затем вычитая из нее долю значений меньше 6:

```
norm.cdf(8, m, s) - norm.cdf(6, m, s)
```

```
0.5671317752921801
```

Таким образом, ИФР — это площадь под кривой ФПВ. Если вы знакомы с математическим анализом, то, по-другому говоря, ИФР — это интеграл ФПВ. И наоборот: ФПВ — это производная ИФР.

Экспоненциальная функция плотности вероятности

Для лучшего понимания концепции плотности вероятности полезно рассмотреть еще один пример. В предыдущей главе для моделирования времени первой заброшенной шайбы в хоккейном матче мы применяли экспоненциальное распределение. Для вычисления экспоненциальной ИФР мы использовали такую функцию, в которой `lam` — результативность в голах за единицу времени:

```
def exponential_cdf(x, lam):
    return 1 - np.exp(-lam * x)
```

Тогда ФПВ экспоненциального распределения можно рассчитать следующим образом:

```
def exponential_pdf(x, lam):
    return lam * np.exp(-lam * x)
```

В модуле `thinkstats` реализован объект `ExponentialPdf`, который задействует эту функцию для расчета экспоненциальной ФПВ. Мы можем использовать

этот объект для моделирования экспоненциального распределения с показателем результативности в шесть шайб за игру:

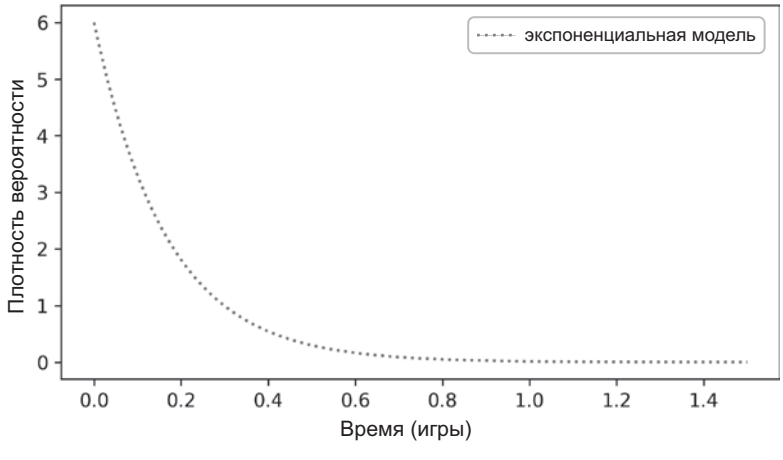
```
from thinkstats import ExponentialPdf

lam = 6
pdf_expo = ExponentialPdf(lam, name="экспоненциальная модель")
pdf_expo
```

```
ExponentialPdf(6, name='экспоненциальная модель')
```

У объекта `ExponentialPdf` есть метод `plot`, который можно использовать для построения графика ФПВ. Обратите внимание, что единицей измерения времени здесь являются игры, а не секунды, как в предыдущей главе:

```
qs = np.linspace(0, 1.5, 201)
pdf_expo.plot(qs, ls=":", color="gray")
decorate(xlabel="Время (игры)", ylabel="Плотность вероятности")
```



Взглянув на ось y, можно заметить, что плотность вероятности бывает больше 1, что лишний раз напоминает, что плотность вероятности — это не вероятность. В то время как площадь под кривой плотности — это вероятность, поэтому она никогда не должна превышать 1.

Если посчитать площадь под этой кривой в диапазоне от 0 до 1.5 игры, можно убедиться, что результат будет близок к 1:

```
area_under(pdf_expo, 0, 1.5)
```

```
0.999876590779019
```

Если существенно расширить диапазон, то результат будет немного больше 1, но лишь потому, что мы используем приближенные вычисления площади. Математически это в точности 1, что можно подтвердить при помощи экспоненциальной ИФР:

```
from thinkstats import exponential_cdf

exponential_cdf(7, lam)
```

```
1.0
```

Площадь под графиком плотности можно использовать для вычисления вероятности заброшенной шайбы в любой промежуток времени. Например, вот вероятность того, что шайба будет заброшена на первой минуте 60-минутной игры:

```
area_under(pdf_expo, 0, 1 / 60)
```

```
0.09516258196404043
```

К такому же результату можно прийти, используя экспоненциальную ИФР:

```
exponential_cdf(1 / 60, lam)
```

```
0.09516258196404048
```

Подведем итоги. При вычислении ФПВ результатом является плотность вероятности, не являющаяся вероятностью. Однако результатом вычисления площади под ФПВ-кривой будет вероятность того, что величина попадет в соответствующий интервал. Эту же вероятность можно найти, если вычислить ИФР в начале и в конце интервала, а затем посчитать их разницу.

Сравнение функции вероятности и функции плотности вероятности

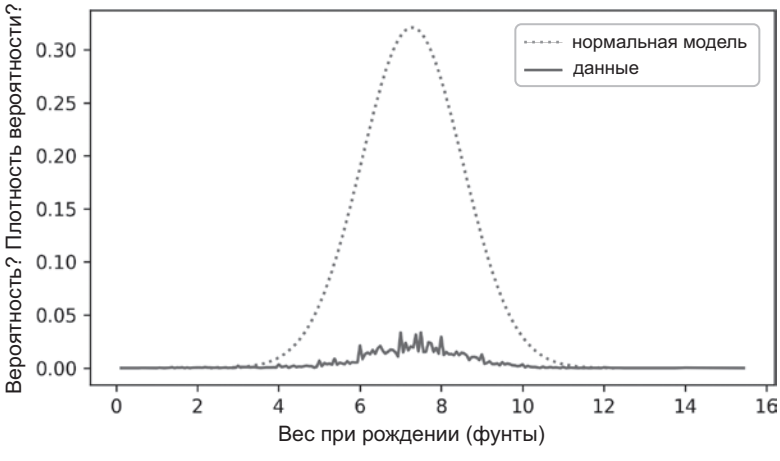
Распространенной ошибкой является сравнение ФВ некоторой выборки с ФПВ теоретической модели. Например, предположим, что требуется сравнить распределение веса новорожденных с нормальной моделью. Вот объект `Pmf`, представляющий распределение данных:

```
pmf_birth_weight = Pmf.from_seq(birth_weights, name="данные")
```

У нас также имеется объект `pdf_model`, представляющий ФПВ нормального распределения с теми же средним значением и стандартным отклонением. Вот что произойдет, если мы изобразим их на одном графике:

```
pdf_model.plot(ls=":", color="gray")
pmf_birth_weight.plot()

decorate(xlabel="Вес при рождении (фунты)", ylabel="Вероятность?
Плотность вероятности?")
```



Так делать не стоит. Хотя бы по причине того, что эти распределения имеют разные единицы измерения. ФВ содержит значения вероятности, а PDF — плотности вероятности, поэтому нельзя сравнивать их и строить их графики в одних и тех же осях координат.

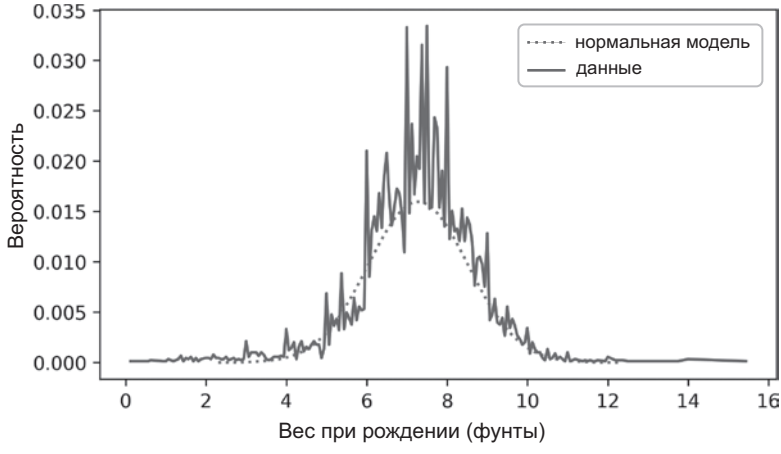
Один из способов решить эту проблему — создать объект `Pmf`, аппроксимирующий нормальное распределение путем вычисления ФПВ на дискретном наборе точек. В классе `NormalPdf` есть метод `make_pmf`, который это делает:

```
pmf_model = pdf_model.make_pmf()
```

Результатом является нормированный объект `Pmf`, содержащий вероятностные меры. Поэтому можно по крайней мере построить его график в тех же осях, что и ФВ данных:

```
pmf_model.plot(ls=":", color="gray")
pmf_birth_weight.plot()

decorate(xlabel="Вес при рождении (фунты)", ylabel="Вероятность")
```



Но это все еще плохой способ сравнения распределений. Одна из проблем заключается в том, что в этих двух объектах `Pmf` содержится разное количество величин, к тому же в `pmf_birth_weight` они распределены неравномерно. По этой причине вероятностные меры этих распределений едва ли сопоставимы:

```
len(pmf_model), len(pmf_birth_weight)
```

```
(201, 184)
```

Другим препятствием является зашумленность `Pmf`-данных. Поэтому поищем другое решение.

Ядерная оценка плотности

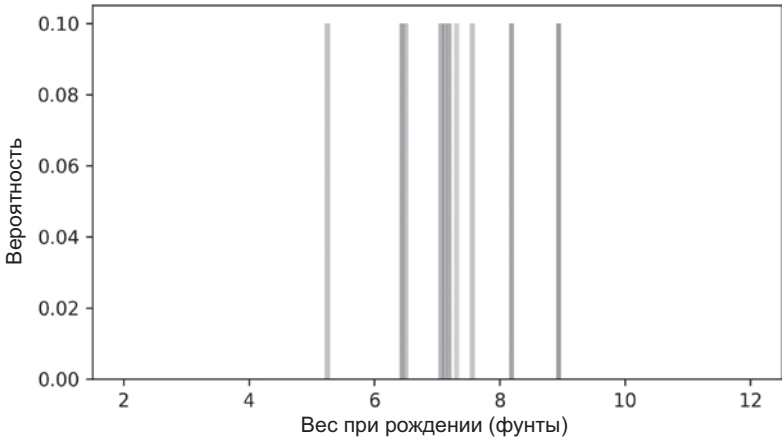
Вместо создания ФВ с помощью модельного распределения можно использовать данные для конструирования ФПВ. Чтобы посмотреть, как это работает, начнем с небольшой выборки из нашего датасета:

```
n = 10
sample = birth_weights.sample(n)
```

ФВ этой выборки выглядит следующим образом:

```
for weight in sample:
    pmf = Pmf.from_seq([weight]) / n
    pmf.bar(width=0.08, alpha=0.5)

xlim = [1.5, 12.5]
decorate(xlabel="Вес при рождении (фунты)", ylabel="Вероятность", xlim=xlim)
```

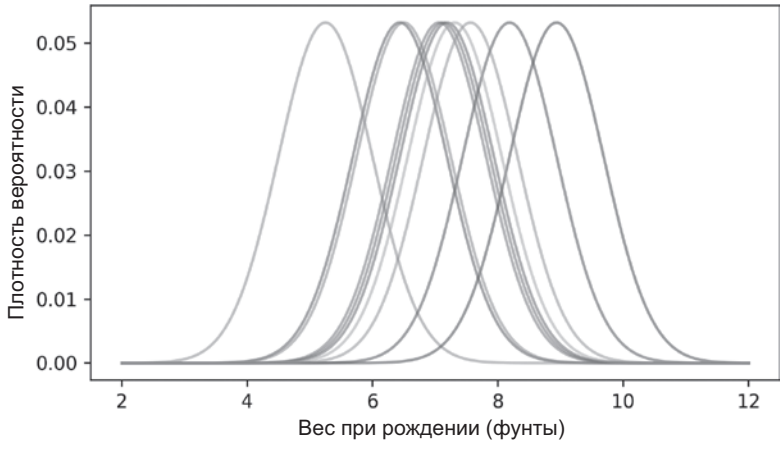


При таком способе представления распределения данные рассматриваются так, как если бы они были дискретными, поэтому каждая вероятностная мера концентрируется в одной точке. Но вес новорожденных на самом деле непрерывен, поэтому в промежутках между элементами выборки также могут присутствовать данные. Учесть эту возможность можно, заменив каждую дискретную вероятностную меру непрерывной плотностью вероятности, как показано ниже:

```
qs = np.linspace(2, 12, 201)

for weight in sample:
    ps = NormalPdf(weight, 0.75)(qs) / n
    plt.plot(qs, ps, alpha=0.5)

decorate(xlabel="Вес при рождении (фунты)", ylabel="Плотность вероятности",
xlim=xlim)
```

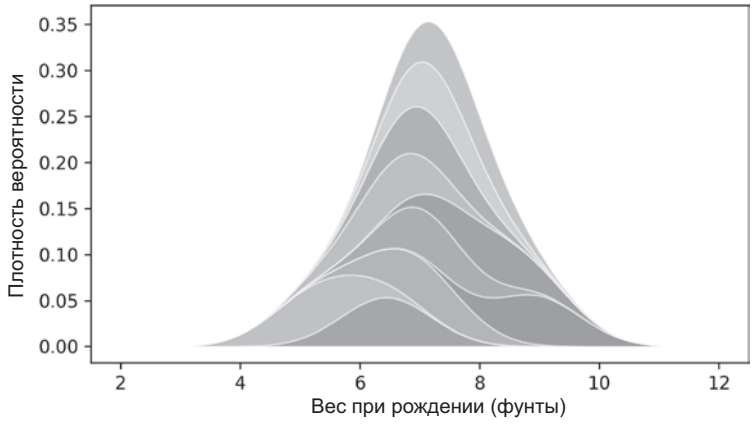


Для каждого значения веса в выборке создадим `NormalPdf`, у которого наблюдаемый вес служит средним значением, а затем сложим их:

```
low_ps = np.zeros_like(qs)

for weight in sample:
    ps = NormalPdf(weight, 0.75)(qs) / n
    high_ps = low_ps + ps
    plt.fill_between(qs, low_ps, high_ps, alpha=0.5, lw=1, ec="white")
    low_ps = high_ps

decorate(xlabel="Вес при рождении (фунты)", ylabel="Плотность вероятности",
xlim=xlim)
```



Когда мы складываем плотности вероятности в каждой точке данных, результатом является оценка плотности вероятности для всей выборки. Этот процесс называется **ядерной оценкой плотности, ЯОП** (kernel density estimation, KDE). В данном контексте «ядро» — это одна из тех элементарных функций плотности, которые мы суммировали. Поскольку ядра, которые мы использовали, являются нормальными распределениями, также известными как гауссовы, можно конкретизировать, что мы вычислили гауссову ЯОП.

В библиотеке SciPy определен класс `gaussian_kde`, который реализует этот алгоритм. Вот как можно его использовать для оценки распределения веса новорожденных:

```
from scipy.stats import gaussian_kde

kde = gaussian_kde(birth_weights)
```

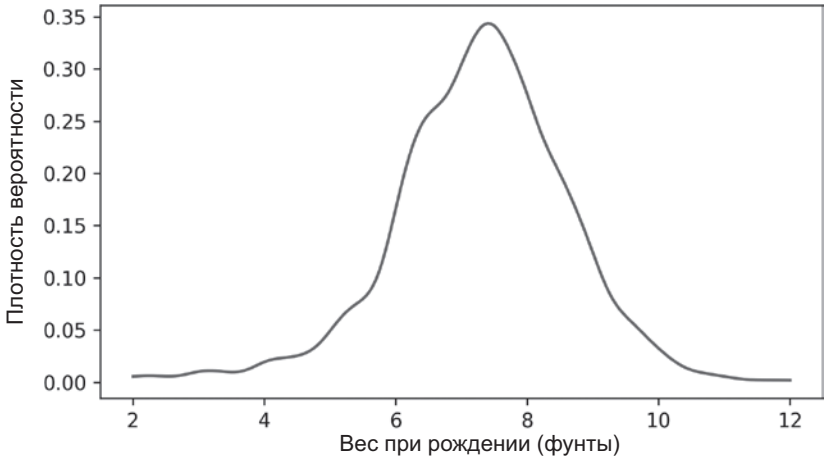
Результатом является объект, представляющий оцениваемую ФПВ. Последнюю можно вычислить, если вызвать объект как функцию:

```
ps = kde(qs)
```

Вот как выглядит полученное распределение:

```
plt.plot(qs, ps)

decorate(xlabel="Вес при рождении (фунты)", ylabel="Плотность вероятности")
```



В модуле `thinkstats` определен объект `Pdf`, при создании которого передается результат выполнения `gaussian_kde` и интервал значений (domain), на котором необходимо вычислить оценку плотности. Вот как его можно создать:

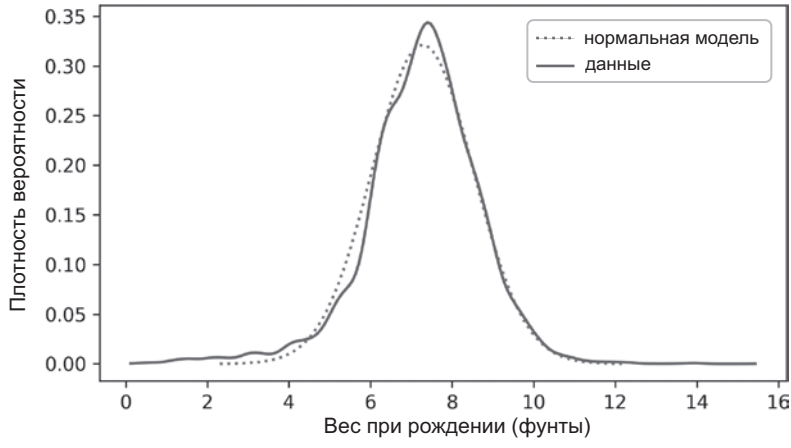
```
from thinkstats import Pdf

domain = np.min(birth_weights), np.max(birth_weights)
kde_birth_weights = Pdf(kde, domain, name="данные")
```

В `Pdf` реализован метод отображения `plot`, который можно использовать для сравнения оценки плотности выборки и ФПВ нормального распределения:

```
pdf_model.plot(ls=":", color="gray")
kde_birth_weights.plot()

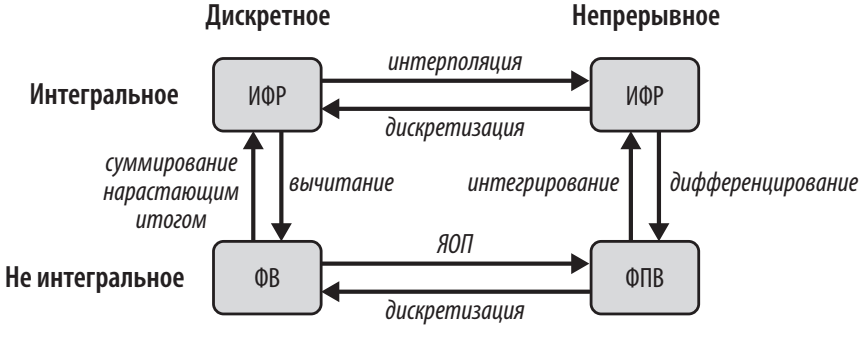
decorate(xlabel="Вес при рождении (фунты)", ylabel="Плотность вероятности")
```



Ядерная оценка плотности позволяет сравнить распределение данных с теоретической моделью. В некоторых сценариях такой способ визуального представления сравнения может подойти. Но тем, кто знаком с ИФР, лучше использовать их: сравнение двух ИФР, как правило, более эффективно.

Схема взаимосвязи распределений

Итак, у нас есть полный набор способов представления распределений: ФВ, ИФР и ФПВ. На следующей схеме указаны все эти способы и переходы между ними. Например, если у нас есть ФВ, то для вычисления накопленной суммы вероятностей можно использовать функцию `cumsum`, что дает нам ИФР того же распределения:



Чтобы продемонстрировать эти переходы, воспользуемся новым датасетом, который, согласно описанию, «содержит время рождения, пол и вес при рождении

для каждого из 44 младенцев, родившихся за один промежуток времени в 24 часа в больнице австралийского города Брисбен». Инструкции по загрузке данных приведены в Jupyter-блокноте для этой главы.

Данные можно считать следующим образом:

```
from thinkstats import read_baby_boom

boom = read_baby_boom()
boom.head()
```

	time (время)	sex (пол)	weight_g (вес_г)	minutes (минуты)
0	5	1	3837	5
1	104	1	3334	64
2	118	2	3554	78
3	155	2	3838	115
4	257	2	3625	177

В столбце `minutes` указано «количество минут, прошедших с начала суток до каждого рождения». Таким образом, для вычисления интервала между двумя последовательными рождениями можно использовать метод `diff`:

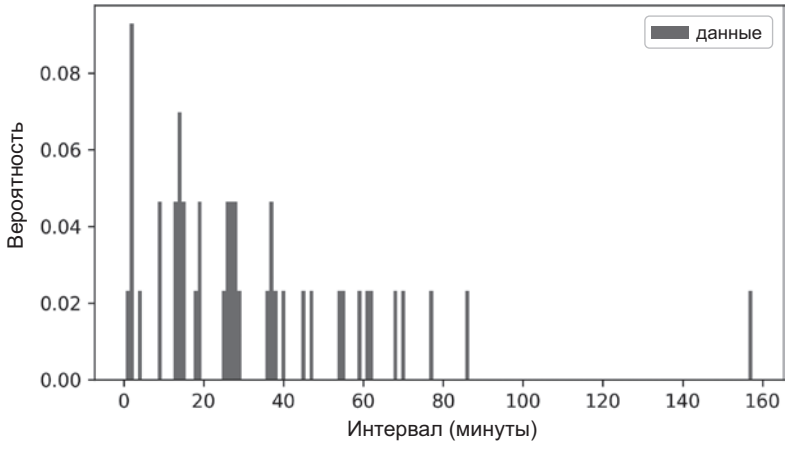
```
diffs = boom["minutes"].diff().dropna()
```

Если роды происходят с одинаковой вероятностью в любую минуту дня, стоит ожидать, что эти интервалы будут распределены по экспоненте. На самом деле это не совсем так, но экспоненциальная модель все же может неплохо подойти для этих данных.

Чтобы убедиться в этом, начнем с создания объекта `Pmf`, представляющего распределение интервалов:

```
pmf_diffs = Pmf.from_seq(diffs, name="данные")
pmf_diffs.bar(width=1)

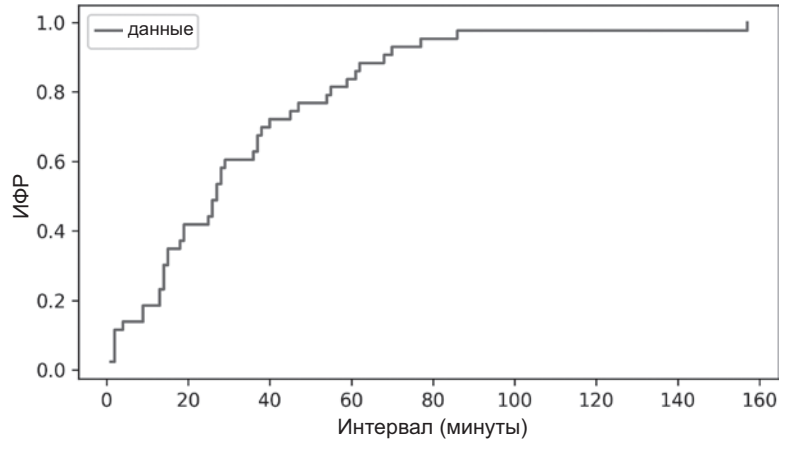
decorate(xlabel="Интервал (минуты)", ylabel="Вероятность")
```



Теперь для вычисления интегрального распределения вероятности и создания объекта `Cdf` воспользуемся методом `make_cdf`:

```
cdf_diffs = pmf_diffs.make_cdf()
cdf_diffs.step()

decorate(xlabel="Интервал (минуты)", ylabel="ИФР")
```



Объекты `Pmf` и `Cdf` эквивалентны с той точки зрения, что если известен один из них, то можно вычислить второй. Для примера воспользуемся методом `make_pmf`, который вычисляет разность между последовательными значениями интегральной вероятности в объекте `Cdf` и возвращает объект `Pmf`:

```
pmf_diffs2 = cdf_diffs.make_pmf()
```

Результат должен быть идентичен исходному `Pmf`, но могут быть небольшие отличия из-за погрешности арифметики с плавающей точкой. Чтобы проверить, что результат близок к исходному `Pmf`, можно вызвать функцию `allclose`:

```
np.allclose(pmf_diffs, pmf_diffs2)
```

```
True
```

Ответ положительный.

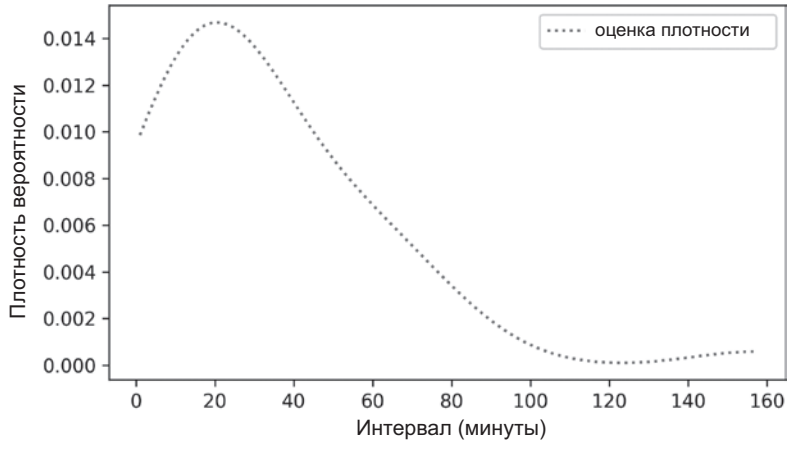
Если имеется объект `Pmf`, то оценку функции плотности можно получить путем вызова `gaussian_kde`, указывая вероятности из `Pmf` в качестве весов (weights):

```
kde = gaussian_kde(pmf_diffs.qs, weights=pmf_diffs.ps)
```

Чтобы отобразить результаты, можно из объекта `gaussian_kde` создать объект класса `Pdf` и у последнего вызвать метод `plot`:

```
domain = np.min(pmf_diffs.qs), np.max(pmf_diffs.qs)
kde_diffs = Pdf(kde, domain=domain, name="оценка плотности")

kde_diffs.plot(ls=":", color="gray")
decorate(xlabel="Интервал (минуты)", ylabel="Плотность вероятности")
```



Чтобы проверить, соответствует ли оценка плотности экспоненциальной модели, можно создать объект `ExponentialCdf` с тем же средним значением, что и данные:

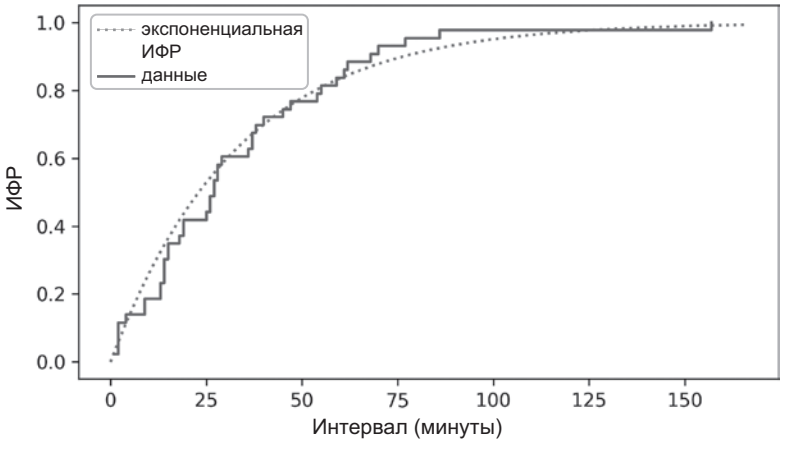
```
from thinkstats import ExponentialCdf

m = diffs.mean()
lam = 1 / m
cdf_model = ExponentialCdf(lam, name="экспоненциальная ИФР")
```

Вот как экспоненциальная ИФР выглядит в сравнении с ИФР данных:

```
cdf_model.plot(ls=":", color="gray")
cdf_diffs.step()

decorate(xlabel="Интервал (минуты)", ylabel="ИФР")
```

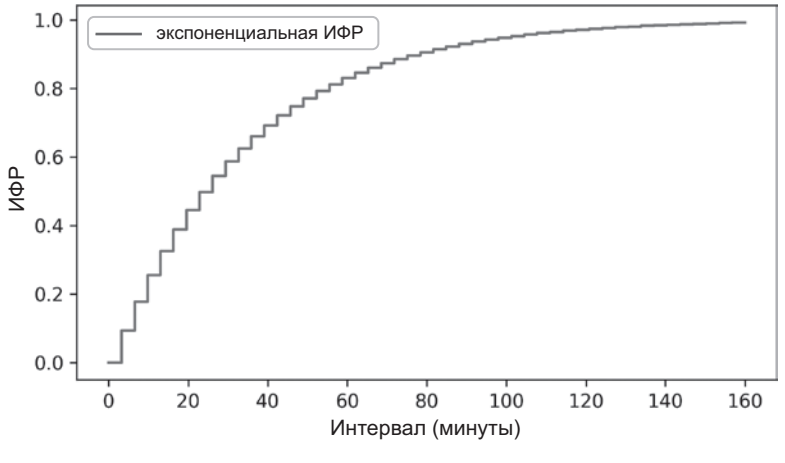


Экспоненциальная модель хорошо соответствует ИФР данных.

У объекта `ExponentialCdf` можно вызвать метод `make_cdf` для **дискретизации** ИФР, то есть для получения дискретной аппроксимации путем вычисления ИФР для последовательности равномерно распределенных величин:

```
discrete_cdf_model = cdf_model.make_cdf(qs)
discrete_cdf_model.step()

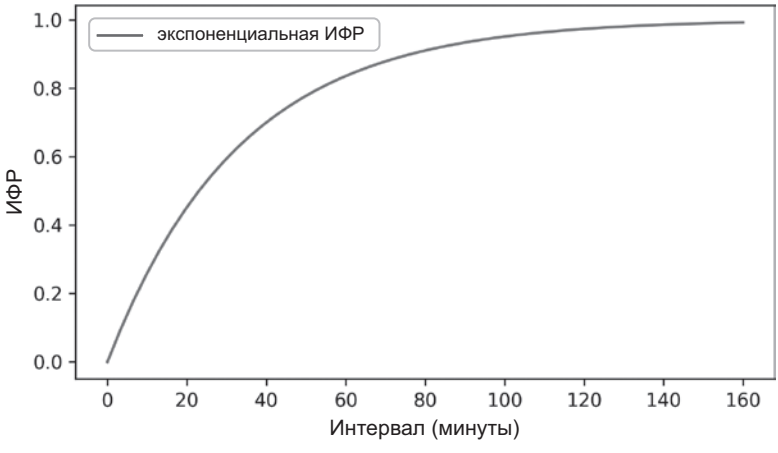
decorate(xlabel="Интервал (минуты)", ylabel="ИФР")
```



Чтобы перейти от дискретной ИФР к непрерывной ИФР, можно выполнить интерполяцию между шагами. Именно это происходит, когда мы используем метод `plot` вместо метода `step`:

```
discrete_cdf_model.plot(color="gray")

decorate(xlabel="Интервал (минуты)", ylabel="ИФР")
```



Наконец, ФПВ — это производная непрерывной ИФР, а ИФР — это интеграл ФПВ.

На этом примере вы узнали, как для работы с ФВ, ИФР и ФПВ можно использовать объекты классов `Pmf`, `Cdf` и `Pdf` соответственно, а также преобразовывать одни в другие.

Глоссарий

Непрерывный (continuous)

Величина является непрерывной, если она может принимать любое значение в некотором диапазоне на числовой оси. Большинство измеряемых нами физических величин, таких как вес, расстояние и время, являются непрерывными.

Дискретный (discrete)

Величина является дискретной, если она может принимать ограниченный набор значений, например целых чисел или категорий. Точные значения являются дискретными, так же как категориальные переменные.

Функция плотности вероятности, ФПВ (probability density function, PDF)

Функция, которая показывает, как плотность вероятности (не вероятность) распределена по значениям непрерывной переменной. Площадь под графиком

ФПВ в некотором интервале значений дает вероятность того, что случайная величина попадет в этот интервал.

Плотность вероятности (probability density)

Значение ФПВ в некоторой точке; само по себе оно не является вероятностью, но его можно использовать для вычисления вероятности.

Ядерная оценка плотности, ЯОП (kernel density estimation, KDE)

Метод оценки ФПВ на основе выборки.

Дискретизация (discretization)

Аппроксимация непрерывной величины путем деления ее диапазона на дискретные уровни или категории.

Упражнения

Упражнение 6.1

Предположим, что время до первого забитого гола в матче чемпионата мира по футболу хорошо моделируется экспоненциальным распределением с коэффициентом результативности `lam=2.5` гола за игру. Создайте объект `ExponentialPdf`, представляющий это распределение, и используйте метод `area_under` для вычисления вероятности того, что время до первого гола составит меньше половины матча. Затем используйте объект `ExponentialCdf`, чтобы вычислить ту же вероятность и проверить согласованность обоих результатов.

Используйте объект `ExponentialPdf` для вычисления вероятности того, что первый гол будет забит во второй половине матча. Затем используйте объект `ExponentialCdf` для вычисления той же вероятности и проверьте соответствие результатов.

Упражнение 6.2

Чтобы стать членом Blue Man Group, вы должны быть мужчиной ростом от 5 футов и 10 дюймов до 6 футов и 1 дюйма, то есть примерно от 178 до 185 см. Узнаем, какая часть взрослого мужского населения Соединенных Штатов соответствует этому требованию.

Рост (heights) участников исследования BRFSS мужского пола (male) хорошо моделируется с помощью нормального распределения: среднее значение составляет 178 см, а стандартное отклонение — 7 см:

```
from thinkstats import read_brfss

brfss = read_brfss()
male = brfss.query("sex == 1")
heights = male["htm3"].dropna()
```

```
from scipy.stats import trimboth

trimmed = trimboth(heights, 0.01)
m, s = np.mean(trimmed), np.std(trimmed)
m, s
```

```
(178.10278947124948, 7.017054887136004)
```

Вот объект `NormalCdf`, который представляет нормальное распределение с теми же средним значением и стандартным отклонением, что и усеченные данные:

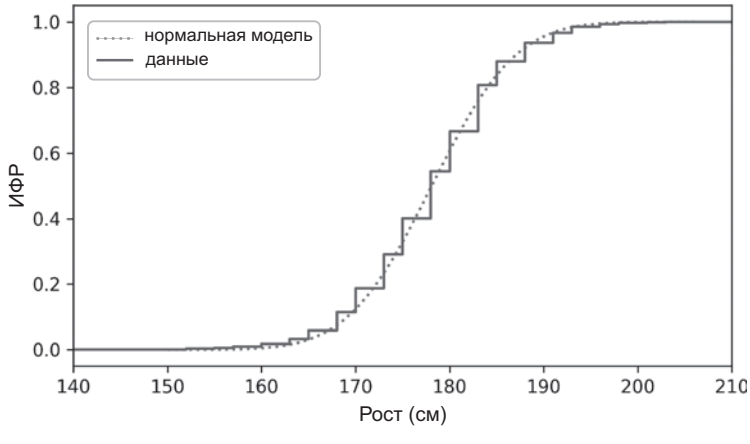
```
from thinkstats import NormalCdf

cdf_normal_model = NormalCdf(m, s, name='нормальная модель')
```

И вот как оно соотносится с ИФР данных:

```
cdf_height = Cdf.from_seq(heights, name="данные")
cdf_normal_model.plot(ls=":", color="gray")
cdf_height.step()

xlim = [140, 210]
decorate(xlabel="Рост (см)", ylabel="ИФР", xlim=xlim)
```



Используйте функцию `gaussian_kde` для создания объекта `Pdf`, аппроксимирующего ФПВ роста мужчин. Подсказка: изучите аргумент `bw_method`, предназначенный для настройки гладкости оценки плотности. Постройте график оценки плотности и сравните его с нормальным распределением `NormalPdf` со средним значением `m` и стандартным отклонением `s`.

С помощью объекта `NormalPdf` и метода `area_under` вычислите долю людей ростом от 178 до 185 см в рамках нормальной модели. Используйте объект класса `NormalCdf` для вычисления той же доли и убедитесь, что результаты не противоречат друг другу. Наконец, используйте объект `Cdf` эмпирических данных, чтобы определить, какая доля людей, описанных в датасете, находится в том же диапазоне.

ГЛАВА 7

Связь между переменными

До сих пор мы рассматривали переменные по отдельности. В этой главе начнем обсуждать связи между переменными. Две переменные взаимосвязаны, если знание одной дает информацию о второй. Например, существует связь между ростом и весом: люди более высокого роста, как правило, тяжелее. Конечно, здесь нет однозначного соответствия: есть невысокие полные и высокие худые люди. Но если вы пытаетесь наугад определить вес человека, то сделать это удастся точнее, когда вам известен его рост, чем в случае отсутствия такой информации.

В этой главе описано несколько способов визуализации связи между переменными и один из способов количественной оценки силы взаимосвязи — коэффициент корреляции.

Диаграмма рассеяния

Если вы встретите человека, отлично разбирающегося в математике, будете ли вы ожидать, что он так же хорош в знании языков? С одной стороны, можно предположить, что люди обычно являются специалистами только в одной области, и поэтому тот, кто силен в одной, будет слабее в другой. С другой стороны, можно ожидать, что навыки интеллектуально развитого человека будут выше среднего уровня в обеих областях. Выясним, какая из этих двух гипотез верна.

Мы будем использовать данные Национального лонгитюдного исследования молодежи США 1997 года (National Longitudinal Survey of Youth, NLSY97), которое «отслеживает жизнь 8984 молодых американцев, родившихся в 1980–84 годах». Общедоступный датасет включает в себя оценки участников по нескольким стандартизированным тестам, включая тесты, наиболее часто используемые в США при поступлении в колледж, — SAT и ACT. Поскольку сдающие тест получают отдельные баллы за математическую и языковую часть, можно воспользоваться этими данными для исследования взаимосвязи между математическими и лингвистическими способностями.

Инструкции по загрузке данных приведены в Jupyter-блокноте для этой главы. Для чтения данных и замены специальных кодов отсутствующих данных (`missing_codes`) на `np.nan` можно использовать Pandas-функции `read_csv` и `replace`:

```
missing_codes = [-1, -2, -3, -4, -5]
nlsy = pd.read_csv("nlsy97-extract.csv.gz").replace(missing_codes, np.nan)
nlsy.shape
```

```
(8984, 34)
```

Таблица `DataFrame` содержит по одной строке для каждого из 8984 участников исследования и по одному столбцу для каждой из 34 выбранных мной переменных. Имена столбцов как таковые не имеют особого смысла, поэтому заменим названия тех, которые мы будем использовать, на более понятные:

```
nlsy["sat_verbal"] = nlsy["R9793800"]
nlsy["sat_math"] = nlsy["R9793900"]
```

Оба столбца содержат несколько значений меньше 200, что невозможно, поскольку 200 — это наименьший балл. Поэтому заменим их на `np.nan`:

```
columns = ["sat_verbal", "sat_math"]

for column in columns:
    invalid = nlsy[column] < 200
    nlsy.loc[invalid, column] = np.nan
```

Далее будем использовать метод `dropna` для выбора только тех строк, где оба балла имеют корректные значения:

```
nlsy_valid = nlsy.dropna(subset=columns).copy()
nlsy_valid.shape
```

```
(1398, 36)
```

Баллы SAT стандартизированы, поэтому среднее значение равно 500, а стандартное отклонение — 100. В выборке NLSY средние значения и стандартные отклонения близки к этим величинам:

```
sat_verbal = nlsy_valid["sat_verbal"]
sat_verbal.mean(), sat_verbal.std()
```

```
(501.80972818311875, 108.36562024213643)
```

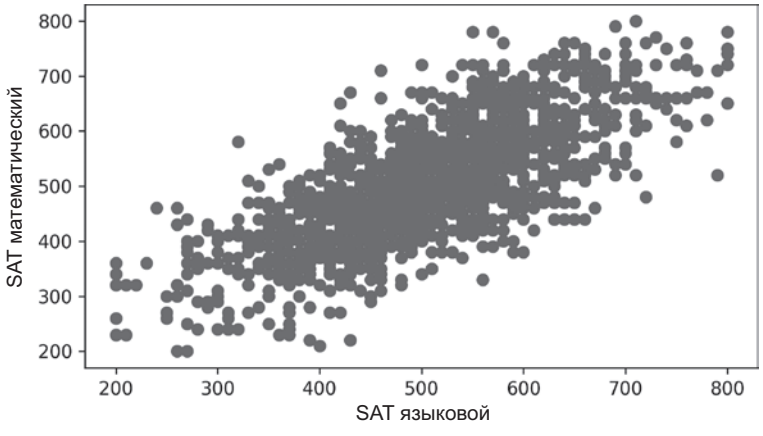
```
sat_math = nlsy_valid["sat_math"]
sat_math.mean(), sat_math.std()
```

```
(503.0829756795422, 109.8329973731453)
```

Теперь, чтобы узнать, существует ли связь между этими переменными, посмотрим на **диаграмму рассеяния** (scatter plot):

```
plt.scatter(sat_verbal, sat_math)

decorate(xlabel="SAT языковой", ylabel="SAT математический")
```



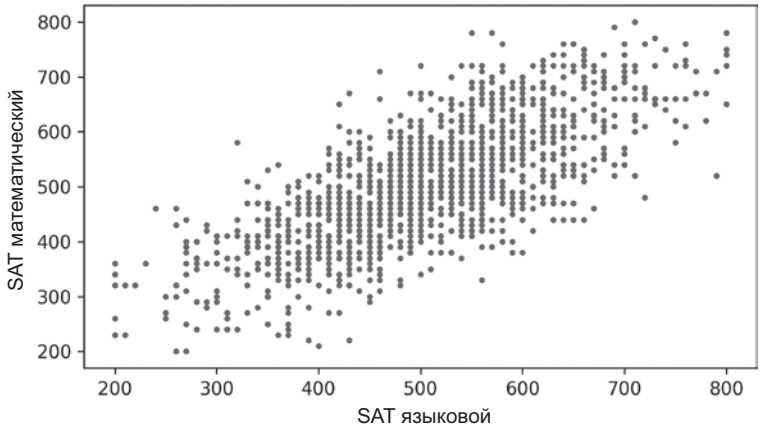
Используя параметры функции `scatter` по умолчанию, можно увидеть общую форму взаимосвязи. Учащиеся, которые хорошо справляются с одной частью теста, как правило, лучше справляются и с другой.

Однако такое представление диаграммы **перенасыщено** (overplotted), что означает присутствие множества накладывающихся друг на друга точек, которые могут дать обманчивое впечатление о взаимосвязи. Центр, где плотность точек максимальна, не такой темный, каким должен быть, и наоборот — значения по краям темнее, чем должны быть. Перенасыщение, как правило, придает слишком большой визуальный вес выбросам.

Мы можем улучшить диаграмму, уменьшив размер точек, чтобы они меньше перекрывали друг друга:

```
plt.scatter(sat_verbal, sat_math, s=5)

decorate(xlabel="SAT языковой", ylabel="SAT математический")
```



Теперь становится видно, что из-за округления баллов до ближайшего значения, кратного 10, точки выровнены по строкам и столбцам. То есть при обработке была потеряна часть информации.

Мы не можем восстановить потерянную информацию, но можем минимизировать ее эффект на точечную диаграмму, используя **джиттеринг** (jittering) данных — добавление случайного шума для компенсации эффекта округления. Функция ниже принимает последовательность данных и зашумляет ее, добавляя случайные значения из нормального распределения со средним значением 0 и заданным стандартным отклонением. Функция возвращает NumPy-массив:

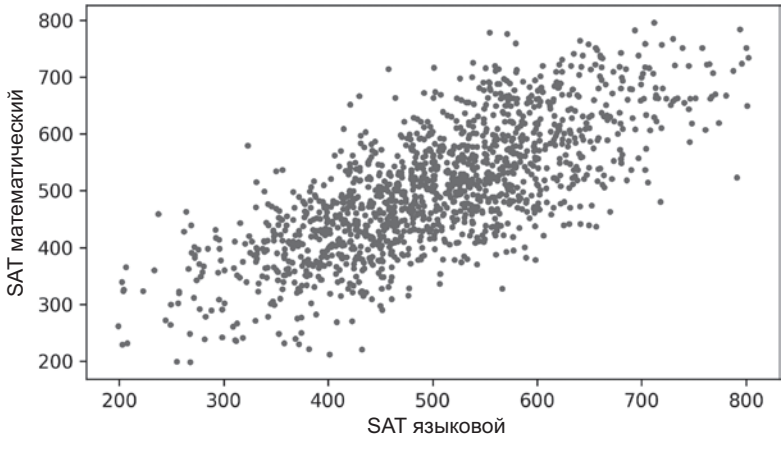
```
def jitter(seq, std=1):
    n = len(seq)
    return np.random.normal(0, std, n) + seq
```

Если применить джиттеринг со стандартным отклонением 3 к нашим данным, то строки и столбцы пропадут с диаграммы рассеяния:

```
sat_verbal_jittered = jitter(sat_verbal, 3)
sat_math_jittered = jitter(sat_math, 3)
```

```
plt.scatter(sat_verbal_jittered, sat_math_jittered, s=5)

decorate(xlabel="SAT языковой", ylabel="SAT математический")
```

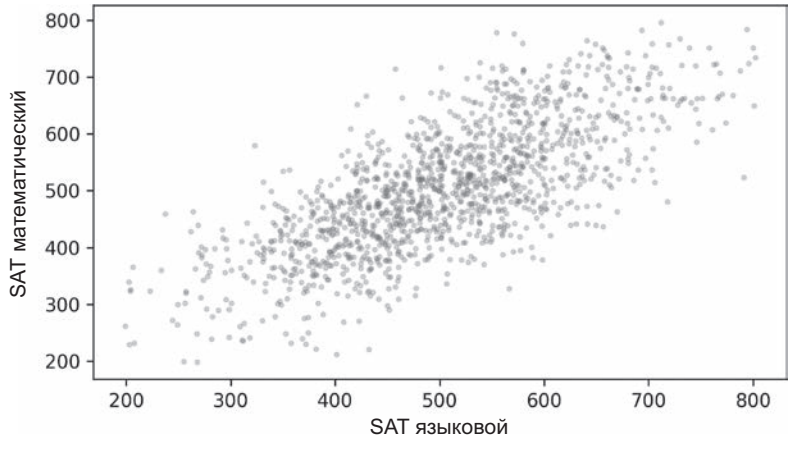


Джиттеринг уменьшает визуальный эффект округления и делает форму взаимосвязи четче. Но вообще случайный шум следует добавлять в данные только в целях визуализации и не использовать его для анализа.

В данном примере даже после корректировки размера точки и применения джиттеринга данных все равно остается небольшое перенасыщение. Поэтому добавим еще одну настройку: воспользуемся параметром `alpha`, чтобы сделать точки частично прозрачными:

```
plt.scatter(sat_verbal_jittered, sat_math_jittered, s=5, alpha=0.2)

decorate(xlabel="SAT языковой", ylabel="SAT математический")
```



При использовании прозрачности перекрывающиеся точки данных становятся темнее, поэтому потемнение пропорционально их плотности.

Хотя диаграммы рассеяния — простое и широко используемое средство визуализации, их бывает сложно «довести до ума». Обычно требуется сделать несколько попыток подбора размера, прозрачности и джиттеринга, чтобы найти оптимальный режим отображения связи между переменными.

Децильная диаграмма

Диаграмма рассеяния дает общее представление о связи между переменными, но существуют и другие типы визуализации, позволяющие лучше понять природу связи. Один из них — **децильная диаграмма** (decile plot).

Чтобы построить децильную диаграмму, нужно отсортировать записи респондентов по баллу языкового теста и разделить их на 10 групп, называемых **децилями**. Для этих вычислений можно использовать функцию Pandas `qcut`:

```
deciles = pd.qcut(nlsy_valid["sat_verbal"], 10, labels=False) + 1
deciles.value_counts().sort_index()
```

```
sat_verbal
```

```
1     142
2     150
3     139
4     140
5     159
6     130
7     148
8     121
9     138
10    131
Name: count, dtype: int64
```

Количество респондентов в каждом дециле примерно одинаково.

Теперь для разделения датафрейма на группы по номеру в Pandas-серии `deciles` можно использовать метод `groupby`:

```
df_groupby = nlsy_valid.groupby(deciles)
df_groupby
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7f369c918a60>
```

Результат, представляющий сгруппированные данные, имеет тип `DataFrameGroupBy`. Выберем в нем столбец `sat_math`:

```
series_groupby = df_groupby["sat_math"]
series_groupby
```

```
<pandas.core.groupby.generic.SeriesGroupBy object at 0x7f369fdb3760>
```

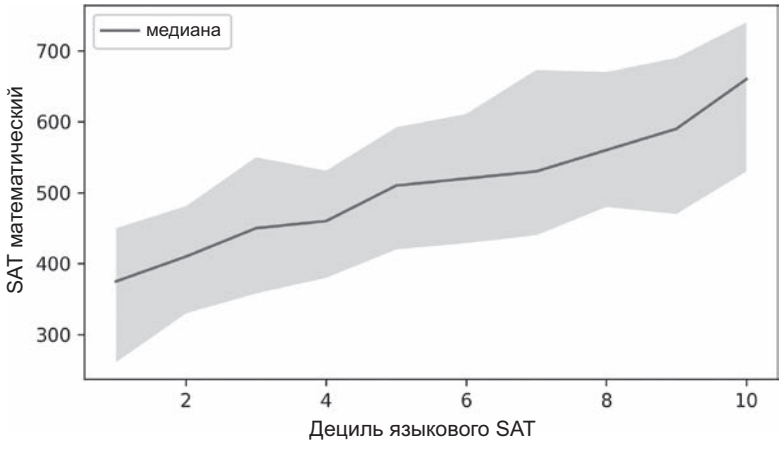
Результат является объектом типа `SeriesGroupBy` и представляет баллы по математике, сгруппированные по децилю. Для вычисления 10-го, 50-го и 90-го процентилей в каждой группе можно вызвать метод `quantile`:

```
low = series_groupby.quantile(0.1)
median = series_groupby.quantile(0.5)
high = series_groupby.quantile(0.9)
```

На децильной диаграмме эти процентили показаны для каждой децильной группы. На следующем рисунке линия обозначает медиану, а затемненная область — диапазон между 10-м и 90-м процентилями:

```
xs = median.index
plt.fill_between(xs, low, high, alpha=0.2)
plt.plot(xs, median, label="медиана")

decorate(xlabel="Дециль языкового SAT", ylabel="SAT математический")
```

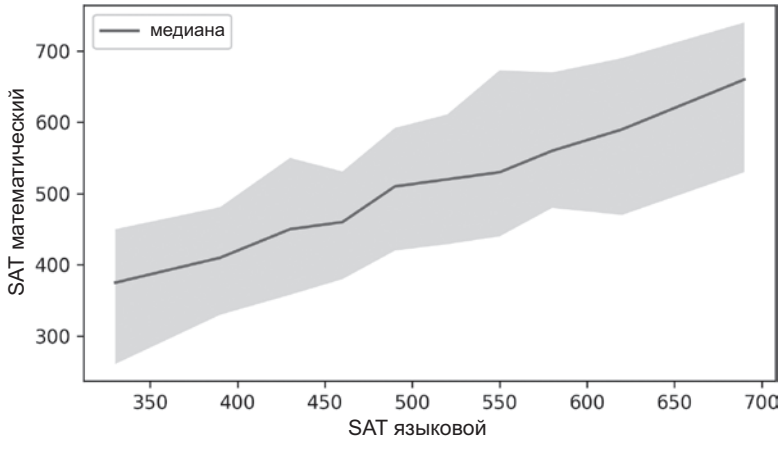


Так же можно вычислить средний балл языкового теста в каждой группе и отложить эти значения по оси x вместо номеров децилей:

```
xs = df_groupby["sat_verbal"].median()

plt.fill_between(xs, low, high, alpha=0.2)
plt.plot(xs, median, color="C0", label="медиана")

decorate(xlabel="SAT языковой", ylabel="SAT математический")
```



Складывается впечатление, что связь между этими переменными линейна: то есть каждому приращению среднего балла по языковому тесту соответствует примерно одинаковое увеличение среднего балла по математике.

Вообще респондентов можно разделить на любое произвольное количество групп, необязательно на 10, и по каждой группе в дополнение к процентилям рассчитать другие сводные статистические показатели.

Корреляция

Когда участники исследования NLSY учились в девятом классе, многие из них сдавали математическую часть индивидуального теста успеваемости Пибоди (Peabody Individual Achievement Test, PIAT). Дадим столбцу с этими результатами более понятное название:

```
nlsy["piat_math"] = nlsy["R1318200"]
nlsy["piat_math"].describe()
```

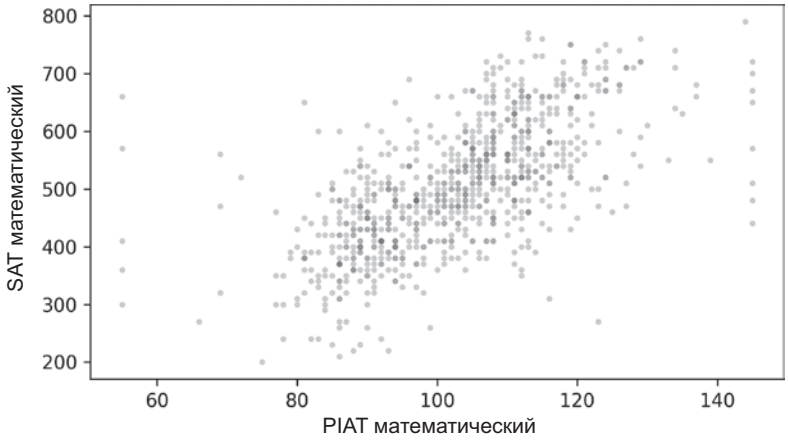
```
count    6044.000000
mean       93.903706
std        14.631148
min        55.000000
25%        84.000000
50%        92.000000
75%       103.000000
max       145.000000
Name: piat_math, dtype: float64
```

Учащиеся, которые в девятом классе хорошо сдали PIAT, скорее всего, в двенадцатом классе получат высокие баллы по математическому разделу SAT. На следующей диаграмме рассеяния для участников исследования NLSY, сдававших оба теста, показана связь между их баллами. Она построена с помощью функции `scatter` из модуля `thinkstats`, которая регулирует размер и прозрачность точки, а также при необходимости добавляет джиттеринг:

```
from thinkstats import scatter

scatter(nlsy, "piat_math", "sat_math")

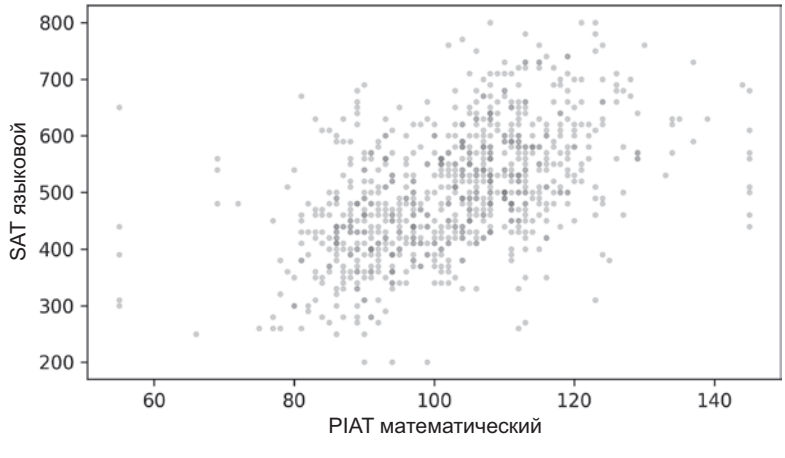
decorate(xlabel="PIAT математический", ylabel="SAT математический")
```



Как и ожидалось, ученики, хорошо сдавшие PIAT, с большей вероятностью получают высокий балл по математическому разделу SAT. И если математические и языковые способности взаимосвязаны, то стоит предположить, что эти ученики также преуспели в сдаче языкового раздела SAT. На следующем рисунке показана взаимосвязь между баллами PIAT и баллами языковой части SAT:

```
scatter(nlsy, "piat_math", "sat_verbal")

decorate(xlabel="PIAT математический", ylabel="SAT языковой")
```



Учащиеся с более высокими баллами PIAT также в среднем имеют более высокие баллы языкового SAT.

Сравнивая диаграммы рассеяния, складывается впечатление, что точки на первом рисунке расположены компактнее, а на втором — разреженнее. Если так, то это означает, что математические баллы PIAT более точно предсказывают результаты SAT по математике, чем баллы по языковой части SAT, — и это логично.

Чтобы количественно оценить силу этих взаимосвязей, можно посчитать **коэффициент корреляции Пирсона**, который часто называют просто **корреляцией**. Чтобы разобраться в корреляции, начнем со стандартизации.

Чтобы стандартизировать переменную, нужно вычесть среднее значение и разделить на стандартное отклонение — именно то, что делает эта функция:

```
def standardize(xs):
    return (xs - np.mean(xs)) / np.std(xs)
```

Чтобы показать, как она работает, выберем строки с корректными (`valid`) значениями `piat_math` и `sat_math`:

```
valid = nlsy.dropna(subset=["piat_math", "sat_math"])
piat_math = valid["piat_math"]
sat_math = valid["sat_math"]
```

И стандартизуем баллы PIAT по математике:

```
piat_math_standard = standardize(piat_math)
np.mean(piat_math_standard), np.std(piat_math_standard)
```

```
(-2.4321756236287047e-16, 1.0)
```

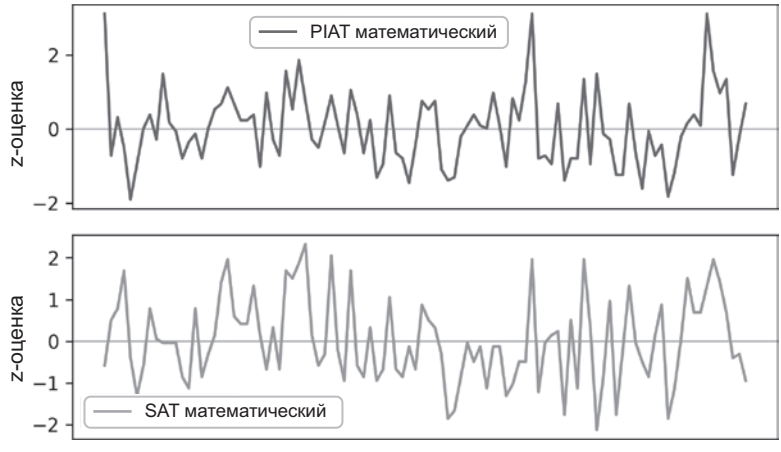
Полученные результаты представляют собой **стандартную оценку** (standard score), также называемую **z-оценкой** (z-score). Благодаря процедуре расчета стандартной оценки ее среднее значение близко к 0, а стандартное отклонение — к 1.

Стандартизируем также баллы SAT по математике:

```
sat_math_standard = standardize(sat_math)
np.mean(sat_math_standard), np.std(sat_math_standard)
```

```
(-1.737268302591932e-16, 0.9999999999999998)
```

На этом рисунке показана последовательность стандартных оценок для первых 100 участников:



Эти переменные очевидно связаны: если одна из них превышает среднее значение, то и другая, чаще всего, тоже выше среднего. Чтобы количественно оценить силу этой связи, перемножим стандартные оценки поэлементно и вычислим среднее значение произведения.

Когда обе оценки положительны, их произведение тоже положительно, что приводит к увеличению среднего значения. Когда обе оценки отрицательны, их произведение становится положительным, что также повышает среднее. Когда оценки имеют разный знак, их произведение отрицательно, что уменьшает среднее. Как результат, среднее значение произведения измеряет сходство между последовательностями:

```
np.mean(piat_math_standard * sat_math_standard)
```

```
0.639735816517885
```

Полученное значение около 0.64 — это коэффициент корреляции. Вот один из способов его интерпретации: если чей-то балл PIAT по математике на одно стандартное отклонение превышает среднее значение, то можно ожидать, что его балл SAT по математике среднестатистически тоже будет на 0.64 стандартных отклонения выше среднего.

Если перемножить элементы в обратном порядке, результат не изменится:

```
np.mean(sat_math_standard * piat_math_standard)
```

```
0.639735816517885
```

Таким образом, коэффициент корреляции симметричен: если чей-то балл SAT по математике на одно стандартное отклонение выше среднего, то ожидается, что его балл PIAT по математике статистически будет на 0.64 стандартных отклонения выше среднего.

Корреляция — это широко используемый статистический показатель, поэтому в библиотеке NumPy есть функция для ее вычисления:

```
np.corrcoef(piat_math, sat_math)
```

```
array([[1.       , 0.63973582],
       [0.63973582, 1.        ]])
```

На выходе получается **корреляционная матрица** с одной строкой и одним столбцом для каждой переменной. Значение в верхнем левом углу — это корреляция `piat_math` с самой собой. Значение в правом нижнем углу — это корреляция `sat_math` с самой собой. Коэффициент корреляции любой переменной с самой собой равен 1, что означает идеальную силу связи.

Значения в правом верхнем углу и левом нижнем углу представляют собой корреляцию `piat_math` с `sat_math` и корреляцию `sat_math` с `piat_math`, которые с неизбежностью равны.

Реализованная в модуле `thinkstats` функция `corrcoef` принимает объект `DataFrame` и имена двух столбцов, выбирает строки, в которых обе переменные имеют действительные значения, и вычисляет их корреляцию:

```
from thinkstats import corrcoef

corrcoef(nlsy, "piat_math", "sat_math")
```

```
0.6397358165178849
```

Воспользуемся этой функцией для вычисления корреляции между `piat_math` и `sat_verbal`:

```
corrcoef(nlsy, "piat_math", "sat_verbal")
```

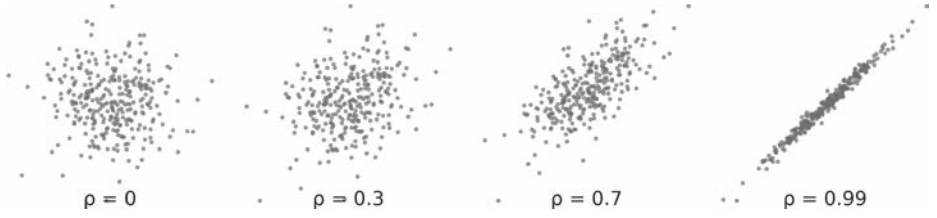
```
0.509413914696731
```

Коэффициент корреляции примерно равен 0.51, поэтому, если чей-то математический балл PIAT на одно стандартное отклонение выше среднего, то стоит ожидать, что его языковой балл SAT среднестатистически будет на 0.51 стандартного отклонения выше среднего.

Как и следовало ожидать, результаты PIAT по математике лучше предсказывают результаты SAT по математике, чем по языковому разделу.

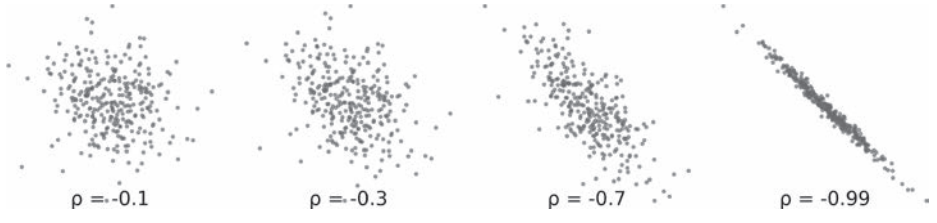
Сила корреляционной связи

Чтобы получить представление о том, как выглядят различные виды корреляции, достаточно посмотреть на другие диаграммы рассеяния. Чтобы вам было проще, на следующем рисунке представлены точечные диаграммы случайно сгенерированных данных с различными значениями коэффициента корреляции:



Греческая буква ρ (произносится как «ро») является условным обозначением коэффициента корреляции.

Корреляция также может быть отрицательной. Ниже приведены диаграммы рассеяния для случайных данных с отрицательным коэффициентом корреляции:



Коэффициент корреляции всегда находится в диапазоне от –1 до 1. Если между двумя переменными нет связи, их корреляция равна 0, но если корреляция равна 0, из этого не следует, что связи нет. В частности, при наличии нелинейной зависимости коэффициент корреляции может быть близок к 0. В каждом из приведенных ниже примеров существует четкая взаимосвязь между переменными — в том смысле, что если задано одно из значений, можно довольно точно предсказать другое. Но в каждом из этих случаев коэффициент корреляции близок к 0:



Корреляция количественно характеризует силу *линейной* зависимости между переменными. Если существует нелинейная зависимость, коэффициент корреляции может вводить в заблуждение. И если корреляция близка к 0, это *не* означает, что взаимосвязи нет.

Ранговая корреляция

Исследование NLSY является лонгитюдным, это означает, что оно отслеживает одну и ту же группу людей в течение длительного времени. Группа, которую мы

изучали, включает людей, родившихся в период с 1980 по 1984 год. Те, кто сдавал SAT, вероятно, сдавали его в конце 1990-х, примерно в 18-летнем возрасте. Поэтому когда в 2021 году их спрашивали об уровне дохода (income), им было под 40 или немного за 40 лет. Присвоим столбцу с данными о доходах более понятное имя:

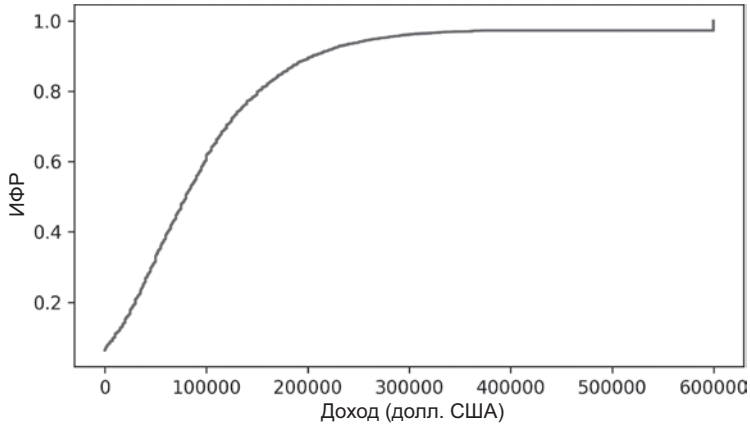
```
nlsy["income"] = nlsy["U4949700"]
nlsy["income"].describe()
```

```
count      6051.000000
mean     104274.239960
std      108470.571497
min           0.000000
25%       38000.000000
50%       80000.000000
75%      134157.000000
max      599728.000000
Name: income, dtype: float64
```

В этом столбце представлен валовой доход семьи (gross family income) — совокупный доход респондента и других членов его домохозяйства из всех источников, выраженный в долларах США. Вот как выглядит распределение доходов:

```
cdf_income = Cdf.from_seq(nlsy["income"])
cdf_income.step()

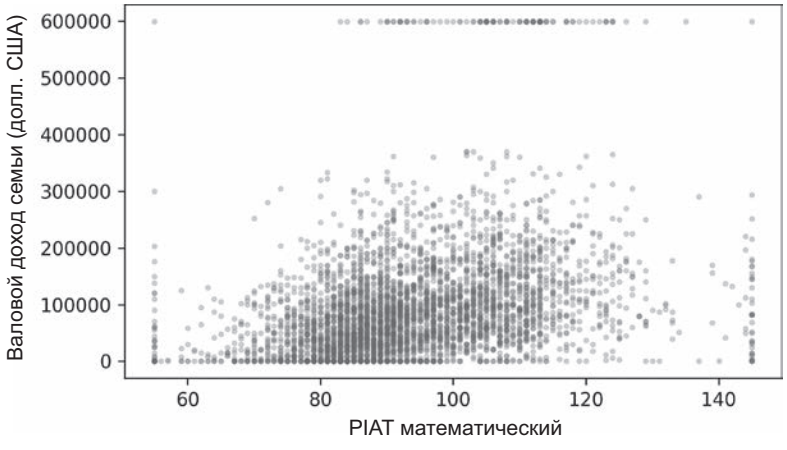
decorate(xlabel="Доход (долл. США)", ylabel="ИФР")
```



Обратите внимание на ступеньку на графике вблизи отметки $600 000 — для защиты анонимности участников суммы, превышающие этот порог, были ограничены. Вот диаграмма рассеяния баллов SAT по математике респондентов и их доходов на более позднем этапе жизни:

```
scatter(nlsy, "piat_math", "income")

decorate(xlabel="PIAT математический", ylabel="Валовой доход семьи (долл. США)")
```



Похоже, что между этими переменными существует некоторая связь. Вот значение коэффициента корреляции:

```
corrcoef(nlsy, "piat_math", "income")
```

```
0.30338587288641233
```

Корреляция составляет около 0.3. Это означает, что если в возрасте 15 лет кто-то получает балл PIAT по математике на одно стандартное отклонение выше среднего, то можно ожидать, что в возрасте 40 лет его доход будет приблизительно на 0.3 стандартного отклонения выше среднего. Связь не такая сильная, как корреляция между баллами PIAT и SAT, но, учитывая количество факторов, влияющих на доход, не стоит ею пренебрегать.

В действительности коэффициент корреляции Пирсона может занижать силу связи. Как вы могли видеть на предыдущей диаграмме рассеяния, обе переменные очевидно имеют избыток точек на краях. Поскольку коэффициент корреляции определяется произведением отклонений от среднего значения, он чувствителен к таким экстремальным значениям.

Более робастным показателем является **ранговая корреляция**, при расчете которой вместо стандартизованных оценок используются ранги оценок. Для вычисления ранга каждого балла и каждого уровня дохода можно применить Pandas-метод `rank`:

```
valid = nlsy.dropna(subset=["piat_math", "income"])

piat_math_rank = valid["piat_math"].rank(method="first")
income_rank = valid["income"].rank(method="first")
```

Если указан аргумент `method="first"`, метод `rank` присваивает ранги от 1 до длины последовательности, составляющей в данном случае 4101:

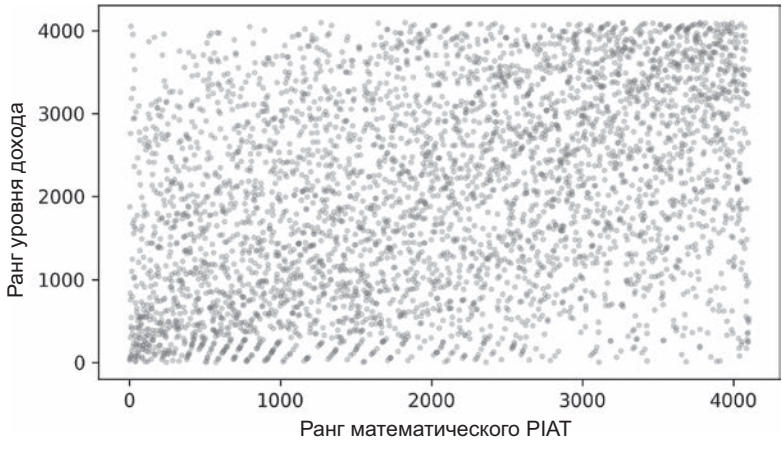
```
income_rank.min(), income_rank.max()
```

```
(1.0, 4101.0)
```

Вот точечная диаграмма зависимости ранга уровня дохода от ранга балла по математике:

```
plt.scatter(piat_math_rank, income_rank, s=5, alpha=0.2)

decorate(xlabel="Ранг математического PIAT", ylabel="Ранг уровня дохода")
```



А вот коэффициент корреляции рангов:

```
np.corrcoef(piat_math_rank, income_rank)[0, 1]
```

```
0.38148396696764847
```

Полученное значение около 0.38 несколько превышает коэффициент корреляции Пирсона, равный 0.30. Поскольку ранговая корреляция менее чувствительна к влиянию экстремальных значений, она, пожалуй, лучше оценивает силу связи между этими переменными.

Функция `rankcorr` из модуля `thinkplot` содержит код из этого раздела:

```
from thinkstats import rankcorr

rankcorr(nlsy, "piat_math", "income")
```

```
0.38474681505344815
```

В качестве упражнения вы можете посчитать коэффициент корреляции между языковыми баллами SAT и доходом — как по формуле Пирсона, так и с использованием рангов.

Корреляция и причинно-следственная связь

Если переменные A и B коррелируют, то наблюдаемая взаимосвязь может быть обусловлена тем, как была составлена случайная выборка, быть следствием нерепрезентативности выборки или указывать на наличие реальной взаимозависимости между количественными величинами в генеральной совокупности.

Если корреляция действительная, этому существует три возможных объяснения: A вызывает B, B вызывает A или какой-то другой набор факторов вызывает как A, так и B. Такие объяснения называют причинно-следственными связями.

Одно лишь наличие корреляции не позволяет установить, какое из этих объяснений в данном случае справедливо. Этот принцип часто резюмируют одной фразой: «Корреляция не подразумевает причинно-следственной связи» (Correlation does not imply causation). Она настолько лаконична и точна, что ей посвящена отдельная страница в английской википедии.

Как же в таком случае доказать наличие причинно-следственной связи?

Используйте фактор времени.

> Если A предшествует B, то A может вызвать B, но не наоборот. Порядок событий может помочь в определении направления причинно-следственной связи, но он не исключает возможности того, что что-нибудь другое вызывает как A, так и B.

Используйте рандомизацию.

> Если большую выборку разделить случайным образом на две группы и вычислить среднее значение большинства переменных в двух группах, то можно предположить, что различие в средних будет мало. Если группы почти идентичны по всем переменным, кроме A и B, то вероятность того, что что-то другое вызывает как A, так и B, можно исключить.

Эти два принципа лежат в основе **рандомизированных контролируемых исследований** (randomized controlled trial). В них испытуемые случайным образом распределяются на две (или более) группы — **экспериментальную** (treatment group), получающую какое-либо вмешательство (например, новое лекарство), и **контрольную**, не получающую никакого вмешательства или подвергающуюся экспериментальному воздействию, эффекты которого известны. Рандомизированное контролируемое исследование — это самый надежный способ доказать причинно-следственную связь и основа доказательной медицины.

К сожалению, контролируемые исследования порой невозможны или неэтичны. Альтернативой им является проведение **естественного эксперимента** (natural experiment). В нем за похожими группами наблюдают в различных условиях, обстоятельства которых не зависят от экспериментатора.

Выявление и измерение причинно-следственных связей — это предмет раздела статистики, называемого **причинно-следственным анализом** (causal inference).

Глоссарий

Диаграмма рассеяния (scatter plot)

График, показывающий взаимосвязь между двумя переменными при помощи точек, каждая из которых представляет собой одно наблюдение в датасете.

Перенасыщенный (overplotted)

Диаграмма рассеяния перенасыщена, если в ней много накладывающихся друг на друга точек, что затрудняет разграничение областей с разной плотностью, а это, в свою очередь, может исказить взаимосвязь.

Джиттеринг (jittering)

Случайный шум, добавляемый на диаграмме к точкам данных и делающий накладывающиеся значения заметнее.

Децильная диаграмма (decile plot)

Диаграмма, отображающая сводную статистику по зависимой переменной для каждого дециля (одной из десяти групп данных) независимой переменной.

Дециль (decile)

Одна из групп, полученная в результате сортировки данных и разбиения их на десять примерно равных частей.

Коэффициент корреляции Пирсона (Pearson correlation coefficient)

Статистический показатель, который измеряет силу и знак (положительный или отрицательный) линейной связи между двумя переменными.

Стандартная оценка (standard score)

Величина, которая, пройдя процедуру стандартизации, выражается в единицах стандартного отклонения от среднего значения.

Корреляционная матрица (correlation matrix)

Таблица, показывающая коэффициенты корреляции для каждой пары переменных в датасете.

Ранговая корреляция (rank correlation)

Робастный способ количественной оценки силы связи при помощи рангов значений вместо самих значений.

Рандомизированное контролируемое исследование (randomized controlled trial)

Эксперимент, в ходе которого испытуемые случайным образом распределяются по группам, получающим различное воздействие.

Экспериментальная группа (treatment group)

Группа, которая в ходе эксперимента получает исследуемое вмешательство.

Контрольная группа (control group)

Группа, в ходе эксперимента не получающая вмешательства или подвергающаяся воздействию, эффект которого известен.

Естественный эксперимент (natural experiment)

Эксперимент, использующий группы, сформированные естественным образом, что иногда может имитировать случайное распределение.

Причинно-следственный анализ (causal inference)

Методы выявления и количественной оценки причинно-следственных связей.

Упражнения

Упражнение 7.1

Функция `decile_plot` из модуля `thinkstats` включает в себя приведенный выше в этой главе код. Вот как с ее помощью можно визуализировать взаимосвязь между баллами по языковому и математическому разделам SAT:

```
from thinkstats import decile_plot

decile_plot(nlsy, "sat_verbal", "sat_math")
decorate(xlabel="SAT языковой", ylabel="SAT математический")
```

Постройте децильную диаграмму зависимости уровня дохода от оценок по математическому тесту PIAT. Похожа ли эта зависимость на линейную?

Упражнение 7.2

Постройте диаграмму рассеяния зависимости дохода от оценок по математике SAT. Вычислите корреляцию Пирсона и ранговую корреляцию. Существенно ли они различаются?

Постройте точечную диаграмму зависимости дохода от баллов языкового SAT и вычислите обе корреляции. Какой из баллов лучше предсказывает будущий доход — математический или языковой?

Упражнение 7.3

Посмотрим, как средний балл (grade point average, GPA) ученика средней школы США соотносится с его баллами SAT. Вот переменная в датасете NLSY, содержащая балл GPA:

```
missing_codes = [-6, -7, -8, -9]
nlsy["gpa"] = nlsy["R9871900"].replace(missing_codes, np.nan) / 100
nlsy["gpa"].describe()
```

```
count    6004.000000
mean        2.818408
std         0.616357
min         0.100000
25%         2.430000
50%         2.860000
75%         3.260000
max         4.170000
Name: gpa, dtype: float64
```

Постройте диаграмму рассеяния, показывающую взаимосвязь между баллом GPA и баллом математического раздела SAT, и вычислите их коэффициент корреляции. Проделайте то же самое для взаимосвязи между баллом GPA и баллом языковой части SAT. Какой из баллов SAT является лучшим предиктором (предсказывающей переменной) балла GPA?

Упражнение 7.4

Исследуем связь между уровнем образования и доходом. В датасете NLSY есть столбец, хранящий наивысший уровень (degree) образования для каждого респондента. Значения закодированы целыми числами:

```
nlsy["degree"] = nlsy["Z9083900"]
nlsy["degree"].value_counts().sort_index()
```

```
degree
0.0     877
1.0    1167
2.0    3531
3.0     766
4.0    1713
5.0     704
6.0      64
7.0     130
Name: count, dtype: int64
```

Для их расшифровки можно воспользоваться следующими списками[1]:

[1] Соответствие российским уровням образования не вполне точное (в частности, Associate's degree — степень, присваиваемая после двух лет обучения в колледже; PhD — это постдипломное образование, но не совсем «кандидат наук»), но в данном случае это не имеет значения для понимания задачи. — *Примеч. ред.*

```
positions = [0, 1, 2, 3, 4, 5, 6, 7]
labels = [
    "None (Нет)",
    "GED (Неполное среднее образование)",
    "High school diploma (Полное среднее образование)",
    "Associate's degree (Среднее профессиональное образование)",
    "Bachelor's degree (Бакалавр)",
    "Master's degree (Магистр)",
    "PhD (Кандидат наук)",
    "Professional degree (Послевузовское профессиональное образование)",
]
```

Постройте диаграмму рассеяния зависимости уровня дохода (`income`) от уровня образования (`degree`). Чтобы избежать перенасыщения, примените джиттеринг к переменной `degree`, а также подберите размер точек и их прозрачность.

При помощи метода `groupby` сгруппируйте респондентов по значениям переменной `degree`. У объекта класса `DataFrameGroupBy` выберите столбец `income`; затем, пользуясь методом `quantile`, для каждой группы рассчитайте медиану, 10-й и 90-й процентили. Используйте Matplotlib-функцию `fill_between` для заполнения области между 10-м и 90-м процентилями и функцию `plot` для построения графика медианы.

Что можно сказать о прибавке дохода, связанной с каждой последующей ступенью образования?

Упражнение 7.5

В датасете Системы наблюдения за поведенческими факторами риска США (Behavioral Risk Factor Surveillance System, BRFSS) содержатся данные о росте и весе, записанные со слов около 400 000 респондентов. Инструкции по загрузке данных приведены в Jupyter-блокноте к этой главе.

Постройте диаграмму рассеяния, демонстрирующую связь между ростом и весом. Возможно, придется использовать джиттеринг для размытия эффекта строк и столбцов, возникающего из-за округления данных. При такой большой выборке, чтобы избежать перенасыщения, также потребуется подобрать размер и прозрачность точки. Кроме того, поскольку в обеих переменных есть выбросы, то, чтобы сфокусироваться на области значений, охватывающей основную часть респондентов, пригодятся аргументы `xlim` и `ylim`.

Постройте децильную диаграмму связи роста и веса. Похожа ли эта зависимость на линейную? Вычислите коэффициент корреляции и ранговую корреляцию. Существенно ли их отличие? Какой из них, по вашему мнению, лучше измеряет взаимосвязь между этими переменными?

ГЛАВА 8

Оценка

Предположим, что вы живете в городе с населением 10 000 человек и хотите предсказать, кто победит на предстоящих выборах. Теоретически можно спросить всех жителей города, за кого они собираются голосовать. Если бы все из опрошенных ответили честно, то можно было бы составить достоверный прогноз.

Но даже в небольшом городе, пожалуй, затруднительно провести опрос всей популяции. К счастью, в этом нет необходимости. Если опросить случайное подмножество жителей, то можно использовать эту выборку для определения избирательных предпочтений всей популяции. Такой процесс использования выборки для получения выводов о генеральной совокупности называется статистическим выводом (statistical inference).

Статистический вывод включает в себя оценку (estimation) — тема этой главы, а также проверку гипотез — предмет обсуждения следующей главы.

Вес пингвинов

Представьте, что вы — исследователь в Антарктиде, изучающий местные популяции пингвинов. Одна из ваших задач — следить за средним весом пингвинов, который меняется в течение года. Было бы нецелесообразно взвешивать каждого пингвина, обитающего на территории, поэтому вы планируете каждую неделю случайным образом выбирать 10 пингвинов, взвешивать их и использовать эту выборку для оценки среднего значения по всей популяции, называемого **средним значением генеральной совокупности** (population mean).

Существуют разные способы использования выборки для оценки среднего значения генеральной совокупности, но мы рассмотрим только два из них: выборочное среднее значение (sample mean) и выборочную медиану (sample median). Оба варианта приемлемы, но попробуем определить, какой из них лучше, а также подумаем, что можно называть «лучшим».

Для примера предположим, что вес пингвинов описывается нормальным распределением с известными средним значением и стандартным отклонением. Обозначим их как `mu` и `sigma` и присвоим им значения в килограммах:

```
mu = 3.7
sigma = 0.46
```

Эти величины являются **параметрами** нормального распределения, что означает, что они точно определяют конкретное распределение. При заданных параметрах

мы можем использовать библиотеку NumPy для моделирования процесса выборки и генерации выборки (sample) любого размера. Например, вот гипотетическая выборка из 10 значений веса:

```
sample = np.random.normal(mu, sigma, size=10)
sample
```

```
array([4.44719887, 3.41859205, 3.45704099, 3.20643443, 4.09808751,
       2.6412922 , 4.50261341, 3.34984483, 3.84675798, 3.58528963])
```

А вот выборочные среднее значение и медиана:

```
np.mean(sample), np.median(sample)
```

```
(3.6553151902291945, 3.521165310619601)
```

Среднее значение и медиана достаточно сильно отличаются друг от друга, поэтому стоит задаться вопросом, какая оценка лучше. Чтобы выяснить это, воспользуемся следующей функцией для генерации гипотетических выборок заданного размера `n`:

```
def make_sample(n):
    return np.random.normal(mu, sigma, size=n)
```

Проведем первый эксперимент и посмотрим, как ведут себя выборочное среднее значение и выборочная медиана по мере увеличения размера выборки. Воспользуемся NumPy-функцией `logspace`, чтобы создать массив равномерно распределенных значений на логарифмической шкале в диапазоне от 10 до 100 000 и присвоить его переменной `ns`:

```
ns = np.logspace(1, 5).astype(int)
```

Чтобы сгенерировать возможную выборку для каждого значения `n`, вычислить среднее значение и сохранить результаты, воспользуемся списковым включением (list comprehension):

```
means = [np.mean(make_sample(n)) for n in ns]
```

Поступим так же с медианой:

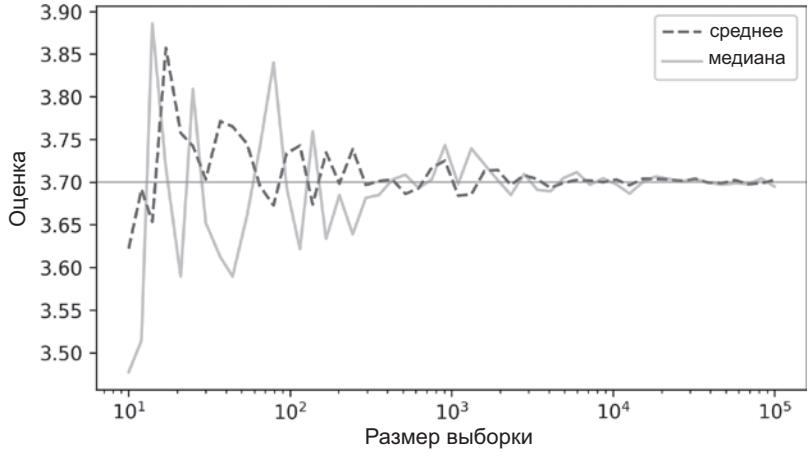
```
medians = [np.median(make_sample(n)) for n in ns]
```

Статистический показатель, такой как выборочное среднее значение или медиана, используемый для оценки какого-либо свойства генеральной совокупности, называется **оценкой** (estimator).

Следующий рисунок демонстрирует, как ведут себя эти оценки при увеличении размера выборки. Горизонтальная линия обозначает фактическое среднее значение по популяции:

```
plt.axhline(mu, color="gray", lw=1, alpha=0.5)
plt.plot(ns, means, "--", label="среднее")
plt.plot(ns, medians, alpha=0.5, label="медиана")

decorate(xlabel="Размер выборки", xscale="log", ylabel="Оценка")
```



Обе оценки (estimate) с увеличением объема выборки приближаются к фактическому значению. Это говорит о том, что они **состоятельные** (consistent) — одно из свойств, которым должна обладать хорошая статистическая оценка. Если руководствоваться этим критерием, то среднее значение и медиана кажутся одинаково хорошими.

Глядя на рисунок выше, можно заметить, что оценка временами чрезмерно завышена, а порой слишком занижена, и похоже, что эти отклонения примерно симметричны относительно истинного значения. Это наводит на мысль о следующем эксперименте: что, если собрать много выборок одинакового размера и посчитать оценку для каждой? Каково будет тогда среднее значение этих оценок?

Цикл ниже воспроизводит этот сценарий, генерируя 10 001 выборку из 10 пингвинов и вычисляя среднее значение для каждой выборки:

```
means = [np.mean(make_sample(n=10)) for i in range(10001)]
np.mean(means)
```

```
3.70034508492869
```

Среднее значение средних близко к фактическому среднему значению, которое мы использовали для генерации выборок, — 3.7 кг.

Следующий цикл имитирует тот же сценарий, но на этот раз он вычисляет медиану для каждой выборки:

```
medians = [np.median(make_sample(n=10)) for i in range(10001)]
np.mean(medians)
```

```
3.701214089907223
```

Среднее значение этих гипотетических медиан также очень близко к фактическому среднему значению генеральной совокупности.

Эти результаты показывают, что выборочное среднее и выборочная медиана являются **несмещенными** (unbiased) оценками, то есть в среднем они верны. Слово «смещенный» в разных ситуациях означает разное, что может приводить к путанице. В описанном контексте «несмещенный» подразумевает, что среднее значение оценок является истинным значением.

Пока что мы убедились, что обе оценки являются состоятельными и несмещенными, но до сих пор неясно, какая из них лучше. Проведем еще один эксперимент: посмотрим, какая из оценок более верная. Смысл слова «верный» (accurate) также зависит от контекста. В качестве одного из способов перевода его на язык цифр рассмотрим показатель **средний квадрат ошибки** (mean squared error, MSE). Функция ниже вычисляет разницу между оценками (estimates) и истинным значением (actual), возвращая среднее значение квадратов этих ошибок (errors):

```
def mse(estimates, actual):
    """Средний квадрат ошибки последовательности оценок."""
    errors = np.asarray(estimates) - actual
    return np.mean(errors**2)
```

Обратите внимание, что величину MSE можно вычислить только в том случае, если известно фактическое значение. На деле оно известно редко: ведь если бы мы знали истинное значение, нам не пришлось бы его оценивать. Но в нашем эксперименте мы знаем, что истинное среднее значение генеральной совокупности составляет 3.7 кг, поэтому можем использовать его для вычисления MSE выборочного среднего:

```
mse(means, mu)
```

```
0.020871984891289382
```

При наличии выборок размера 10 и использовании выборочного среднего для оценки среднего значения по популяции средний квадрат ошибки составляет около 0.021 килограмма в квадрате. Теперь посчитаем MSE выборочных медиан:

```
mse(medians, mu)
```

```
0.029022273128644173
```

Если использовать выборочную медиану для оценки среднего значения совокупности, то средний квадрат ошибки составит около 0.029 килограмма в квадрате. В этом примере выборочное среднее значение оказывается лучше, чем выборочная

медиана; и в целом, если данные получены на основе нормального распределения, то это *наилучшая* несмещенная оценка среднего значения генеральной совокупности, поскольку она минимизирует величину MSE.

Минимизация MSE — полезное свойство оценки, но MSE не всегда является лучшей характеристикой ошибок. Прежде всего ее трудно интерпретировать. В нашем примере единицами измерения MSE являются килограммы в квадрате, поэтому трудно понять, что эта величина означает.

Одно из решений — использовать квадратный корень из MSE, то есть корень из среднего квадрата ошибки (root mean squared error, RMSE). Другой вариант — использовать среднее значение абсолютных значений ошибок, называемое «средней абсолютной ошибкой» (mean absolute error, MAE). Следующая функция вычисляет MAE для последовательности оценок:

```
def mae(estimates, actual):
    """Средняя абсолютная ошибка последовательности оценок."""
    errors = np.asarray(estimates) - actual
    return np.mean(np.abs(errors))
```

Вот значение MAE для выборочных средних:

```
mae(means, mu)
```

```
0.11540433749505272
```

И для выборочных медиан:

```
mae(medians, mu)
```

```
0.13654429774596036
```

Можно ожидать в среднем, что выборочное среднее отклонится от истинного среднего примерно на 0.115 кг, а выборочная медиана — на 0.137 кг. Таким образом, выборочное среднее, вероятно, является более удачным методом оценки — по крайней мере, для данного примера.

Робастность

Теперь рассмотрим другой сценарий. Предположим, что, когда вы пытаетесь взвесить пингвина, в 2 % случаев он случайно нажимает кнопку единиц измерения на весах, и вес записывается в фунтах вместо килограммов. Если ошибка остается незамеченной, она приводит к появлению в выборке выбросов.

Функция ниже моделирует этот сценарий, умножая 2 % показаний веса на переводной коэффициент, равный 2.2 фунта на килограмм:

```
def make_sample_with_errors(n):
    sample = np.random.normal(mu, sigma, size=n)
```

```
    factor = np.random.choice([1, 2.2], p=[0.98, 0.02], size=n)
    return sample * factor
```

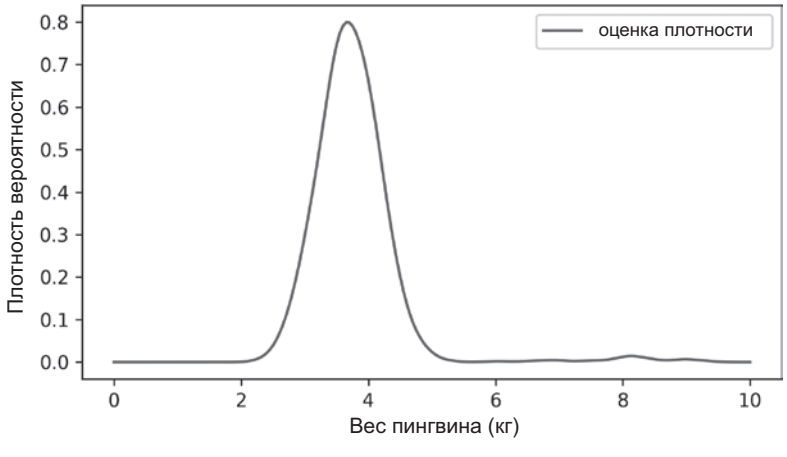
Чтобы проверить, как это влияет на распределение, сгенерируем бóльшую выборку:

```
sample = make_sample_with_errors(n=1000)
```

Чтобы построить график распределения выборки, используем ядерную оценку плотности и `Pdf`-объект из раздела «Ядерная оценка плотности» главы 6 на с. 111:

```
from scipy.stats import gaussian_kde
from thinkstats import Pdf

kde = gaussian_kde(sample)
domain = 0, 10
pdf = Pdf(kde, domain)
pdf.plot(label='оценка плотности')
decorate(xlabel="Вес пингвина (кг)", ylabel="Плотность вероятности")
```



В дополнение к моде в районе 3.7 кг, ошибки измерения приводят к появлению второй моды около 8 кг.

Теперь повторим предыдущий эксперимент с моделированием множества выборок размером 10, подсчетом среднего значения для каждой выборки и последующим вычислением среднего значения выборочных средних:

```
means = [np.mean(make_sample_with_errors(n=10)) for i in range(10001)]
np.mean(means)
```

```
3.786352945690677
```

Ошибки измерения приводят к тому, что среднее значение выборочных средних превышает 3.7 кг.

Теперь проведем такой же эксперимент с расчетом выборочных медиан:

```
medians = [np.median(make_sample_with_errors(n=10)) for i in range(10001)]
np.mean(medians)
```

```
3.7121869836715353
```

Среднее значение выборочных медиан также превышает 3.7 кг, но отклонение не так значительно. Если сравнить MSE этих двух оценок, то можно убедиться, что выборочная медиана значительно вернее:

```
mse(means, mu), mse(medians, mu)
```

```
(0.06853430354724438, 0.031164467796883758)
```

Если измерения действительно описываются нормальным распределением, то выборочное среднее значение минимизирует MSE. Но в данном примере это предположение нарушено, поэтому выборочное среднее не минимизирует MSE. Выборочная медиана менее чувствительна к выбросам, поэтому она является менее смещенной, к тому же ее MSE меньше. Оценки, которые хорошо работают с выбросами и другими похожими нарушениями теоретических допущений, называют **робастными** (robust) или устойчивыми.

Оценка дисперсии

Рассмотрим еще один пример. Представьте, что вам надо оценить дисперсию веса пингвинов. В разделе «Сводные статистические показатели» главы 1 на с. 28 вы познакомились с двумя способами вычисления дисперсии выборки. Тогда я пообещал объяснить разницу между ними позже. Это время пришло.

Причина, по которой существует два разных способа вычисления дисперсии выборки, заключается в том, что один из них является смещенной оценкой дисперсии генеральной совокупности, а другой — несмещенной. Следующая функция вычисляет смещенную (biased) оценку, которая равна сумме квадратов отклонений (deviations), деленной на n:

```
def biased_var(xs):
    # Вычислить дисперсию с n в знаменателе.
    n = len(xs)
    deviations = xs - np.mean(xs)
    return np.sum(deviations**2) / n
```

Чтобы ее протестировать, сгенерируем множество выборок размером 10, вычислим смещенную оценку дисперсии каждой выборки, а затем рассчитаем среднее значение всех оценок:

```
biased_vars = [biased_var(make_sample(n=10)) for i in range(10001)]
np.mean(biased_vars)
```

```
0.19049277659404473
```

Результат составляет около 0.19, но в данном случае нам известно, что фактическая дисперсия генеральной совокупности примерно равна 0.21, так что эта версия выборочной дисперсии в среднем дает заниженное значение, что подтверждает ее смещенность.

Следующая функция вычисляет несмещенную (unbiased) оценку, которая равна сумме квадратов отклонений, деленной на `n-1`:

```
def unbiased_var(xs):
    # Вычислить дисперсию с n-1 в знаменателе.
    n = len(xs)
    deviations = xs - np.mean(xs)
    return np.sum(deviations**2) / (n - 1)
```

Проверим ее, сгенерировав множество выборок и вычислив несмещенную оценку дисперсии для каждой из них:

```
unbiased_vars = [unbiased_var(make_sample(n=10)) for i in range(10001)]
np.mean(unbiased_vars)
```

```
0.21159109492300626
```

Среднее значение несмещенных оценок выборочных дисперсий очень близко к фактическому значению — это именно то, что ожидается от несмещенной оценки.

При размере выборки 10 разница между смещенной и несмещенной оценками составляет около 10 %, что нельзя считать пренебрежимо малой величиной. При размере выборки 100 разница составляет всего 1 %, что достаточно мало, чтобы ею можно было пренебречь на практике.

Выборочное распределение

До сих пор мы работали с данными моделирования, предполагая, что значения веса пингвинов описываются нормальным распределением с известными параметрами. Теперь посмотрим, что получится, если воспользоваться реальными данными.

В промежутке с 2007 по 2010 год исследователи на антарктической станции Палмер измерили и взвесили 342 пингвина из местных популяций. Собранные ими данные общедоступны — инструкции по их загрузке приведены в Jupyter-блокноте этой главы.

Используем библиотеку Pandas для считывания данных:

```
penguins = pd.read_csv("penguins_raw.csv").dropna(subset=["Body Mass (g)"])
penguins.shape
```

```
(342, 17)
```

Датасет содержит данные для трех видов (species) пингвинов:

```
penguins["Species"].value_counts()
```

```
Species
Adelie Penguin (Pygoscelis adeliae)          151
Gentoo penguin (Pygoscelis papua)            123
Chinstrap penguin (Pygoscelis antarctica)     68
Name: count, dtype: int64
```

Для первого примера выберем только пингвинов вида антарктический пингвин (Chinstrap penguin):

```
chinstrap = penguins.query('Species.str.startswith("Chinstrap")')
```

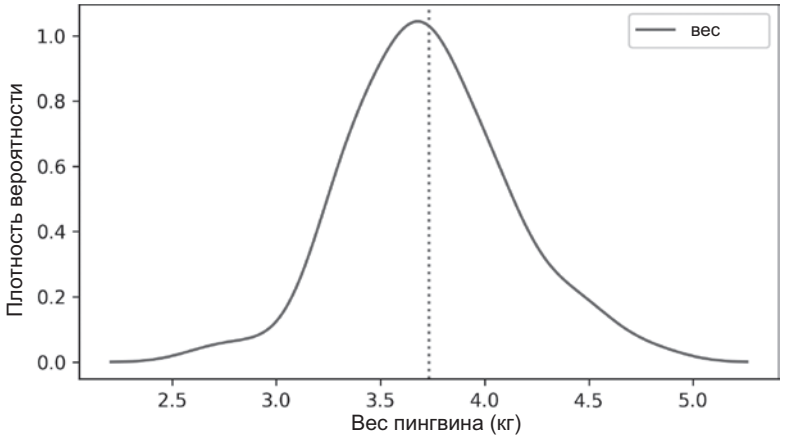
Используем эту функцию для построения графика оценки функции плотности вероятности (ФПВ):

```
def plot_kde(sample, name="оценка плотности", **options):
    kde = gaussian_kde(sample)
    m, s = np.mean(sample), np.std(sample)
    plt.axvline(m, color="gray", ls=":")

    domain = m - 4 * s, m + 4 * s
    pdf = Pdf(kde, domain, name)
    pdf.plot(**options)
```

Вот распределение веса антарктического пингвина в килограммах. Вертикальной пунктирной линией обозначено выборочное среднее значение:

```
weights = chinstrap["Body Mass (g)"] / 1000
plot_kde(weights, "weights")
decorate(xlabel="Вес пингвина (кг)", ylabel="Плотность вероятности")
```



Выборочное среднее веса составляет около 3.7 кг:

```
sample_mean = np.mean(weights)
sample_mean
```

```
3.733088235294118
```

Оценка 3.7 кг выглядит разумной для среднего значения по популяции, но насколько она точна (precise)?

Один из способов ответить на этот вопрос — вычислить **выборочное распределение** среднего значения, которое показывает, насколько существенно оценка среднего изменяется от выборки к выборке. Если бы мы знали фактическое среднее значение и стандартное отклонение генеральной совокупности, то могли бы смоделировать процесс выборки и рассчитать выборочное распределение. Но если бы мы знали фактическое среднее значение по популяции, нам не пришлось бы его оценивать!

К счастью, существует простой способ аппроксимировать выборочное распределение, называемый **повторной выборкой** или ресемплингом (resampling). Его основная идея в том, чтобы использовать выборку для создания модели генеральной совокупности, а затем использовать эту модель для воспроизведения процесса выборки.

Точнее говоря, нам поможет **параметрический ресемплинг** (parametric resampling), когда сначала с помощью выборки оцениваются параметры генеральной совокупности, а затем используется теоретическое распределение для генерации новых выборок.

Следующая функция реализует этот алгоритм для нормального распределения. Обратите внимание, что новая выборка имеет тот же размер, что и исходная:

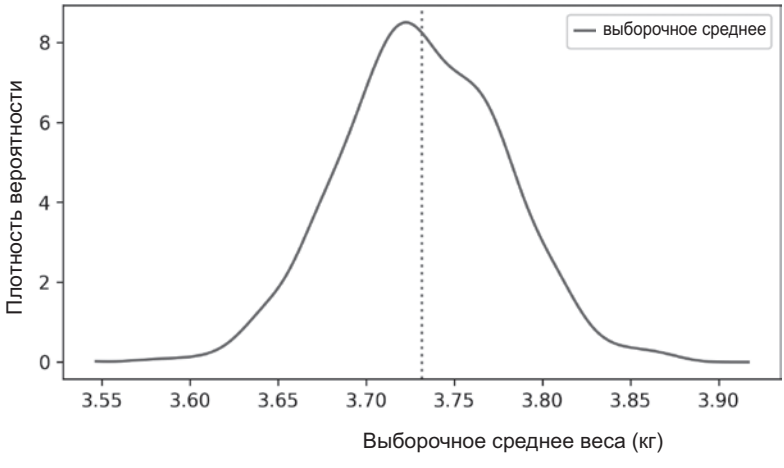
```
def resample(sample):
    m, s = np.mean(sample), np.std(sample)
    return np.random.normal(m, s, len(sample))
```

В цикле ниже функция `resample` используется для генерации совокупности выборок и вычисления среднего значения для каждой из них:

```
sample_means = [np.mean(resample(weights)) for i in range(1001)]
```

На рисунке ниже показано распределение этих выборочных средних:

```
plot_kde(sample_means, "выборочное среднее")
decorate(xlabel="Выборочное среднее веса (кг)", ylabel="Плотность вероятности")
```



Мы получаем аппроксимацию выборочного распределения среднего по выборке. Она показывает, насколько сильно может меняться выборочное среднее значение, если собрать множество выборок одинакового размера — при условии, что наша модель генеральной совокупности верна.

Проще говоря, выборочное среднее для выборок одинакового размера может принимать значение как 3.55, так и 3.9.

Стандартная ошибка

Чтобы количественно оценить ширину выборочного распределения, можно вычислить его стандартное отклонение. Полученную величину называют **стандартной ошибкой** (standard error):

```
standard_error = np.std(sample_means)
standard_error
```

```
0.04626531069684985
```

В данном случае стандартная ошибка составляет около 0.045 кг, поэтому если мы соберем множество выборок, то ожидается, что выборочное среднее значение будет варьироваться в среднем примерно на 0.045 кг.

Часто путают стандартную ошибку и стандартное отклонение. Запомните:

- стандартное отклонение количественно выражает разброс в измерениях;
- стандартная ошибка количественно характеризует точность оценки.

В этом датасете стандартное отклонение веса пингвинов вида антарктический пингвин равно примерно 0.38 кг:

```
np.std(weights)
```

```
0.3814986213564681
```

Стандартная ошибка среднего веса составляет около 0.046 кг:

```
np.std(sample_means)
```

```
0.04626531069684985
```

Стандартное отклонение показывает, насколько пингвины различаются по весу. Стандартная ошибка показывает, насколько точна оценка. Они отвечают на разные вопросы.

Однако между ними существует взаимосвязь. Если мы знаем стандартное отклонение и размер выборки, то можем приблизительно рассчитать стандартную ошибку среднего значения следующим образом:

```
def approximate_standard_error(sample):
    n = len(sample)
    return np.std(sample) / np.sqrt(n)
```

```
approximate_standard_error(weights)
```

```
0.046263503290595163
```

Это значение близко к тому, что мы получили при помощи ресемплинга.

Доверительный интервал

Другим способом охарактеризовать выборочное распределение является вычисление **доверительного интервала** (confidence interval). Например, 90%-ный доверительный интервал содержит 90 % значений в выборочном распределении. Его мы можем найти, вычислив 5-й и 95-й процентили. Вот 90%-ный доверительный интервал для среднего веса антарктического пингвина:

```
ci90 = np.percentile(sample_means, [5, 95])
ci90
```

```
array([3.6576334 , 3.80737506])
```

При интерпретации доверительного интервала возникает соблазн сказать, что существует 90%-ная вероятность того, что истинное значение данного параметра генеральной совокупности попадает в 90%-ный доверительный интервал. В данном случае это означало бы заявить, что с вероятностью 90 % среднее значение веса по популяции антарктического пингвина попадает в интервал от 3.66 до 3.81 кг.

Согласно строгой концепции вероятности, называемой **частотным подходом**[1] (frequentism), такая интерпретация была бы недопустима. К тому же во многих книгах по статистике утверждается, что она ошибочна.

На мой взгляд, такое ограничение чрезмерно строго. При разумных взглядах на вероятность доверительный интервал означает именно то, что люди ожидают от него: существует 90%-ная вероятность того, что истинное значение попадает в 90%-ный доверительный интервал.

Однако доверительный интервал позволяет оценить только вариативность, обусловленную выборкой, то есть характеризует только часть генеральной совокупности. Выборочное распределение оставляет за скобками другие источники ошибок, в частности смещение выборки (sampling bias) и погрешность измерений, которые рассматриваются в следующем разделе.

Источники ошибок

Представьте теперь, что вместо среднего веса пингвинов Антарктиды вы хотите узнать средний вес женщин вашего города. Вы не можете случайным образом составить репрезентативную выборку женщин и взвесить их всех.

Как альтернативный вариант можно попробовать телефонную выборку (telephone sampling): в телефонной книге можно произвольным образом выбрать номер, позвонить по нему и попросить к телефону взрослую женщину, а дальше поинтересоваться ее весом. Но такая телефонная выборка имеет очевидные недостатки.

Например, выборка ограничена людьми, чьи телефонные номера указаны в телефонном справочнике. Поэтому в ней заведомо отсутствуют люди без телефона (вероятно, с достатком ниже среднего) и абоненты, чьи номера не указаны (возможно, с достатком выше среднего). Кроме того, если звонить на домашние телефоны в дневное время, то в выборку с меньшей вероятностью попадут работающие люди. А если отбирать только тех, кто отвечает на телефонный звонок,

[1] Одно из двух основных направлений в статистике (наряду с байесовским подходом), которое определяет вероятность как предел относительной частоты события при бесконечно большом числе повторений эксперимента. — *Примеч. пер.*

то в выборку с меньшей вероятностью попадут абоненты, пользующиеся одной телефонной линией.

Если такие факторы, как доход, занятость и размер домохозяйства, связаны с весом, что вполне вероятно, то это так или иначе повлияет на результаты вашего опроса. Подобное затруднение называют **смещенностью выборки** (sampling bias), поскольку оно является особенностью процесса составления выборки.

Также на процесс выборки может повлиять самоотбор (self-selection) респондентов, что тоже является своего рода смещенностью выборки. Некоторые люди отказываются отвечать на подобные вопросы, и если вероятность отказа связана со значением веса, то это может сказаться на результатах.

Наконец, если спрашивать людей про их вес вместо того, чтобы их взвешивать, результаты могут оказаться неверными. Даже отзывчивые респонденты могут округлять свой реальный вес в бóльшую или меньшую сторону, если он у них вызывает неловкость. К тому же не все респонденты оказываются готовы помочь. Такие погрешности являются примерами **ошибки измерений** (measurement error).

Когда вы приводите оценку некоторой величины, полезно также описать меру изменчивости, обусловленную выборкой, указав стандартную ошибку или доверительный интервал. Но помните, что эта вариативность является только одним из источников ошибок, и зачастую не самым главным.

Глоссарий

Среднее значение популяции (population mean)

Истинное среднее значение некоторой величины во всей популяции, в отличие от среднего значения выборки, вычисляемого на некотором ее подмножестве.

Параметр (parameter)

Одно из значений, выделяющих конкретное распределение из некоторого множества распределений: например, параметрами нормального распределения являются математическое ожидание (среднее) и стандартное отклонение.

Оценка (estimator)

Статистический показатель, рассчитываемый по выборке и используемый для оценки некоторого параметра популяции.

Состоятельный (consistent)

Оценка является состоятельной, если она стремится к фактическому значению параметра по мере увеличения размера выборки.

Несмещенный (unbiased)

Оценка является несмещенной, если для конкретного размера выборки среднее значение выборочных оценок равно фактическому значению параметра.

Средний квадрат ошибки (mean squared error, MSE)

Мера верности оценки, равная среднему квадрату разности между расчетным и истинным значениями параметра, при условии, что истинное значение известно.

Робастный (robust)

Оценка является робастной (устойчивой), если она остается верной даже в том случае, когда набор данных содержит выбросы или ошибки либо не полностью соответствует теоретическому распределению.

Ресемплинг, или повторная выборка (resampling)

Способ аппроксимации выборочного распределения оценки путем моделирования процесса выборки.

Выборочное распределение (sampling distribution)

Распределение статистического показателя по возможным выборкам из одной и той же популяции.

Параметрический ресемплинг (parametric resampling)

Вид ресемплинга (повторной выборки), в котором сначала на основе данных выборки оцениваются параметры популяции, а затем для моделирования процесса выборки используется теоретическое распределение.

Стандартная ошибка (standard error)

Стандартное отклонение выборочного распределения, которое количественно характеризует изменчивость оценки из-за случайного отбора (но не из-за ошибки измерения или нерепрезентативности выборки).

Доверительный интервал (confidence interval)

Интервал, содержащий наиболее вероятные значения в выборочном распределении.

Смещенность выборки (sampling bias)

Недостаток в способе отбора выборки, который делает ее нерепрезентативной в отношении популяции.

Ошибка измерения (measurement error)

Погрешность процесса наблюдения, измерения или регистрации данных.

Упражнения

Упражнение 8.1

Одно из преимуществ метода ресемплинга заключается в том, что его легко распространить на другие статистические показатели. В этой главе мы вычислили

выборочное среднее веса пингвинов, а затем использовали повторную выборку для аппроксимации выборочного распределения среднего. Теперь проделаем то же самое для стандартного отклонения.

Вычислите выборочное стандартное отклонение веса пингвинов вида антарктический пингвин (Chinstrap). Затем используйте функцию `resample` для аппроксимации выборочного распределения стандартного отклонения. Задействуйте выборочное распределение для вычисления стандартной ошибки оценки и 90%-ного доверительного интервала.

Упражнение 8.2

Датасет Системы наблюдения за поведенческими факторами риска (Behavioral Risk Factor Surveillance System, BRFSS) США содержит данные о росте и весе, записанные со слов респондентов, для некоторой выборки взрослых жителей Соединенных Штатов. Используйте эти данные для оценки среднего роста взрослых мужчин. Воспользуйтесь ресемплингом для аппроксимации выборочного распределения и вычисления 90%-ного доверительного интервала.

Поскольку объем выборки очень большой, доверительный интервал совсем мал. Это означает, что изменчивость, обусловленная случайной выборкой, невелика. Но вклад других факторов может быть больше. Какие еще источники ошибок, по вашему мнению, влияют на результаты?

Упражнение 8.3

В таких играх, как футбол и хоккей, время между забитыми мячами обычно описывается экспоненциальным распределением (как вы узнали из раздела «Экспоненциальная функция плотности вероятности» главы 6 на с. 107). Предположим, у нас имеется выборка со значениями промежутков времени между голами. Если предположить, что выборка получена из экспоненциального распределения, как оценить фактическое среднее значение распределения? Потенциально можно использовать либо выборочное среднее, либо выборочную медиану. Посмотрим, является ли какая-либо из этих оценок состоятельной и несмещенной. В наших опытах будем считать, что фактическое среднее (`actual_mean`) время между забитыми мячами составляет 10 минут:

```
actual_mean = 10
```

Следующая функция генерирует выборку из экспоненциального распределения с указанным средним значением и заданного объема:

```
def make_exponential(n):
    return np.random.exponential(actual_mean, size=n)
```

Используйте эту функцию для генерации выборок различного объема и вычисления среднего значения для каждой из них. При увеличении `n` стремятся ли выборочные средние значения к фактическому среднему?

Затем снова сгенерируйте выборки разного объема и вычислите медиану для каждой из них. Сходятся ли выборочные медианы к фактическому значению медианы?

Далее сгенерируйте множество выборок размером 10 и проверьте, является ли выборочное среднее значение несмещенной оценкой среднего значения популяции.

Наконец, проверьте, является ли выборочная медиана несмещенной оценкой медианы популяции.

Упражнение 8.4

В этой главе мы проверили смещенную оценку дисперсии и убедились, что она действительно смещенная. Мы также показали, что несмещенная оценка является несмещенной. Теперь поработаем со стандартным отклонением.

Чтобы оценить стандартное отклонение популяции, вычислим квадратный корень из смещенной (biased_std) или несмещенной оценки (unbiased_std) дисперсии следующим образом:

```
def biased_std(sample):
    # Квадратный корень из смещенной оценки дисперсии
    var = biased_var(sample)
    return np.sqrt(var)
```

```
def unbiased_std(sample):
    # Квадратный корень из несмещенной оценки дисперсии
    var = unbiased_var(sample)
    return np.sqrt(var)
```

Используйте функцию `make_sample`, чтобы сгенерировать множество выборок размером 10 из нормального распределения со средним значением 3.7 и стандартным отклонением 0.46. Проверьте, является ли какая-либо из этих двух оценок несмещенной оценкой стандартного отклонения.

Упражнение 8.5

Идея этого упражнения заимствована из задачи о немецких танках (German tank problem), которая представляет собой упрощенную версию реального анализа, проведенного отделом экономической войны (Economic Warfare Division) американского посольства в Лондоне во время Второй мировой войны.

Представьте, что вы разведчик союзников и ваша задача — оценить, сколько танков построили немцы. В качестве данных у вас есть серийные номера, снятые с `k` трофейных танков.

Если предположить, что у немцев произведено N танков с номерами от 1 до N и что все эти танки могут быть захвачены с равной вероятностью, то N можно оценить следующим образом:

```
def estimate_tanks(sample):
    m = np.max(sample)
    k = len(sample)
    return m + (m - k) / k
```

В качестве примера предположим, что N равно 122:

```
N = 122
tanks = np.arange(1, N + 1)
```

Для генерации случайной выборки из k танков можно использовать следующую функцию:

```
def sample_tanks(k):
    return np.random.choice(tanks, replace=False, size=k)
```

Вот пример выборки:

```
sample = sample_tanks(5)
sample
```

```
array([74, 71, 95, 10, 17])
```

И вот оценка на ее основе:

```
estimate_tanks(sample)
```

```
113.0
```

Проверьте, является ли эта оценка смещенной.

Упражнение 8.6

В некоторых видах спорта, особенно в баскетболе, многие игроки и болельщики верят в феномен «горячей руки» (hot hand), который подразумевает, что игрок, поразивший цель несколько раз подряд, с большей вероятностью попадет и в следующий раз, а игрок, промахнувшийся несколько раз, скорее всего, промахнется.

В одной известной статье был предложен способ проверить, реален ли данный феномен или он является заблуждением, при помощи анализа последовательности попаданий и промахов в профессиональных баскетбольных матчах[1]. Для

[1] Гилович Т. (T. Gilovich), Валлоне Р. (R. Vallone) и Тверски А. (A. Tversky). «The Hot Hand in Basketball: On the Misperception of Random Sequences», Cognitive Psychology, 17 (3): 295–314.

каждого игрока авторы рассчитали общую вероятность попадания и условную вероятность попадания после трех последовательных попаданий. Как они выяснили, у восьми из девяти игроков вероятность попадания после трех попаданий подряд была *ниже*. Отталкиваясь от этого и других результатов, они пришли к выводу, что «нет доказательств положительной корреляции между результатами последовательных попыток». И несколько десятилетий многие считали, что миф о «горячей руке» был развенчан.

Однако этот вывод был основан, по крайней мере частично, на статистической ошибке. В статье, опубликованной в 2018 году, было показано, что статистический показатель, использованный в первой статье, — вероятность попадания после трех попаданий — является смещенным[1]. Даже если вероятность попадания в цель при каждом ударе равна 0.5 и между результатами в действительности нет корреляции, вероятность поразить цель после трех попаданий составляет *менее 0.5*.

Справедливость этого утверждения неочевидна — именно поэтому эта ошибка так долго оставалась незамеченной, и я не буду пытаться ее здесь объяснить. Но мы можем задействовать методы этой главы для проверки данной гипотезы. Воспользуемся следующей функцией для создания последовательности из нулей и единиц с вероятностью 0.5 и в отсутствие корреляции:

```
def make_hits_and_misses(n):
    # Генерировать случайную последовательность из 0 и 1
    return np.random.choice([0, 1], size=n)
```

В блокноте этой главы я привожу функцию, которая находит все подпоследовательности из трех попаданий (единиц) и возвращает элементы последовательности, следующие за ними.

Сгенерируйте значительное количество последовательностей длиной 100 и для каждой последовательности найдите все результаты попыток, следующих за тремя попаданиями. Среди этих попыток вычислите процент попаданий. Подсказка: если в последовательности нет трех попаданий подряд, то функция возвращает пустую последовательность, поэтому ваш код должен уметь обрабатывать такие случаи.

Если вы повторите такое моделирование множество раз, каков будет средний процент попаданий? Как меняется этот результат при увеличении или уменьшении длины последовательности?

---

[1] Миллер Дж. Б. (J. B. Miller) и Санджурджо А. (A. Sanjurjo). «Surprised by the Hot Hand Fallacy? A Truth in the Law of Small Numbers», Econometrica 86 (6): 2019-2047.

ГЛАВА 9

Проверка гипотез

В датасетах, которые мы до сих пор исследовали в этой книге, мы наблюдали различия между группами людей (и пингвинов), корреляции между переменными и углы наклона линий регрессии. Подобные результаты называются **наблюдаемыми эффектами**, поскольку они проявляются в выборке — в отличие от реальных эффектов в генеральной совокупности, которые обычно не поддаются непосредственному наблюдению. Когда мы видим такой эффект, стоит задуматься, может ли он присутствовать в более многочисленной генеральной совокупности или его появление в выборке случайно.

Существуют разные способы ответить на этот вопрос, такие как проверка нулевой гипотезы Фишера, теория принятия решений Неймана — Пирсона или байесовская проверка гипотез. Здесь я представлю комбинацию этих подходов, часто используемую на практике.

Эксперимент с монетой

Начнем с простого примера[1]. Когда в 2002 году были введены в обращение монеты евро, один пытливый нумизмат-любитель провел эксперимент с бельгийской монетой номиналом в один евро. Он раскрутил монету на ребре 250 раз, из которых она упала орлом вверх 140 раз, а решкой — 110 раз. Если монета идеально сбалансирована, то стоит ожидать только 125 орлов, так что эти данные говорят о несимметричности монеты. С другой стороны, не стоит ожидать, что в каждом таком эксперименте выпадет ровно 125 орлов, поэтому вполне возможно, что монета на самом деле справедливая[2], а видимое отклонение от ожидаемого значения обусловлено случайностью. Чтобы убедиться в правдоподобности такого объяснения, проведем проверку гипотезы (hypothesis test).

Для симметричной монеты будем использовать следующую функцию, вычисляющую абсолютную разность между наблюдаемым числом выпавших орлов (heads) и ожидаемым (expected):

---

[1] Пример заимствован из книги Д. Дж. Маккея (D. J. MacKay) «Information Theory, Inference and Learning Algorithms» (Cambridge University Press, 2003).

[2] Справедливая монета — понятие из теории вероятностей, означающее монету, при броске которой вероятность выпадения любой из двух сторон (орла или решки) равна. Такая монета считается симметричной. — *Примеч. ред.*

```
n = 250
p = 0.5

def abs_deviation(heads):
    expected = n * p
    return np.abs(heads - expected)
```

В наблюдаемых данных это отклонение составляет 15:

```
heads = 140
tails = 110

observed_stat = abs_deviation(heads)
observed_stat
```

```
15.0
```

Если монета действительно справедливая, то эксперимент с ее вращением можно смоделировать путем генерации последовательности случайных строк — равновероятно либо `H`[1], либо `T`[2] — и подсчитав количество выпадений `H`:

```
def simulate_flips():
    flips = np.random.choice(["H", "T"], size=n)
    heads = np.sum(flips == "H")
    return heads
```

Каждый раз при вызове этой функции мы получаем исход модельного эксперимента:

```
simulate_flips()
```

```
119
```

Цикл ниже повторяет этот эксперимент множество раз, вычисляет отклонение для каждого из них и с помощью спискового включения сохраняет результаты в виде списка:

```
simulated_stats = [abs_deviation(simulate_flips()) for i in range(10001)]
```

В предположении, что монета является справедливой, мы получаем выборку из распределения отклонений. Вот как выглядит это распределение:

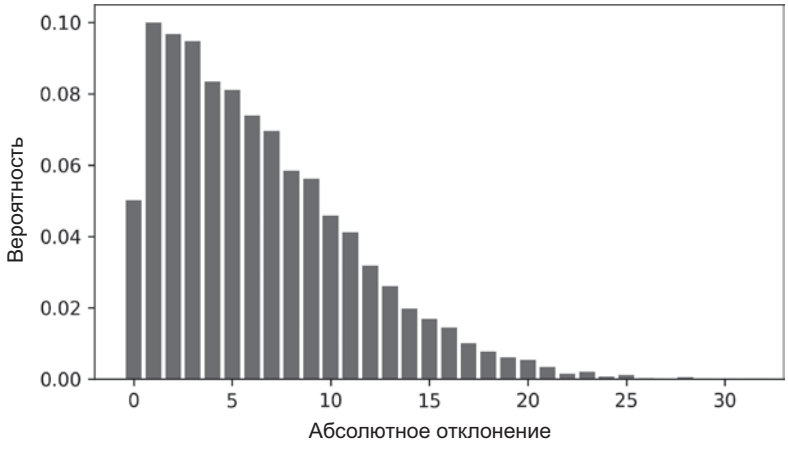
```
from empiricaldist import Pmf

pmf_effects = Pmf.from_seq(simulated_stats)
pmf_effects.bar()

decorate(xlabel="Абсолютное отклонение", ylabel="Вероятность")
```

[1] От англ *heads* — «орел». — *Примеч. пер.*

[2] От англ *tails* — «решка». — *Примеч. пер.*



Значения вблизи 0 — самые распространенные; значения больше 10 встречаются реже. Вспоминая, что отклонение в наблюдаемых данных равно 15, можно прийти к выводу, что отклонения такого порядка редки, но не исключены. В этом примере результаты моделирования больше или равны 15 примерно в 7.1 % случаев:

```
(np.array(simulated_stats) >= 15).mean() * 100
```

```
7.079292070792921
```

Итак, если монета справедливая, то можно ожидать отклонения такой же величины, как наблюдаемое, примерно в 7.1 % случаев, просто в силу случайности.

Это приводит к выводу, что эффект такого масштаба встречается нечасто, но он, безусловно, возможен даже с симметричной монетой. На базе проведенного эксперимента мы не можем исключить возможность того, что монета справедливая.

На этом примере видна логика проверки статистических гипотез:

- Мы начали с наблюдения — 140 выпадений орла из 250 вращений монеты — и гипотезы, что монета несимметричная, другими словами, что вероятность выпадения орла отличается от 50 %.
- Мы выбрали **статистику критерия**, или тестовую статистику (test statistic), которая количественно измеряет величину наблюдаемого эффекта. В данном примере статистикой критерия является абсолютное отклонение от ожидаемого исхода.
- Далее мы определили **нулевую гипотезу**, которая представляет собой модель, построенную на предположении, что наблюдаемый эффект обусловлен случайностью. В данном случае нулевая гипотеза заключается в том, что монета симметричная.
- Затем мы вычислили ***p*-значение**, которое представляет собой вероятность обнаружения наблюдаемого эффекта в условиях верности нулевой гипотезы.

В этом примере *p*-значение — это вероятность отклонения, большего или равного 15.

На последнем шаге интерпретируется результат. Если *p*-значение мало, то делается вывод, что эффект вряд ли может быть обусловлен случайностью. Если оно велико, мы приходим к заключению, что эффект может быть правдоподобно объяснен случайностью. А если он находится где-то посередине, как в этом примере, то можно сделать вывод, что эффект вряд ли появился случайно, но нельзя исключать такую возможность.

Любая проверка гипотезы строится из этих элементов — статистики критерия, нулевой гипотезы и *p*-значения.

Проверка разницы средних

На материале датасета NSFG мы видели, что средняя продолжительность беременности первым ребенком немного дольше, чем последующими детьми. Теперь посмотрим, может ли эта разница быть случайной.

Функция `get_nsfg_groups` считывает данные, выбирает случаи рождения живых младенцев (live) и разделяет их на группы первенцев (firsts) и остальных новорожденных (others):

```
from nsfg import get_nsfg_groups

live, firsts, others = get_nsfg_groups()
```

Теперь можно выбрать продолжительность беременности в неделях для обеих групп:

```
data = firsts["prglngth"].values, others["prglngth"].values
```

Следующая функция принимает на вход данные в виде кортежа из двух последовательностей и вычисляет абсолютную разность средних значений:

```
def abs_diff_means(data):
    group1, group2 = data
    diff = np.mean(group1) - np.mean(group2)
    return np.abs(diff)
```

Наблюдаемая разница в продолжительности беременности первыми и остальными детьми составляет 0.078 недели:

```
observed_diff = abs_diff_means(data)
observed_diff
```

```
0.07803726677754952
```

Итак, гипотеза, которую мы проверим, заключается в том, что продолжительность беременности первым ребенком обычно дольше. Нулевая гипотеза заключается в том, что продолжительность беременности в обеих группах на самом деле одинакова, а кажущаяся разница обусловлена случайностью. Если продолжительность беременности в обеих группах одинакова, то эти группы можно объединить. Чтобы смоделировать такой эксперимент, можно при помощи функции библиотеки NumPy `shuffle` расположить объединенные значения в произвольном порядке, а затем разделить их на две группы первоначального размера с использованием индексов срезов:

```
def simulate_groups(data):
    group1, group2 = data
    n, m = len(group1), len(group2)

    pool = np.hstack(data)
    np.random.shuffle(pool)
    return pool[:n], pool[-m:]
```

Эта функция возвращает кортеж из двух последовательностей, который можно передать в функцию `abs_diff_means`:

```
abs_diff_means(simulate_groups(data))
```

```
0.031193045602279312
```

В следующем цикле многократно повторяется этот эксперимент и для каждого моделируемого датасета вычисляется абсолютная разность средних значений:

```
simulated_diffs = [abs_diff_means(simulate_groups(data)) for i in range(1001)]
```

Чтобы отобразить результаты, воспользуемся функцией ниже, которая принимает выборку результатов моделирования и создает объект `Pmf`, аппроксимирующий ее распределение:

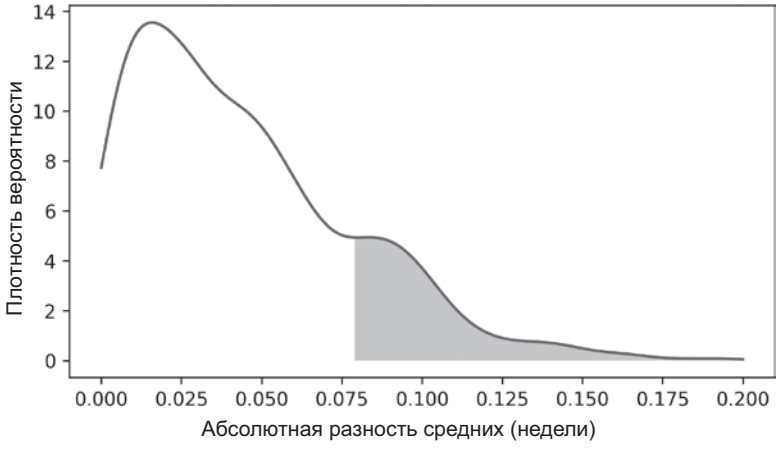
```
from scipy.stats import gaussian_kde
from empiricaldist import Pmf

def make_pmf(sample, low, high):
    kde = gaussian_kde(sample)
    qs = np.linspace(low, high, 201)
    ps = kde(qs)
    return Pmf(ps, qs)
```

Вот как выглядит распределение результатов моделирования. Заштрихованная область показывает случаи, когда при справедливости нулевой гипотезы разница в средних значениях превышает наблюдаемую разницу. Площадь этой области дает *p*-значение:

```
from thinkstats import fill_tail

pmf = make_pmf(simulated_diffs, 0, 0.2)
pmf.plot()
fill_tail(pmf, observed_diff, "right")
decorate(xlabel="Абсолютная разность средних (недели)", ylabel="Плотность
вероятности")
```



Следующая функция вычисляет *p*-значение, представляющее собой долю смоделированных значений, которые больше наблюдаемого или равны ему:

```
def compute_p_value(simulated, observed):
    """Доля смоделированных значений, которые больше наблюдаемого или равны ему."""
    return (np.asarray(simulated) >= observed).mean()
```

В нашем примере *p*-значение примерно равно 18 %, из чего следует, что разница в 0.078 недели может быть правдоподобно объяснена случайностью:

```
compute_p_value(simulated_diffs, observed_diff)
```

```
0.1838161838161838
```

Получив такой результат, мы не можем быть уверены, что продолжительность беременности первым ребенком обычно дольше. Возможно, что наблюдаемая разница в этом датасете обусловлена случайностью.

Обратите внимание, что в обоих примерах проверки гипотез присутствовали одни и те же элементы: статистика критерия, нулевая гипотеза и модель нулевой гипотезы. В последнем примере статистика критерия представляет собой абсолютную разность средних значений. Нулевая гипотеза заключается в том,

что распределение продолжительности беременности в обеих группах на самом деле одинаковое. И мы смоделировали нулевую гипотезу, объединив данные из обеих групп, перетасовав и разделив их на две группы с теми же размерами, что и у исходных данных. Такой процесс перетасовки по-другому называется **перестановкой** (permutation).

Такой вычислительный подход к проверке гипотез позволяет легко комбинировать перечисленные выше элементы для проверки различных статистических гипотез.

Прочие статистики критерия

Возникает вопрос: может ли быть срок беременности первым ребенком не просто более продолжительным, но и более изменчивым? Чтобы проверить эту гипотезу, в качестве статистики критерия можно использовать абсолютную разность стандартных отклонений двух групп. Следующая функция вычисляет эту статистику:

```
def abs_diff_stds(data):
    group1, group2 = data
    diff = np.std(group1) - np.std(group2)
    return np.abs(diff)
```

В данных NSFG разница в стандартных отклонениях составляет около 0.18:

```
observed_diff = abs_diff_stds(data)
observed_diff
```

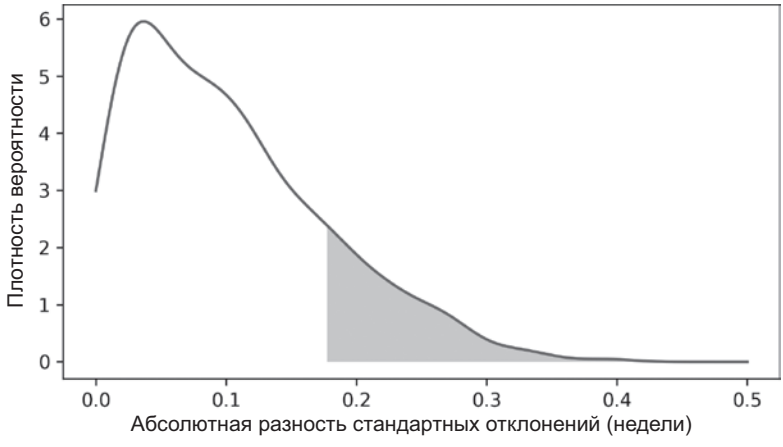
```
0.17600895913991677
```

Чтобы проверить, может ли это различие объясняться случайностью, снова используем перестановку. Следующий цикл многократно моделирует нулевую гипотезу и вычисляет абсолютную разность стандартных отклонений для каждого смоделированного датасета:

```
simulated_diffs = [abs_diff_stds(simulate_groups(data)) for i in range(1001)]
```

Вот как выглядит распределение результатов. Штриховка по-прежнему обозначает область, где статистика критерия, при условии истинности нулевой гипотезы, превышает наблюдаемую разницу:

```
pmf = make_pmf(simulated_diffs, 0, 0.5)
pmf.plot()
fill_tail(pmf, observed_diff, "right")
decorate(xlabel="Абсолютная разность стандартных отклонений (недели)",
ylabel="Плотность вероятности")
```



Площадь этой области можно оценить, вычислив долю результатов, которые равны наблюдаемой разнице или превышают ее:

```
compute_p_value(simulated_diffs, observed_diff)
```

```
0.17082917082917082
```

В данном случае *p*-значение составляет около 0.17, так что даже в случае, если эти две группы одинаковы, появление разницы такой величины вполне правдоподобно. Подводя итог, мы не можем быть уверены в том, что продолжительность беременности первым ребенком в целом более изменчива: разница, присутствующая в этом датасете, может быть случайной.

Проверка значимости корреляции

Такую же схему можно использовать для проверки значимости корреляции. Например, в датасете NSFG есть корреляция между весом новорожденного и возрастом матери: у матерей более старшего возраста в среднем рождаются более крупные дети. Но может ли это явление объясняться случайностью?

Давайте выясним. Начнем с подготовки данных. Из всех случаев рождения живых детей выберем только те, когда известны возраст матери и вес новорожденного:

```
valid = live.dropna(subset=["agepreg", "totalwgt_lb"]) valid.shape
```

```
(9038, 244)
```

Теперь выберем нужные столбцы:

```
ages = valid["agepreg"]
birthweights = valid["totalwgt_lb"]
```

Следующая функция принимает на вход кортеж из двух последовательностей, `xs` и `ys`, и вычисляет модуль корреляции, которая может быть положительной или отрицательной:

```
def abs_correlation(data):
    xs, ys = data
    corr = np.corrcoef(xs, ys)[0, 1]
    return np.abs(corr)
```

В датасете NSFG корреляция составляет около 0.07:

```
data = ages, birthweights
observed_corr = abs_correlation(data)
observed_corr
```

```
0.0688339703541091
```

В качестве нулевой гипотезы примем отсутствие корреляции между возрастом матери и весом новорожденного. Перетасовав наблюдаемые значения, смоделируем ситуацию, в которой распределения возраста матери и веса новорожденного остаются теми же, но переменные перестают быть связаны.

Следующая функция принимает кортеж из последовательностей `xs` и `ys`, перетасовывает `xs` и возвращает кортеж, содержащий перетасованную `xs` и исходную `ys`. Мы бы добились того же результата, если бы перетасовали только `ys` или обе последовательности одновременно:

```
def permute(data):
    xs, ys = data
    new_xs = xs.values.copy()
    np.random.shuffle(new_xs)
    return new_xs, ys
```

Корреляция перетасованных значений обычно близка к 0:

```
abs_correlation(permute(data))
```

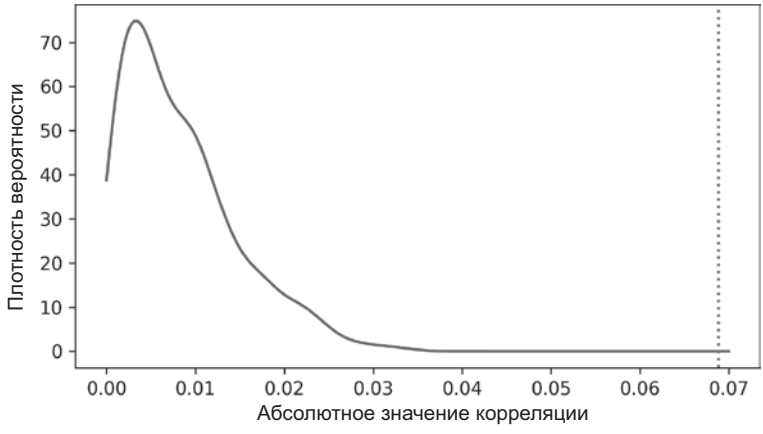
```
0.0019269515502894237
```

В цикле ниже генерируется множество перетасованных датасетов и в каждом из них вычисляется корреляция:

```
simulated_corrs = [abs_correlation(permute(data)) for i in range(1001)]
```

Вот как выглядит распределение полученных результатов. Вертикальная пунктирная линия обозначает наблюдаемую корреляцию:

```
pmf = make_pmf(simulated_corrs, 0, 0.07)
pmf.plot()
plt.axvline(observed_corr, color="gray", ls=":")
decorate(xlabel="Абсолютное значение корреляции", ylabel="Плотность
вероятности")
```



Обращает на себя внимание то, что наблюдаемая корреляция находится в хвосте распределения, где видимая область под кривой отсутствует. Если мы попытаемся вычислить *p*-значение, результат будет равен 0, это указывает на то, что ни в одной из моделей корреляция в перетасованных данных не превышает наблюдаемого значения:

```
compute_p_value(simulated_corrs, observed_corr)
```

```
0.0
```

Эти цифры говорят, что значение *p*, вероятно, меньше 1 на 1000, хотя оно на самом деле и не равно нулю. Маловероятно, что корреляция перетасованных данных превысит наблюдаемое значение, но такую возможность нельзя исключать.

Когда *p*-значение мало, по сложившейся традиции меньше 0.05, можно сказать, что результат **статистически значим**. Но такая интерпретация *p*-значения всегда имела свои недостатки, и постепенно от нее отказываются.

Один из таких недостатков — произвольность традиционного значения порога, которое к тому же подходит не для всех применений. Другая сложность заключается в том, что использование термина «значимый» может приводить к неверным выводам, поскольку оно намекает, что эффект важен на практике. Корреляция между возрастом матери и весом новорожденного — хороший пример. Она статистически значима, но настолько мала, что практически не важна.

С учетом вышеказанного можно прибегнуть к качественной интерпретации *p*-значения:

- если *p*-значение велико, то вполне вероятно, что наблюдаемый эффект мог возникнуть случайно;
- если *p*-значение мало, то, как правило, вероятность того, что эффект вызван случайностью, можно исключить. Но нужно помнить, что он все еще может объясняться нерепрезентативностью выборки или ошибками измерений.

Проверка пропорций

В качестве завершающего примера данной главы возьмем случай, когда выбор статистики критерия не так очевиден. Представьте, что вы управляете казино и подозреваете, что какой-то клиент жульничает и пользуется игральной костью, модифицированной таким образом, чтобы одна из граней выпадала с большей вероятностью, чем другие. Вы задерживаете предполагаемого мошенника и конфискуете игральную кость, но теперь вам нужно доказать, что она поддельная. Вы бросаете кубик 60 раз и записываете число повторений каждого исхода от 1 до 6. Вот результаты в виде объекта класса `Hist`:

```
from empiricaldist import Hist

qs = np.arange(1, 7)
freqs = [8, 9, 19, 5, 8, 11]
observed = Hist(freqs, qs)
observed.index.name = "исход"
observed
```

исход	частота
1	8
2	9
3	19
4	5
5	8
6	11

Вы ожидаете, что в среднем каждое значение должно выпадать 10 раз. В этих данных значение 3 появляется чаще, чем ожидалось, а значение 4 — реже. Но могут ли такие расхождения возникать по стечению обстоятельств?

Чтобы проверить эту гипотезу, начнем с вычисления ожидаемой частоты (expected) всех исходов (outcomes):

```
num_rolls = observed.sum()
outcomes = observed.qs
expected = Hist(num_rolls / 6, outcomes)
```

Следующая функция принимает на вход наблюдаемые частоты и вычисляет сумму их абсолютных разностей с ожидаемыми частотами:

```
def total_abs_deviation(observed):
    return np.sum(np.abs(observed - expected))
```

У наблюдаемых данных эта статистика критерия равна 20:

```
observed_dev = total_abs_deviation(observed)
observed_dev
```

```
20.0
```

Функция ниже принимает на вход наблюдаемые данные, моделирует бросание кости то же самое количество раз и возвращает объект класса `Hist`, содержащий смоделированные частоты:

```
def simulate_dice(observed):
    num_rolls = np.sum(observed)
    rolls = np.random.choice(observed.qs, num_rolls, replace=True)
    hist = Hist.from_seq(rolls)
    return hist
```

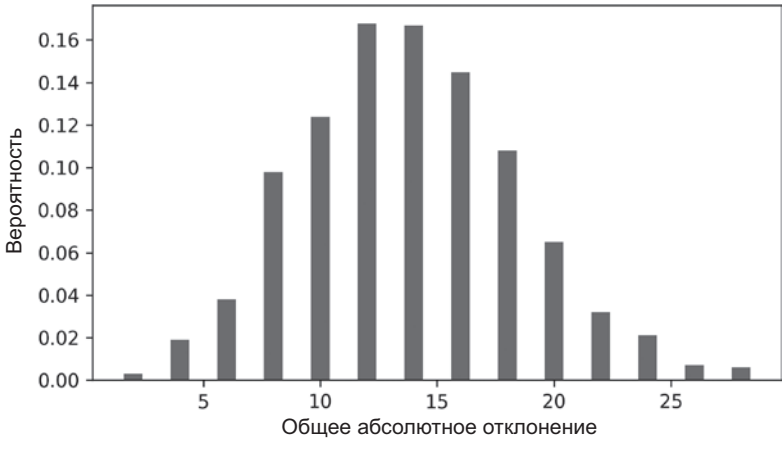
В следующем цикле многократно повторяется данный эксперимент и вычисляется общее абсолютное отклонение для каждого из них:

```
simulated_devs = [total_abs_deviation(simulate_dice(observed))
                  for i in range(1001)]
```

Вот как выглядит распределение статистики критерия в условиях нулевой гипотезы. Обратите внимание, что общая сумма всегда четная, потому что каждый раз, когда один из исходов выпадает чаще, чем ожидалось, какой-то другой исход автоматически выпадает реже:

```
pmf_devs = Pmf.from_seq(simulated_devs)
pmf_devs.bar()

decorate(xlabel="Общее абсолютное отклонение", ylabel="Вероятность")
```



Теперь становится понятно, что общее отклонение в 20 очков не является необычным. Его *p*-значение составляет около 13 %, поэтому нельзя с уверенностью заявить, что игральная кость поддельная:

```
compute_p_value(simulated_devs, observed_dev)
```

```
0.13086913086913088
```

Но выбранная нами статистика критерия — не единственно возможный вариант. Для решения подобной задачи чаще принято использовать статистику хи-квадрат, которую можно вычислить так:

```
def chi_squared_stat(observed):
    diffs = (observed - expected) ** 2
    return np.sum(diffs / expected)
```

Возведение отклонений в квадрат (вместо использования абсолютных значений) придает больший вес значительным отклонениям. Деление на значения из `expected` стандартизирует отклонения, хотя в данном случае это не влияет на результаты, поскольку все ожидаемые частоты равны:

```
observed_chi2 = chi_squared_stat(observed)
observed_chi2
```

```
11.6
```

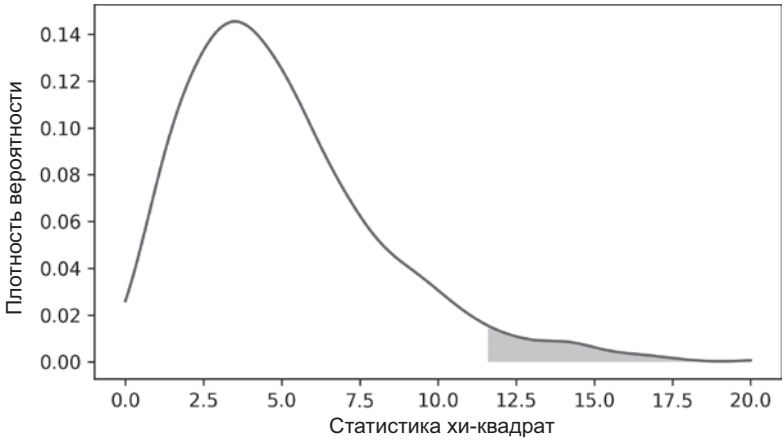
Статистика хи-квадрат наблюдаемых данных равна 11.6. Эта цифра сама по себе не имеет большого значения, но ее можно сравнить с результатами моделирования. В следующем цикле генерируется множество модельных датасетов и вычисляется статистика хи-квадрат для каждого из них:

```
simulated_chi2 = [chi_squared_stat(simulate_dice(observed)) for i
in range(1001)]
```

Вот как выглядит распределение этой статистики критерия в условиях нулевой гипотезы.

Штриховкой обозначена область, где результаты превышают наблюдаемое значение:

```
pmf = make_pmf(simulated_chi2, 0, 20)
pmf.plot()
fill_tail(pmf, observed_chi2, "right")
decorate(xlabel="Статистика хи-квадрат", ylabel="Плотность вероятности")
```



Площадь заштрихованной области опять же дает *p*-значение:

```
compute_p_value(simulated_chi2, observed_chi2)
```

```
0.04495504495504495
```

Полученное с помощью статистики хи-квадрат *p*-значение составляет около 0.04, что значительно меньше, чем получилось по формуле общего отклонения — 0.13. Если серьезно относиться к пороговому значению в 5 %, то можно считать этот эффект статистически значимым. Но, принимая во внимание результаты обеих проверок, я бы сказал, что результаты неубедительны. Я бы не исключал возможности того, что игральная кость поддельная, но и не стал бы осуждать подозреваемого в мошенничестве.

Этот пример демонстрирует один важный момент: *p*-значение зависит от выбора как статистики критерия, так и модели нулевой гипотезы. И принятые решения иногда определяют, является эффект статистически значимым или нет.

Глоссарий

Проверка гипотез (hypothesis testing)

Набор методов для проверки правдоподобности того, что наблюдаемый эффект может быть следствием случайной выборки.

Статистика критерия (test statistic)

Статистический показатель, используемый при проверке гипотезы для количественной оценки величины наблюдаемого эффекта.

Нулевая гипотеза (null hypothesis)

Статистическая модель, основанная на предположении, что эффект, наблюдаемый в выборке, отсутствует в генеральной совокупности.

Перестановка (permutation)

Способ моделирования нулевой гипотезы путем случайного перемешивания набора данных.

p-значение (p-value)

Вероятность появления эффекта величиной с тот, который наблюдается в условиях нулевой гипотезы.

Статистически значимый (statistically significant)

Эффект статистически значим, если *p*-значение меньше выбранного порога, часто равного 5 %. При большом объеме данных наблюдаемый эффект может быть статистически значимым, даже если он слишком мал, чтобы иметь практическое значение.

Упражнения

Упражнение 9.1

Испытаем методику проверки гипотез на данных о пингвинах из раздела «Выборочное распределение» главы 8 на с. 151. Инструкции по загрузке данных приведены в Jupyter-блокноте этой главы.

Для начала считаем данные и выберем пингвинов вида антарктический пингвин:

```
penguins = pd.read_csv("penguins_raw.csv").dropna(subset=["Body Mass (g)"])
chinstrap = penguins.query('Species.str.startswith("Chinstrap")')
chinstrap.shape
```

```
(68, 17)
```

Теперь извлечем вес самцов (male) и самок (female) пингвинов в килограммах:

```
male = chinstrap.query("Sex == 'MALE'")
weights_male = male["Body Mass (g)"] / 1000
weights_male.mean()
```

```
3.9389705882352937
```

```
female = chinstrap.query("Sex == 'FEMALE'")
weights_female = female["Body Mass (g)"] / 1000
weights_female.mean()
```

```
3.5272058823529413
```

Используйте функции `abs_diff_means` и `simulate_groups` для моделирования большого количества датасетов в условиях нулевой гипотезы равенства распределений весов двух групп и вычислите разницу в средних значениях для каждого из датасетов. Сравните результаты моделирования с наблюдаемой разницей и вычислите *p*-значение. Насколько правдоподобно то, что видимая разница между группами обусловлена случайностью?

Упражнение 9.2

Извлечем из данных о пингвинах из предыдущего упражнения глубину (depth) и длину (length) верхнего края клюва (culmen) самок пингвинов:

```
data = female["Culmen Depth (mm)"], female["Culmen Length (mm)"]
```

Корреляция между этими переменными примерно равна 0.26:

```
observed_corr = abs_correlation(data)
observed_corr
```

```
0.2563170802728449
```

Проверим, может ли эта корреляция возникнуть случайно, когда ее в действительности нет. Используйте функцию `permute`, чтобы создать множество перестановок этих данных, и функцию `abs_correlation`, чтобы вычислить корреляцию для каждой такой перестановки. Постройте график распределения корреляций в условиях нулевой гипотезы и вычислите *p*-значение для наблюдаемой корреляции. Как можно интерпретировать этот результат?

ГЛАВА 10

Метод наименьших квадратов

В этой и следующей главах представлена концепция подгонки модели к данным. В данном контексте **модель** состоит из математического описания взаимосвязи между переменными (например, в виде прямой линии) и описания случайной изменчивости (например, в виде нормального распределения).

Когда говорят, что модель соответствует данным, обычно подразумевают, что она минимизирует ошибки, которые представляют собой расстояния между модельной аппроксимацией и данными. Начнем с одного из самых распространенных способов подгонки модели, минимизирующего сумму квадратов ошибок, — метода наименьших квадратов (МНК).

Сначала вы познакомитесь с моделями, которые работают только с двумя переменными. В следующей главе представлены модели, которые способны описывать большее количество переменных.

Линейная регрессия методом наименьших квадратов

Для примера вернемся к сценарию, описанному в разделе «Вес пингвинов» главы 8 на с. 144. Предположим, вы исследователь в Антарктиде, изучающий местные популяции пингвинов. В процессе сбора данных вы отлавливаете выборку пингвинов, измеряете и взвешиваете их, а затем отпускаете целыми и невредимыми.

Как вскоре становится известно, не так просто заставить пингвина стоять на весах нужное время, чтобы провести точные измерения. Представьте, что для некоторых пингвинов у нас есть такие показатели, как размеры крыльев и клюва, но нет показаний веса. Посмотрим, сможем ли мы использовать другие измерения для получения недостающих данных. Этот процесс называют **восстановлением пропущенных данных** или импутацией (imputation).

Начнем с изучения взаимосвязи между весом и размерами, используя данные, собранные в период с 2007 по 2010 год исследователями на станции Палмер в Антарктиде. Собранные ими данные находятся в свободном доступе: инструкции по их загрузке приведены в Jupyter-блокноте этой главы.

Для чтения данных можно использовать функцию Pandas `read_csv`:

```
penguins = pd.read_csv("penguins_raw.csv").dropna(subset=["Body Mass (g)"])
penguins.shape
```

```
(342, 17)
```

В датасете имеются измерения для 151 пингвина Адели (Adélie penguin). Воспользуемся методом `query` для выбора строк, содержащих эти данные:

```
adelie = penguins.query('Species.str.startswith("Adelie")')
len(adelie)
```

```
151
```

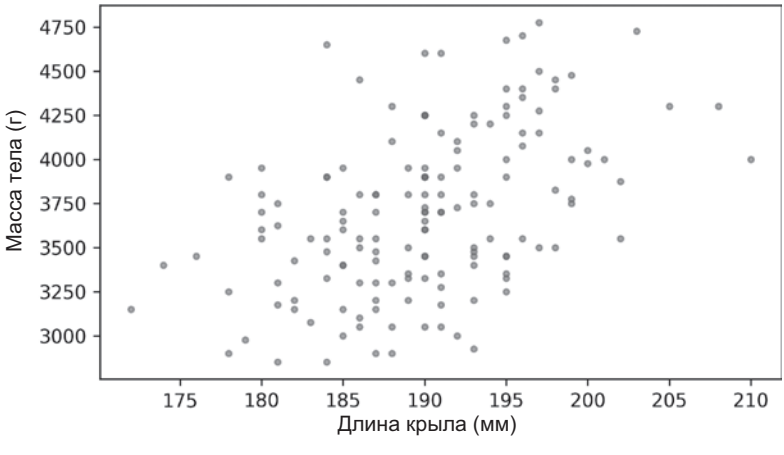
Теперь посмотрим, насколько точно можно предсказать вес пингвина Адели, если известна длина его крыльев (flipper length). Сначала извлечем эти данные из объекта `DataFrame`:

```
xvar = "Flipper Length (mm)"
yvar = "Body Mass (g)"

flipper_length = adelie[xvar]
body_mass = adelie[yvar]
```

Вот диаграмма рассеяния, показывающая взаимосвязь между этими величинами:

```
plt.scatter(flipper_length, body_mass, marker=".", alpha=0.5)
decorate(xlabel="Длина крыла (мм)", ylabel="Масса тела (г)")
```



Похоже, что они связаны, — оценим количественно силу этой взаимосвязи, вычислив коэффициент корреляции:

```
np.corrcoef(flipper_length, body_mass)[0, 1]
```

```
0.4682016942179394
```

Корреляция составляет около 0.47, поэтому пингвины с более длинными крыльями, как правило, тяжелее. Знание такого факта полезно, поскольку предполагает, что мы можем более точно определить вес пингвина, если знаем длину его

крыльев. Но корреляция сама по себе не говорит нам, как получать такие приблизительные оценки. Для этого нам нужно найти **линию наилучшего соответствия**.

Существует много способов определить «наилучшую» линию, но для таких данных обычно используется **линейная регрессия методом наименьших квадратов**, которая дает прямую линию, минимизирующую средний квадрат ошибки (MSE).

В библиотеке SciPy есть функция `linregress`, которая выполняет подгонку по методу наименьших квадратов. Название функции — это сокращение от **линейной регрессии** (linear regression), еще одного названия описанной модели. Аргументами функции `linregress` являются значения `x` и `y` в указанном порядке:

```
from scipy.stats import linregress

result = linregress(flipper_length, body_mass)
result
```

```
LinregressResult(slope=32.83168975115009, intercept=-2535.8368022002514,
rvalue=0.46820169421793933, pvalue=1.3432645947790051e-09,
stderr=5.076138407990821, intercept_stderr=964.7984274994059)
```

Возвращаемое значение — объект `LinregressResult`, содержащий значения углового коэффициента, или наклона (slope), и свободного члена, или пересечения оси ординат (intercept), подогнанной линии, а также другую информацию, которую мы скоро разберем. Наклон составляет около 32.8, это означает, что на каждый дополнительный миллиметр длины крыла приходятся дополнительные 32.8 г веса тела.

Пересечение оси ординат происходит в точке –2535 г, что может показаться абсурдным, поскольку измеренный вес не может быть отрицательным. Возможно, эта цифра приобретет больший смысл, если использовать значения углового коэффициента и свободного члена для вычисления ординаты подогнанной линии (линии регрессии) для средней длины крыльев:

```
x = flipper_length.mean()
y = result.intercept + result.slope * x
x, y
```

```
(189.95364238410596, 3700.662251655629)
```

Для пингвина со средней длиной крыльев около 190 мм ожидаемый вес тела составляет около 3700 г.

Следующая функция извлекает параметры из объекта, возвращаемого функцией `linregress`, и для каждого значения `x` из последовательности `xs` находит ординату точки на подогнанной линии:

```
def predict(result, xs):
    ys = result.intercept + result.slope * xs
    return ys
```

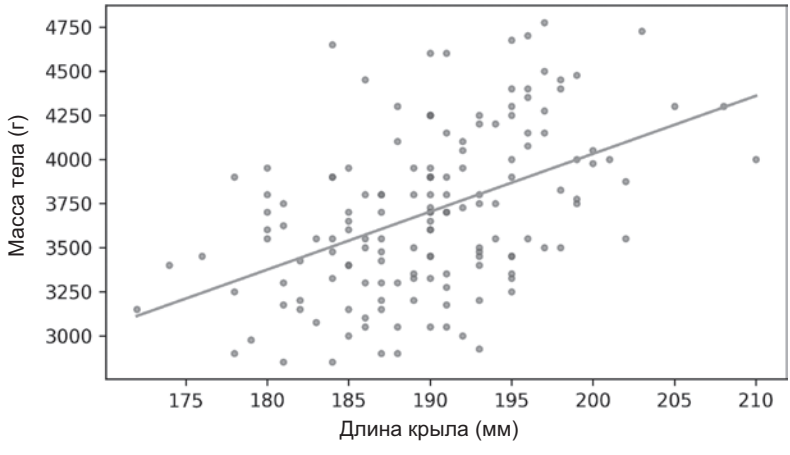
Название функции, `predict`, здесь может показаться странным: в человеческом языке **предсказание** (prediction) обычно относится к происходящему в будущем, но в контексте регрессии точки на подогнанной линии также называются предсказаниями.

Воспользуемся функцией `predict`, чтобы найти точки на подогнанной линии для некоторого диапазона размеров плавников:

```
fit_xs = np.linspace(np.min(flipper_length), np.max(flipper_length))
fit_ys = predict(result, fit_xs)
```

Вот линия, совмещенная с точечной диаграммой данных:

```
plt.scatter(flipper_length, body_mass, marker=".", alpha=0.5)
plt.plot(fit_xs, fit_ys, color="C1")
decorate(xlabel="Длина крыла (мм)", ylabel="Масса тела (г)")
```



Как и ожидалось, линия проходит через центр облака точек данных и отражает тенденцию. Некоторые из полученных прогнозов точны, но множество точек данных находятся далеко от линии. Чтобы понять, насколько хорошо (или плохо) предсказываются данные, можно вычислить ошибку предсказания (prediction error), равную расстоянию по вертикали от каждой точки до линии. Следующая функция вычисляет эти ошибки, которые также называются **остатками** (residuals):

```
def compute_residuals(result, xs, ys):
    fit_ys = predict(result, xs)
    return ys - fit_ys
```

Ниже приведены значения остатков для массы тела как функции длины крыла:

```
residuals = compute_residuals(result, flipper_length, body_mass)
```

В качестве примера посмотрим на результаты для первого пингвина из датасета:

```
x = flipper_length[0]
y = predict(result, x)
x, y
```

```
(181.0, 3406.699042757914)
```

Длина крыльев выбранного пингвина составляет 181 мм, а предсказанная масса тела — 3407 г. Теперь узнаем, какова его реальная масса:

```
body_mass[0], residuals[0]
```

```
(3750.0, 343.30095724208604)
```

Фактическая масса этого пингвина составляет 3750 г, а остаток после вычитания предсказанного значения — 343 г.

Среднее значение квадратов остатков — это средний квадрат ошибки (MSE) предсказаний:

```
mse = np.mean(residuals**2)
mse
```

```
163098.85902884745
```

Само по себе это число не имеет особого значения. Оно станет более осмысленным, когда мы вычислим коэффициент детерминации.

Коэффициент детерминации

Представьте, что вы хотите угадать вес пингвина. Если вы знаете длину его крыльев, то можете использовать метод наименьших квадратов, чтобы предсказать вес, а значение MSE — для количественной оценки средней точности предсказаний.

Но что, если вы не знаете длину крыльев? Какое предположение вы сделаете в этом случае? Оказывается, предсказывать среднее значение — это лучшая стратегия с точки зрения минимизации MSE. Если мы всегда предсказываем среднее значение, то ошибки предсказания — это отклонения (deviations) от среднего значения:

```
deviations = body_mass - np.mean(body_mass)
```

А MSE — это средний квадрат отклонения (mean squared deviation):

```
np.mean(deviations**2)
```

```
208890.28989956583
```

Возможно, вы помните, что средний квадрат отклонения — это дисперсия:

```
np.var(body_mass)
```

```
208890.28989956583
```

Таким образом, если мы всегда угадываем среднее значение, то в качестве MSE можно рассматривать дисперсию масс, а если используем линию регрессии — то дисперсию остатков. Если вычислить отношение этих величин и вычесть его из 1, то результат покажет, насколько уменьшается MSE при использовании длины крыльев для обоснования наших предположений.

Следующая функция вычисляет это значение, которое формально называется **коэффициентом детерминации**, но, поскольку оно обозначается как R^2, многие называют его «R квадрат».

```
def coefficient_of_determination(ys, residuals):
    return 1 - np.var(residuals) / np.var(ys)
```

В приведенном примере значение R^2 равно примерно 0.22, это означает, что подогнанная линия уменьшает MSE на 22 %:

```
R2 = coefficient_of_determination(body_mass, residuals)
R2
```

```
0.21921282646854912
```

Оказывается, существует взаимосвязь между коэффициентом детерминации R^2 и коэффициентом корреляции r. Как вы могли бы догадаться, глядя на эти условные обозначения, $r^2 = R^2$. В этом можно убедиться, вычислив квадратный корень из R^2:

```
r = np.sqrt(R2)
r
```

```
0.4682016942179397
```

И сравнив его с коэффициентом корреляции, который мы считали ранее:

```
corr = np.corrcoef(flipper_length, body_mass)[0, 1]
corr
```

```
0.4682016942179394
```

Они одинаковы, если не считать небольшой разницы, обусловленной погрешностью операций с плавающей точкой.

Функция `linregress` также вычисляет это значение и возвращает его в качестве атрибута объекта `RegressionResult`:

```
result.rvalue
```

```
0.46820169421793933
```

Коэффициенты детерминации и корреляции содержат главным образом одну и ту же информацию, но интерпретируются по-разному:

- корреляция количественно выражает силу взаимосвязи на шкале от –1 до 1;
- R^2 измеряет способность подогнанной линии уменьшить MSE.

Кроме того, величина R^2 всегда положительна, поэтому по ней нельзя понять, является корреляция положительной или отрицательной.

Минимизация MSE

Ранее я упоминал, что результатом подгонки методом наименьших квадратов является прямая линия, которая минимизирует средний квадрат ошибки (MSE). Мы не будем это доказывать, но можем проверить, если добавим небольшие случайные значения к свободному члену (intercept) и коэффициенту наклона (slope) и посмотрим, не ухудшает ли это MSE:

```
intercept = result.intercept + np.random.normal(0, 1)
slope = result.slope + np.random.normal(0, 1)
```

Чтобы прогнать этот тест, нужно создать объект с атрибутами `intercept` и `slope`. Воспользуемся для этого объектом `SimpleNamespace`, определенным в модуле `types`:

```
from types import SimpleNamespace

fake_result = SimpleNamespace(intercept=intercept, slope=slope)
fake_result
```

```
namespace(intercept=-2535.738911382989, slope=34.24509022936497)
```

Передадим этот объект в функцию `compute_residuals` и используем полученные остатки для вычисления MSE:

```
fake_residuals = compute_residuals(fake_result, flipper_length, body_mass)
fake_mse = np.mean(fake_residuals**2)
```

Если сравнить полученный результат с MSE для линии наименьших квадратов, то первый будет всегда хуже:

```
mse, fake_mse, fake_mse > mse
```

```
(163098.85902884745, 235318.11301937344, True)
```

Достичь минимума MSE — хорошо, но это не единственный критерий «наилучшей» модели. Альтернативный вариант — минимизировать абсолютные значения ошибок. Еще один — минимизировать кратчайшее расстояние от каждой точки до подогнанной линии, которое называется **суммарной ошибкой** (total error).

В одних случаях предполагаемое значение лучше завысить, в других — занизить. В таком случае может понадобиться вычислить функцию потерь (cost function) для каждого остатка и минимизировать общую величину потерь (cost).

Впрочем, метод наименьших квадратов используется гораздо чаще, чем упомянутые альтернативы, в первую очередь потому, что он вычислительно эффективен. Следующая функция демонстрирует это:

```
def least_squares(xs, ys):
    xbar = np.mean(xs)
    ybar = np.mean(ys)

    xdev = xs - xbar
    ydev = ys - ybar

    slope = np.sum(xdev * ydev) / np.sum(xdev**2)
    intercept = ybar - slope * xbar

    return intercept, slope
```

Для проверки этой функции снова используем длину крыльев и массу тела:

```
intercept, slope = least_squares(flipper_length, body_mass)
intercept, slope
```

```
(-2535.8368022002524, 32.831689751150094)
```

И можно убедиться, что получены те же результаты, что и при использовании функции `linregress`:

```
np.allclose([intercept, slope], [result.intercept, result.slope])
```

```
True
```

Минимизация MSE имела смысл, когда эффективность вычислений превалировала над выбором метода, наиболее подходящего для конкретной задачи. Но это более не актуально, поэтому стоит поразмышлять о том, насколько правильно минимизировать именно квадраты остатков.

Оценка

Параметры `slope` и `intercept` — это оценки, полученные на основе некоторой выборки. Подобно другим оценкам, на них сказываются нерепрезентативность выборки, погрешности измерений и изменчивость вследствие случайности выборки. Как правило, влияние нерепрезентативной выборки и ошибок измерений численно оценить трудно. Гораздо проще измерить влияние процесса составления случайной выборки.

Один из способов это сделать — использовать разновидность ресемплинга, называемую **бутстрепингом** (bootstrapping). Будем считать, что выборка представляет собой всю популяцию, и генерировать из наблюдаемых данных новые выборки с возвращением (replacement). Следующая функция принимает на вход объект

DataFrame, использует метод sample для повторной выборки строк и возвращает новый объект DataFrame:

```
def resample(df):
    n = len(df)
    return df.sample(n, replace=True)
```

А функция ниже принимает на вход объект DataFrame, находит линию наилучшего соответствия по методу наименьших квадратов и возвращает наклон найденной подогнанной линии:

```
def estimate_slope(df):
    xs, ys = df["Flipper Length (mm)"], df["Body Mass (g)"]
    result = linregress(xs, ys)
    return result.slope
```

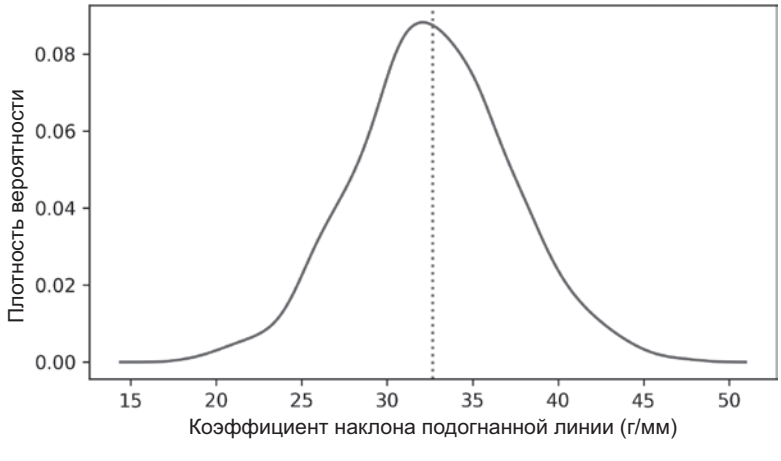
Воспользуемся этими функциями для создания множества моделей датасетов и вычисления коэффициента наклона в случае каждого из них:

```
resampled_slopes = [estimate_slope(resample(adelie)) for i in range(1001)]
```

В результате получаем выборку из выборочного распределения коэффициента наклона. Вот как она выглядит:

```
from thinkstats import plot_kde

plot_kde(resampled_slopes)
decorate(xlabel="Коэффициент наклона подогнанной линии (г/мм)",
ylabel="Плотность вероятности")
```



С помощью функции percentile можно вычислить 90%-ный доверительный интервал (ДИ):

```
ci90 = np.percentile(resampled_slopes, [5, 95])
print(result.slope, ci90)
```

```
32.83168975115009 [25.39604591 40.21054526]
```

Таким образом, в своем отчете мы могли бы указать, что оценка углового коэффициента составляет 33 г/мм с 90%-ным ДИ [25, 40] г/мм.

Стандартная ошибка оценки — это стандартное отклонение выборочного распределения:

```
stderr = np.std(resampled_slopes)
stderr
```

```
4.570238986584832
```

Объект `RegressionResult`, возвращаемый функцией `linregress`, содержит аппроксимацию стандартной ошибки, исходя из некоторых предположений о форме распределения:

```
result.stderr
```

```
5.076138407990821
```

Стандартная ошибка, которую мы вычислили путем ресемплинга, была немного меньше, но на практике разница, вероятно, роли не играет.

Визуализация неопределенности

Каждый раз, когда мы проводим ресемплинг датасета, мы получаем новую подогнанную линию. Чтобы увидеть, насколько велик разброс линий, можно, например, пройтись по ним циклом и отобразить их все. Функция ниже принимает на вход результат ресемплинга в виде объекта `DataFrame`, находит линию наилучшего соответствия по методу наименьших квадратов и для последовательности `xs` генерирует предсказанные значения:

```
def fit_line(df, fit_xs):
    xs, ys = df["Flipper Length (mm)"], df["Body Mass (g)"]
    result = linregress(xs, ys)
    fit_ys = predict(result, fit_xs)
    return fit_ys
```

Вот последовательность `xs`, которую мы будем использовать:

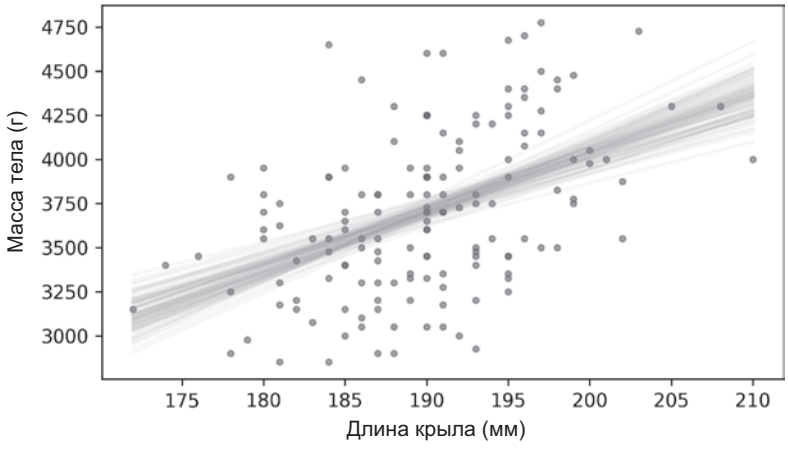
```
xs = adelie["Flipper Length (mm)"]
fit_xs = np.linspace(np.min(xs), np.max(xs))
```

А вот как выглядят подогнанные линии, совмещенные с диаграммой рассеяния исходных данных:

```
plt.scatter(flipper_length, body_mass, marker=".", alpha=0.5)

for i in range(101):
    fit_ys = fit_line(resample(adelie), fit_xs)
    plt.plot(fit_xs, fit_ys, color="C1", alpha=0.05)

decorate(xlabel="Длина крыла (мм)", ylabel="Масса тела (г)")
```



В середине диаграммы линии расположены близко друг к другу, а в крайних точках — расходятся.

Другой способ представить изменчивость подогнанных линий — построить 90%-ный доверительный интервал для каждого предсказанного значения. Начнем с того, что соберем подогнанные линии в списке массивов:

```
fitted_ys = [fit_line(resample(adelie), fit_xs) for i in range(1001)]
```

Этот список массивов можно считать двумерным массивом, где каждая строка соответствует подогнанной линии, а столбец — значению из `xs`.

Чтобы найти 5-й, 50-й и 95-й процентили `ys`, соответствующие каждому из значений в `xs`, можно вызвать функцию `percentile` с аргументом `axis=0`:

```
low, median, high = np.percentile(fitted_ys, [5, 50, 95], axis=0)
```

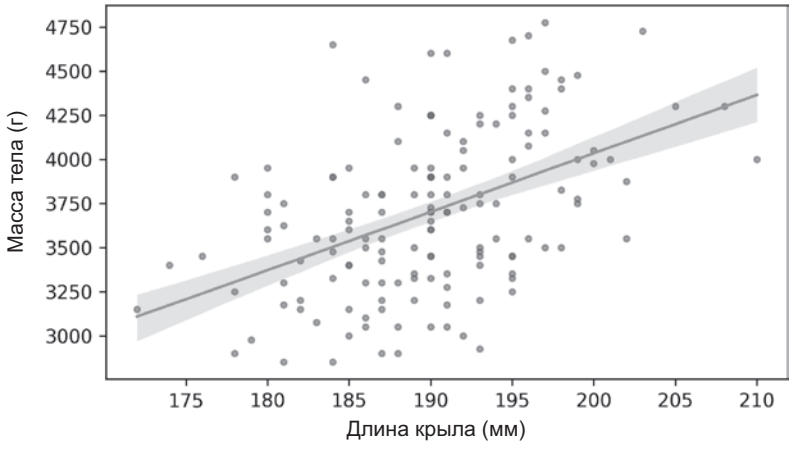
Теперь воспользуемся Matplotlib-функцией `fill_between` для заполнения области между 5-м и 95-м процентилями, представляющей 90 % ДИ, отображения медианных значений всех столбцов и отрисовки точечной диаграммы исходных данных:

```
plt.scatter(flipper_length, body_mass, marker=".", alpha=0.5)

plt.fill_between(fit_xs, low, high, color="C1", lw=0, alpha=0.2)
```

```
plt.plot(fit_xs, median, color="C1")

decorate(xlabel="Длина крыла (мм)", ylabel="Масса тела (г)")
```



Это мой любимый способ визуализации изменчивости подогнанной линии, обусловленной случайностью выборки.

Преобразование переменных

Перед тем как приступать к подгонке линии регрессии данных, иногда полезно преобразовать одну или обе переменные — например, возведя значения в квадрат либо рассчитав их квадратный корень или логарифм. В качестве примера рассмотрим данные о росте и весе из Системы наблюдения за поведенческими факторами риска (BRFSS), описанные в разделе «Логнормальное распределение» главы 5 на с. 94.

Сначала загрузим данные BRFSS:

```
from thinkstats import read_brfss

brfss = read_brfss()
```

Далее отберем строки с корректными данными и выберем столбцы, содержащие значения роста и веса:

```
valid = brfss.dropna(subset=["htm3", "wtkg2"])
heights, weights = valid["htm3"], valid["wtkg2"]
```

С помощью функции `linregress` можно по методу наименьших квадратов получить значения коэффициента наклона и точки пересечения ординаты линией регрессии:

```
result_brfss = linregress(heights, weights)
result_brfss.intercept, result_brfss.slope
```

```
(-82.65926054409877, 0.957074585033226)
```

Коэффициент наклона составляет около 0.96, что можно интерпретировать как прибавление в среднем почти 1 кг веса на каждый добавочный 1 см роста. Снова воспользуемся функцией `predict` для предсказания значений зависимой переменной в диапазоне значений независимой `xs`:

```
fit_xs = np.linspace(heights.min(), heights.max())
fit_ys = predict(result_brfss, fit_xs)
```

Прежде чем строить диаграмму рассеяния данных, полезно применить джиттеринг к значениям роста и веса:

```
from thinkstats import jitter

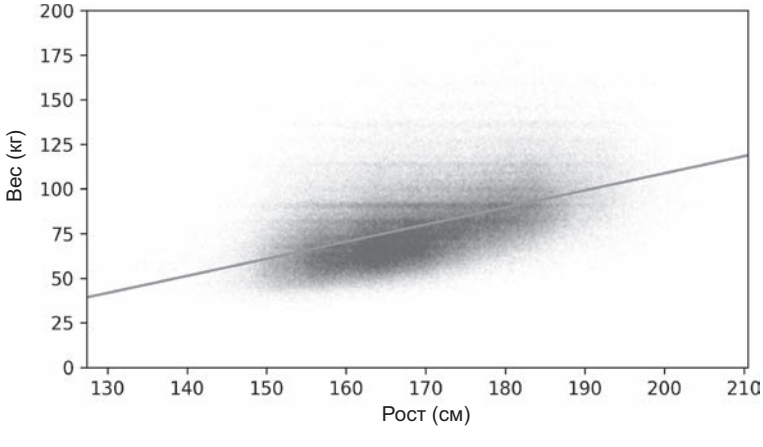
jittered_heights = jitter(heights, 2)
jittered_weights = jitter(weights, 1.5)
```

Также используем среднее значение и стандартное отклонение величины роста, чтобы подобрать граничные значения на оси x:

```
m, s = heights.mean(), heights.std()
xlim = m - 4 * s, m + 4 * s
ylim = 0, 200
```

Вот диаграмма рассеяния данных после добавления шума вместе с подогнанной линией:

```
plt.scatter(jittered_heights, jittered_weights, alpha=0.01, s=0.1)
plt.plot(fit_xs, fit_ys, color="C1")
decorate(xlabel="Рост (см)", ylabel="Вес(кг)", xlim=xlim, ylim=ylim)
```



Линия регрессии на диаграмме проходит несколько выше области наибольшего скопления точек данных. Это объясняется тем, что вес не подчиняется нормальному распределению. Как вы узнали из раздела «Логнормальное распределение» главы 5 на с. 94, вес взрослых людей обычно описывается логнормальным распределением, которое скошено в сторону бóльших значений. Именно этот перекос смещает подогнанную линию вверх.

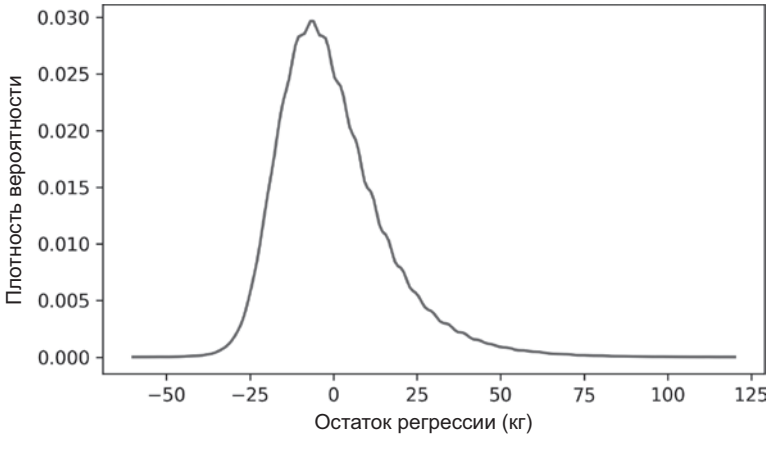
Еще один момент, вызывающий беспокойство, — распределение остатков, которое выглядит так:

```
residuals = compute_residuals(result_brfss, heights, weights)
```

```
from thinkstats import make_pmf

pmf_kde = make_pmf(residuals, -60, 120)
pmf_kde.plot()

decorate(xlabel="Остаток регрессии (кг)", ylabel="Плотность вероятности")
```



Распределение остатков скошено вправо. Сам по себе этот факт может не представлять сложности, но он подразумевает, что методом наименьших квадратов не удалось должным образом описать взаимосвязь между этими переменными.

Если вес описывается логнормальным распределением, то его логарифм должен соответствовать нормальному распределению. Поэтому посмотрим, как изменится результат, если линия регрессии будет выражать логарифм веса как функцию показаний роста:

```
log_weights = np.log10(weights)
result_brfss2 = linregress(heights, log_weights)
result_brfss2.intercept, result_brfss2.slope
```

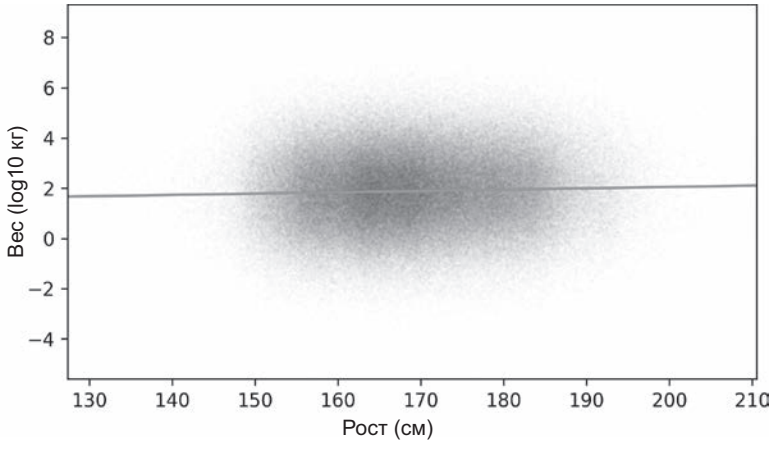
```
(0.9930804163932876, 0.005281454169417777)
```

Поскольку мы преобразовали одну из переменных, коэффициент наклона и свободный член теперь сложнее интерпретировать. Но при помощи `predict` можно найти точки на линии регрессии:

```
fit_xs = np.linspace(heights.min(), heights.max())
fit_ys = predict(result_brfss2, fit_xs)
```

А затем изобразить ее на одном графике с диаграммой рассеяния преобразованных данных:

```
jittered_log_weights = jitter(log_weights, 1.5)
plt.scatter(jittered_heights, jittered_log_weights, alpha=0.01, s=0.1)
plt.plot(fit_xs, fit_ys, color="C1")
decorate(xlabel="Рост (см)", ylabel="Вес (log10 кг)", xlim=xlim)
```

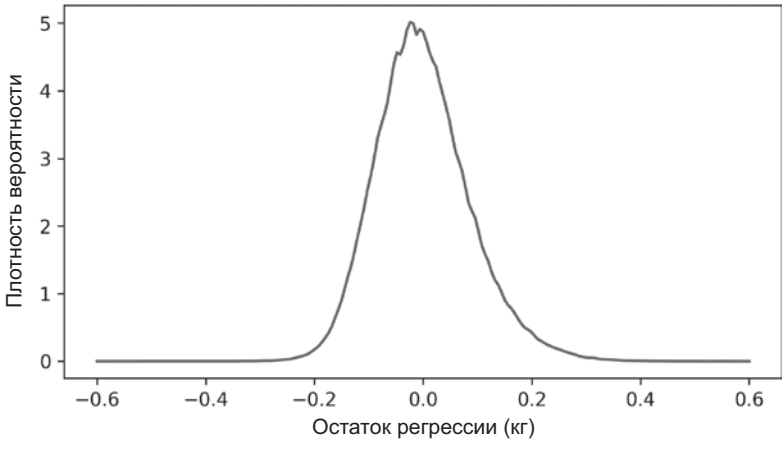


Теперь линия регрессии проходит через область с наибольшей плотностью точек, а сами точки примерно равноудалены по обе стороны от линии. Поэтому распределение остатков приблизительно симметрично:

```
residuals = compute_residuals(result_brfss2, heights, log_weights)
```

```
pmf_kde = make_pmf(residuals, -0.6, 0.6)
pmf_kde.plot()

decorate(xlabel="Остаток регрессии (кг)", ylabel="Плотность вероятности")
```



Внешний вид точечной диаграммы и распределение остатков свидетельствуют, что линия регрессии хорошо описывает связь роста и логарифма веса.

Если сравнить значения r обеих регрессий, то окажется, что коэффициент корреляции роста с логарифмом веса немного больше:

```
result_brfss.rvalue, result_brfss2.rvalue
```

```
(0.5087364789734582, 0.5317282605983435)
```

Это означает, что значение R^2 тоже слегка больше:

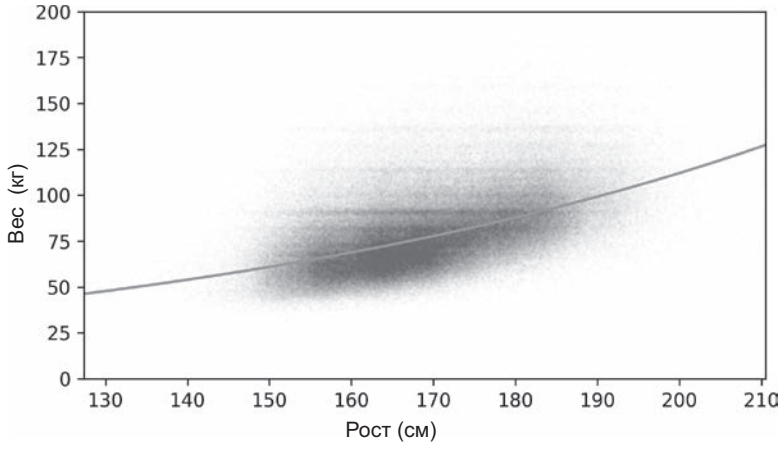
```
result_brfss.rvalue**2, result_brfss2.rvalue**2
```

```
(0.2588128050383119, 0.28273494311893993)
```

При расчете предполагаемых значений веса с помощью показателей роста результаты несколько улучшатся, если работать с логарифмом веса.

Однако преобразование данных усложняет интерпретацию параметров модели — улучшить представление результатов может помочь обратное преобразование. Например, обратным логарифму по основанию 10 является возведение в степень по основанию 10. Вот как выглядит линия регрессии после обратного преобразования вместе с исходными данными:

```
plt.scatter(jittered_heights, jittered_weights, alpha=0.01, s=0.1)
plt.plot(fit_xs, 10**fit_ys, color="C1")
decorate(xlabel="Рост (см)", ylabel="Вес(кг)", xlim=xlim, ylim=ylim)
```



Линия регрессии, выглядящая как прямая в отношении к преобразованным данным, становится изогнутой применительно к исходным данным.

Глоссарий

Модель (model)

Применительно к задаче регрессии модель — это математическое описание взаимосвязи между переменными, например прямая линия, а также описание случайной изменчивости, например нормальное распределение.

Импутация (imputation)

Процесс оценки и заполнения пропущенных значений в датасете.

Линия наилучшего соответствия (line of best fit)

Прямая линия (или кривая), которая наилучшим образом описывает взаимосвязь между переменными в соответствии с некоторым определением «наилучшего».

Линейная регрессия (linear regression)

Метод поиска прямой линии наилучшего соответствия.

Предсказание (prediction)

Точка на линии наилучшего соответствия. В контексте регрессии необязательно относится к будущему.

Остаток (residual)

Разница между наблюдаемым значением и значением, предсказанным с помощью линии наилучшего соответствия.

Линейная регрессия методом наименьших квадратов (linear least squares fit)

Поиск линии, которая минимизирует сумму квадратов остатков.

Коэффициент детерминации (coefficient of determination)

Статистический показатель, обозначаемый как R^2 и часто называемый «R квадрат», который измеряет, насколько хорошо модель соответствует данным.

Бутстрепинг (bootstrap resampling)

Метод ресемплинга (повторной выборки), в котором на основе исходной выборки, рассматриваемой как генеральная совокупность, осуществляется новая выборка того же размера, что и исходная, с возвращением.

Упражнения

Упражнение 10.1

В этой главе мы методом наименьших квадратов находили зависимость веса пингвинов от длины их крыльев. В использованном датасете есть еще два измерения, которые также можно принять во внимание, — длина (culmen length) и глубина (culmen depth) верхней части клюва пингвина.

С помощью метода наименьших квадратов найдите линию регрессии, выражающую зависимость веса от длины верхней части клюва. Постройте диаграмму рассеяния для этих переменных и нанесите на нее подогнанную линию.

Ориентируясь на значение атрибута `rvalue` объекта `RegressionResult`, ответьте на вопросы: какова величина корреляции этих переменных? каков коэффициент детерминации? что лучше предсказывает вес — длина верхней части клюва или длина крыла?

Упражнение 10.2

В этой главе мы использовали ресемплинг, чтобы аппроксимировать выборочное распределение коэффициента наклона линии регрессии. Точно так же можно аппроксимировать выборочное распределение для свободного члена (пересечения оси ординат).

1. Создайте функцию `estimate_intercept`, которая принимает в качестве аргумента объект `DataFrame` — результат ресемплинга, по методу наименьших квадратов находит линейную аппроксимацию веса как функцию длины крыла и возвращает величину свободного члена (intercept).
2. Пройдите в цикле по большому множеству повторных выборок датафрейма `adelie`, вызывая данную функцию, и соберите все значения свободного члена.
3. Используйте функцию `plot_kde` для построения графика выборочного распределения свободного члена.

4. Вычислите стандартную ошибку и 90%-ный доверительный интервал.
5. Убедитесь, что стандартная ошибка, которую вы получаете при повторной выборке, согласуется со значением атрибута `intercept_stderr` объекта `RegressionResult`: она может быть несколько меньше.

Упражнение 10.3

Индекс массы тела, ИМТ (Body Mass Index, BMI), человека равен весу в килограммах, деленному на квадрат роста в метрах. В датасете BRFSS мы можем рассчитать ИМТ после перевода роста из сантиметров в метры:

```
heights_m = heights / 100
bmis = weights / heights_m**2
```

В этом определении рост возводится во вторую, а не какую-либо другую, степень, поскольку на заре истории статистики было обнаружено, что средний вес увеличивается ориентировочно пропорционально квадрату роста.

Чтобы проверить, так ли это, воспользуемся данными из датасета BRFSS, методом наименьших квадратов и математическими приемами. Предположим, что вес пропорционален росту, возведенному в степень с неизвестным показателем a. В этом случае можно записать:

$$w = bh^a,$$

где w — вес, h — рост, а b — неизвестная константа.

Логарифмируя обе части равенства, получаем:

$$\log w = \log b + a \log h.$$

Поэтому, если методом наименьших квадратов вычислить линейную регрессию для логарифма веса как функции логарифма роста, то коэффициент наклона линии регрессии будет давать оценку неизвестного показателя степени a.

Вычислите логарифмы роста и веса. Можно использовать любое основание логарифма, если только оно одинаково для обоих преобразований. Найдите линию регрессии методом наименьших квадратов. Насколько коэффициент наклона близок к 2?

ГЛАВА 11

Множественная регрессия

Линейный метод наименьших квадратов, описанный в предыдущей главе, является примером **регрессии**. В более общем плане регрессия представляет собой задачу моделирования взаимосвязи между одним набором переменных, называемых **зависимыми переменными** (dependent variable) или откликом (response variable), и другим набором переменных, именуемыми **объясняющими переменными** (explanatory variable) или независимыми переменными (independent variable).

В примерах, приведенных в предыдущей главе, есть только одна зависимая переменная и одна объясняющая переменная. Такой случай называется **простой регрессией** (simple regression). В этой главе вы познакомитесь с **множественной регрессией** (multiple regression) с несколькими объясняющими переменными, но по-прежнему только одной переменной-откликом. Случай, когда отклика более одного, представляет собой многомерную регрессию (multivariate regression). Ее мы не будем рассматривать в этой книге.

Модуль StatsModels

В предыдущей главе, чтобы вычислить коэффициенты линии регрессии по методу наименьших квадратов, мы использовали функцию `linregress` библиотеки SciPy. Эта функция считает только простую регрессию. Для множественной регрессии мы будем использовать пакет `StatsModels`, в котором реализованы несколько форм регрессионного и других видов анализа.

В первом примере этой главы продолжим изучение датасета о пингвинах. При загрузке данных зададим столбцам переменных имена без пробелов — это упростит их использование со StatsModels. Для этой цели пригодится следующий словарь:

```
columns = {
    "Body Mass (g)": "mass",
    "Flipper Length (mm)": "flipper_length",
    "Culmen Length (mm)": "culmen_length",
    "Culmen Depth (mm)": "culmen_depth",
}
```

Теперь загрузим данные, удалим строки с отсутствующими значениями массы тела и переименуем столбцы:

```
penguins = (
    pd.read_csv("penguins_raw.csv")
    .dropna(subset=["Body Mass (g)"])
```

```
    .rename(columns=columns)
)
penguins.shape
```

```
(342, 17)
```

Наш датасет содержит данные по трем видам пингвинов. Мы будем работать только с пингвинами Адели:

```
adelie = penguins.query('Species.str.startswith("Adelie")').copy()
len(adelie)
```

```
151
```

В предыдущей главе мы находили линию зависимости веса пингвина от длины его крыльев методом наименьших квадратов:

```
flipper_length = adelie["flipper_length"]
body_mass = adelie["mass"]
```

Напомню, как мы это делали с помощью функции `linregress`:

```
from scipy.stats import linregress

result_linregress = linregress(flipper_length, body_mass)
result_linregress.intercept, result_linregress.slope
```

```
(-2535.8368022002514, 32.83168975115009)
```

Модуль `StatsModels` имеет два типа программных интерфейсов (API). Мы будем использовать «формульный» API, в котором зависимую и объясняющие переменные можно задать с помощью языка формул пакета Patsy. Следующая формульная запись означает, что зависимая переменная `mass` является линейной функцией одной независимой переменной `flipper_length`:

```
formula = "mass ~ flipper_length"
```

Теперь эту формулу вместе с данными можно передать в функцию `ols` модуля `StatsModels`:

```
import statsmodels.formula.api as smf

model = smf.ols(formula, data=adelie)
type(model)
```

```
statsmodels.regression.linear_model.OLS
```

Название `ols` расшифровывается как «обычный метод наименьших квадратов» (ordinary least squares). Слово «обычный» указывает на то, что эта функция реализует метод наименьших квадратов в условиях наиболее распространенного, или «обычного», набора допущений.

Функция возвращает представляющий модель объект `OLS`. В общем случае модель — это упрощенное описание взаимосвязи между переменными. В данном примере это линейная модель, то есть она предполагает, что зависимая переменная является линейной комбинацией объясняющих переменных.

Метод `fit` подгоняет модель под данные и возвращает объект типа `RegressionResults`, содержащий результаты:

```
result_ols = model.fit()
```

Объект `RegressionResults` содержит много информации, поэтому модуль `thinkstats` предоставляет функцию, отображающую только ту информацию, которая нам сейчас понадобится:

```
from thinkstats import display_summary

display_summary(result_ols)
```

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	–2535.8368	–964.798	–2.628	0.009	–4442.291	–629.382
flipper_length	32.8317	5.076	6.468	0.000	22.801	42.862

R-squared: 0.2192

В первом столбце указаны **коэффициенты** модели — свободный член (intercept) и наклон (slope). Можно убедиться, что они совпадают с коэффициентами, полученными из функции `linregress`:

```
result_linregress.intercept, result_linregress.slope
```

```
(-2535.8368022002514, 32.83168975115009)
```

Второй столбец содержит стандартные ошибки коэффициентов. Они также совпадают со значениями, которые мы получили с помощью `linregress`:

```
result_linregress.intercept_stderr, result_linregress.stderr,
```

```
(964.7984274994059, 5.076138407990821)
```

В следующем столбце представлена *t*-статистика, которая используется для вычисления *p*-значений. Мы можем пропустить его, поскольку *p*-значения приведены в следующем столбце с заголовком `P>|t|`. *p*-значение для коэффициента при переменной `flipper_length` округлено до 0, но можно отобразить его так:

```
result_ols.pvalues["flipper_length"]
```

```
1.3432645947789321e-09
```

А затем убедиться, что `linregress` выдала тот же результат:

```
result_linregress.pvalue
```

```
1.3432645947790051e-09
```

При таком маленьком *p*-значении, если допустить отсутствие зависимости между весом и длиной крыла, представляется крайне маловероятным, что угловой коэффициент равен расчетному значению исключительно благодаря случаю.

В последних двух столбцах, обозначенных как `[0.025` и `0.975]`, указан 95%-ный доверительный интервал для свободного члена и наклона. Таким образом, 95 %-ный ДИ для коэффициента наклона составляет [22.8, 42.9].

В последней строке выведено значение R^2 модели — около 0.22. Оно подразумевает, что, если вместо среднего значения веса предсказывать вес с использованием длины крыла, величину MSE можно уменьшить примерно на 22 %.

Получаемое при расчете простой корреляции значение R^2 равно квадрату коэффициента корреляции *r*. Поэтому можно сравнить `rsquared`, вычисленный с помощью функции `ols`, с квадратом `rvalue`, вычисленным функцией `linregress`:

```
result_ols.rsquared, result_linregress.rvalue**2
```

```
(0.21921282646854878, 0.21921282646854875)
```

Если пренебречь небольшой разницей из-за погрешности операций с плавающей точкой, они одинаковы.

Прежде чем переходить к множественной регрессии, рассчитаем еще одну простую регрессию, взяв `culmen_length` (длина верхнего края клюва пингвина) в качестве объясняющей переменной:

```
formula = "mass ~ culmen_length"
result = smf.ols(formula, data=adelie).fit()
display_summary(result)
```

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	34.8830	458.439	0.076	0.939	–870.998	940.764
culmen_length	94.4998	11.790	8.015	0.000	71.202	117.798

R-squared: 0.3013

И опять полученное *p*-значение коэффициента наклона очень мало — случайно наблюдать такую величину наклона в отсутствие связи между массой и длиной клюва крайне маловероятно. Как вы могли заметить, *p*-значение, связанное со свободным членом, велико. Но этот факт нас не должен волновать, поскольку мы не проверяем гипотезу о равенстве свободного члена нулю. У данной модели

свободный член оказался близким к нулю только по стечению обстоятельств — на проблему с моделью он не указывает.

Коэффициент R^2 у этой модели составляет около 0.30. Поэтому если в качестве объясняющей переменной вместо длины крыла использовать длину верхнего края клюва, величина MSE сократится немного больше (значение R^2 для длины крыла равно 0.22). Теперь посмотрим, что получится, если их объединить.

Знакомство с множественной регрессией

Вот формула Patsy для модели множественной регрессии, в которой масса является линейной комбинацией длины плавника и длины верхнего края клюва:

```
formula = "mass ~ flipper_length + culmen_length"
```

А вот результат подгонки этой модели к данным:

```
result = smf.ols(formula, data=adelie).fit()
display_summary(result)
```

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	−3573.0817	866.739	−4.122	0.000	−5285.864	−1860.299
flipper_length	22.7024	4.742	4.787	0.000	13.331	32.074
culmen_length	76.3402	11.644	6.556	0.000	53.331	99.350

R-squared: 0.3949

У этой модели три коэффициента — свободный член и два коэффициента наклона. Наклон для длины крыла равен 22.7. Итак, можно ожидать, что пингвин с крылом, которое длиннее на один миллиметр, будет тяжелее на 22.7 г при условии, что длина верхнего края клюва такая же. Точно так же мы ожидаем, что пингвин с клювом, который длиннее на один миллиметр, будет тяжелее на 76.3 г при условии, что длина крыльев одинакова.

У обоих коэффициентов наклона p-значения малы, это означает, что вклад обеих объясняющих переменных вряд ли случаен.

А коэффициент R^2 равен 0.39 — это выше, чем у модели только с длиной верхней части клюва (0.30) и модели с одной длиной крыла (0.22). Таким образом, предсказания с использованием обеих объясняющих переменных точнее, чем предсказания с помощью только одной из них.

Но не настолько, как можно было ожидать. Если длина крыла уменьшает показатель MSE на 22 %, а длина клюва — на 30 %, то почему бы им обеим вместе не уменьшить его в общей сложности на 52 %? Причина в том, что объясняющие переменные коррелируют друг с другом:

```
from thinkstats import corrcoef

corrcoef(adelie, "flipper_length", "culmen_length")
```

```
0.32578471516515944
```

У пингвинов с более длинными крыльями в среднем также более длинный клюв. Объясняющие переменные содержат информацию друг о друге, а значит, в них присутствует общая информация о зависимой переменной. При добавлении в модель независимой переменной увеличение коэффициента R^2 отражает только появление новой информации, привносимой новой переменной.

Мы увидим ту же картину, если добавим глубину верхнего края клюва (culmen depth) в качестве третьей объясняющей переменной:

```
formula = "mass ~ flipper_length + culmen_length + culmen_depth"
result = smf.ols(formula, data=adelie).fit()
display_summary(result)
```

	coef	std err	t	P > |t|	[0.025	0.975]
Intercept	–4341.3019	795.117	–5.460	0.000	–5912.639	–2769.964
flipper_length	17.4215	4.385	3.973	0.000	8.756	26.087
culmen_length	55.3676	11.133	4.973	0.000	33.366	77.369
culmen_depth	140.8946	24.216	5.818	0.000	93.037	188.752

R-squared: 0.5082

У этой модели четыре коэффициента. Все p-значения малы, а значит, вклад какой-либо независимой переменной вряд ли можно объяснить случайностью. А коэффициент R^2 равен примерно 0.51, что несколько лучше, чем у предыдущей модели с двумя независимыми переменными (0.39), и заметно лучше, чем у любой из моделей с одной переменной (0.22 и 0.30).

Но и в этот раз его увеличение меньше, чем мы могли бы надеяться, потому что величина глубины клюва коррелирует с двумя другими измерениями:

```
[
    corrcoef(adelie, "culmen_depth", "flipper_length"),
    corrcoef(adelie, "culmen_depth", "culmen_length"),
]
```

```
[0.30762017939668534, 0.39149169183587634]
```

Этот пример показывает, как обычно используется множественная регрессия, объединяющая несколько независимых переменных для получения более точных прогнозов. Еще одно ее распространенное применение — количественная оценка вклада одного набора переменных при одновременном контроле вклада другого набора.

Контрольная переменная

Из раздела «Сравнение ИФР» главы 4 на с. 67 вы узнали, что первенцы в среднем легче, чем другие новорожденные. А в разделе «Проверка значимости корреляции» главы 9 на с. 170 убедились, что вес новорожденного коррелирует с возрастом матери: чем старше мать, тем тяжелее в среднем ее дети.

Эти факты могут быть взаимосвязаны. Если матери первенцев моложе матерей других детей, что кажется правдоподобным, то это может объяснять, почему их дети легче. Воспользуемся множественной регрессией, чтобы проверить эту гипотезу, оценивая разницу в весе новорожденного в случае первых и последующих детей и учитывая при этом возраст матери.

Инструкции по загрузке данных NSFG приведены в Jupyter-блокноте этой главы.

Функция `get_nsfg_groups` поможет со считыванием данных, выбором случаев рождения живых детей и разбивкой последних на первенцев и остальных детей:

```
from nsfg import get_nsfg_groups

live, firsts, others = get_nsfg_groups()
```

Используем метод `dropna` для отбора строк с корректными значениями веса при рождении, порядка рождения и возраста матери:

```
valid = live.dropna(subset=["agepreg", "birthord", "totalwgt_lb"]).copy()
```

Теперь задействуем функционал модуля `StatsModels`, чтобы подтвердить, что вес новорожденного коррелирует с возрастом матери, и оценить коэффициент наклона в предположении, что эта зависимость линейна:

```
formula = "totalwgt_lb ~ agepreg"
result_age = smf.ols(formula, data=valid).fit()
display_summary(result_age)
```

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	6.8304	0.068	100.470	0.000	6.697	6.964
agepreg	0.0175	0.003	6.559	0.000	0.012	0.023

R-squared: 0.004738

Коэффициент наклона мал — всего 0.0175 фунта[1] на год. Поэтому, если две матери различаются в возрасте на десять лет, можно ожидать, что вес их детей

[1] Примерно 8 граммов. — *Примеч. пер.*

будет отличаться на 0.175 фунта. Но p-значение тоже мало, так что этот наклон, каким бы малым он ни был, был бы маловероятен, если бы на самом деле не было никакой взаимосвязи.

Значение R^2 также мало, а значит, возраст матери не очень полезен в качестве предсказывающей переменной. Если мы узнаем возраст матери, наша способность предсказать вес ребенка практически не улучшится.

Такое сочетание малых величин p-значения и коэффициента R^2 часто сбивает с толку, поскольку кажется противоречивым: если связь статистически значима, то кажется, что она должна обладать предсказательной силой. Но данный пример показывает, что противоречия нет: взаимосвязь может быть статистически значимой, но малополезной для целей предсказания. Если мы визуализируем результаты, то увидим почему. Сначала выберем соответствующие столбцы:

```
totalwgt = valid["totalwgt_lb"]
agepreg = valid["agepreg"]
```

Чтобы посчитать координаты линии регрессии, можно было бы извлечь величины свободного члена и коэффициента наклона из `result_age`, но в этом нет необходимости. Объект `RegressionResults` предоставляет метод `predict`, который можно использовать взамен. Сначала сгенерируем массив значений для переменной `agepreg`:

```
agepreg_range = np.linspace(agepreg.min(), agepreg.max())
```

Чтобы воспользоваться методом `predict`, нужно поместить значения объясняющих переменных в объект `DataFrame`:

```
df = pd.DataFrame({"agepreg": agepreg_range})
```

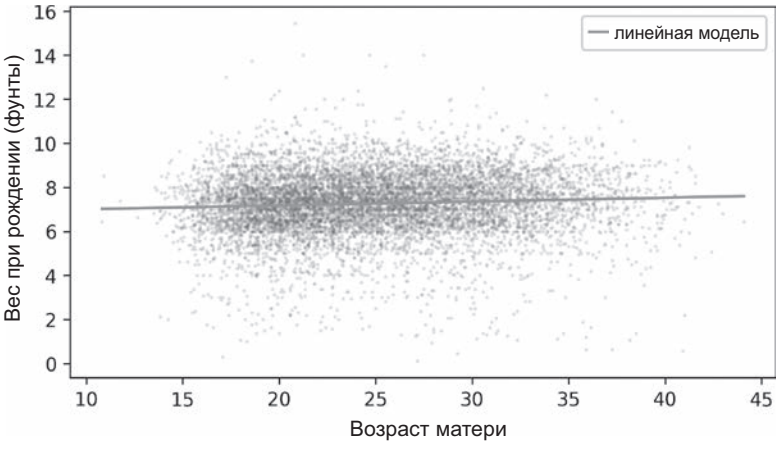
Столбцы в датафрейме должны иметь те же имена, что и объясняющие переменные. Теперь его можно передать методу `predict`:

```
fit_ys = result_age.predict(df)
```

Метод возвращает объект `Series` с предсказанными значениями. Вот как они выглядят вместе с диаграммой рассеяния данных:

```
plt.scatter(agepreg, totalwgt, marker=".", alpha=0.1, s=5)
plt.plot(agepreg_range, fit_ys, color="C1", label="линейная модель")

decorate(xlabel="Возраст матери", ylabel="Вес при рождении (фунты)")
```



Поскольку наклон линии регрессии мал, разница в ожидаемом весе новорожденных у матерей самого юного и самого старшего возраста практически незаметна. Разброс значений веса новорожденного при любом возрасте матери существенно больше.

Теперь с помощью модуля `StatsModels` подтвердим, что первенцы при рождении весят меньше, чем последующие дети. Для этого сначала создадим объект `Series` с логическими значениями `True` для первых детей и `False` для всех остальных и добавим в него новый столбец с именем `is_first`:

```
valid["is_first"] = valid["birthord"] == 1
```

Далее идет формула для модели с весом новорожденного в качестве зависимой переменной и `is_first` в качестве объясняющей переменной. В формульном языке Patsy буква `C` с круглыми скобками вокруг имени переменной указывает на то, что переменная является **категориальной** (categorical), то есть представляющей такие категории, как «первый ребенок», а не измерения, такие как вес новорожденного:

```
formula = "totalwgt_lb ~ C(is_first)"
```

Теперь выполним подгонку (fit) модели и выведем результаты как обычно:

```
result_first = smf.ols(formula, data=valid).fit()
display_summary(result_first)
```

	coef	std err	t	P > |t|	[0.025	0.975]
Intercept	7.3259	0.021	356.007	0.000	7.286	7.366
C(is_first)[T.True]	–0.1248	0.030	–4.212	0.000	–0.183	–0.067

R-squared: 0.00196

В таблице результатов заголовок `C(is_first)[T.True]` указывает на то, что `is_first` является категориальной переменной, а коэффициент связан со значением `True`. Буква `T` перед `True` расшифровывается как «воздействие» (treatment): на языке контролируемого эксперимента первые дети считаются экспериментальной группой (treatment group), а остальные дети — контрольной группой (reference group). Такое соотнесение произвольно: мы могли бы считать, что первенцы являются контрольной группой, а остальные дети — экспериментальной группой. Но для интерпретации результатов нам нужно знать, что есть что.

Свободный член равен примерно 7.3, это означает, что средний вес контрольной группы составляет 7.3 фунта. Коэффициент переменной `is_first` равен –0.12, что означает, что средний вес детей экспериментальной группы, первенцев, на 0.12 фунта меньше. Мы можем проверить оба этих результата, вычислив их напрямую:

```
others["totalwgt_lb"].mean()
```

```
7.325855614973262
```

```
diff_weight = firsts["totalwgt_lb"].mean() - others["totalwgt_lb"].mean()
diff_weight
```

```
-0.12476118453549034
```

В дополнение к этим коэффициентам `StatsModels` вычисляет *p*-значения, доверительные интервалы и коэффициент R^2. Величина *p*-значения, связанного с первенцами, мала, а значит, разница между двумя группами статистически значима. А малость R^2 в свою очередь показывает, что при попытке угадать вес ребенка знание того, первый ли это ребенок, не очень-то помогает.

Теперь посмотрим, насколько правдоподобно, что разница в весе новорожденных обусловлена разницей в возрасте матерей. В среднем матери первенцев примерно на 3.6 года моложе матерей других детей:

```
diff_age = firsts["agepreg"].mean() - others["agepreg"].mean()
diff_age
```

```
-3.5864347661500275
```

А угол наклона линии регрессии для веса новорожденного как функции возраста матери составляет 0.0175 фунта на год:

```
slope = result_age.params["agepreg"]
slope
```

```
0.017453851471802638
```

Если мы умножим величину наклона на разницу в возрасте, то получим ожидаемое различие в весе при рождении первенцев и других детей, обусловленное возрастом матери:

```
slope * diff_age
```

```
-0.0625970997216918
```

Результат равен 0.063 фунта, что составляет примерно половину наблюдаемой разницы. Таким образом, кажется, что наблюдаемую разницу в весе новорожденных можно только частично объяснить различием в возрасте матерей.

Используя множественную регрессию, можно одновременно оценить коэффициенты для возраста матери и категориальной переменной «первый ребенок»:

```
formula = "totalwgt_lb ~ agepreg + C(is_first)"
result = smf.ols(formula, data=valid).fit()
display_summary(result)
```

	coef	std err	t	P > |t|	[0.025	0.975]
Intercept	6.9142	0.078	89.073	0.000	6.762	7.066
C(is_first)[T.True]	–0.0698	0.031	–2.236	0.025	–0.131	–0.009
agepreg	0.0154	0.003	5.499	0.000	0.010	0.021

R-squared: 0.005289

Величина коэффициента при переменной `is_first`, равная –0.0698, означает, что после учета разницы в возрасте матерей первые дети в среднем на 0.0698 фунта легче остальных. Это примерно половина той разницы, которую мы получили без учета возраста матерей.

А *p*-значение равно 0.025, что по-прежнему считается статистически значимым, но находится в пограничном диапазоне, где нельзя исключить вероятность того, что разница такого размера могла возникнуть случайно.

Поскольку эта модель учитывает изменение веса младенца в зависимости от возраста матери, можно сказать, что она **контролирует** влияние возраста матери. Но в ней предполагается линейность зависимости между весом и возрастом матери. Посмотрим, такова ли она на самом деле.

Нелинейная зависимость

Чтобы проверить, может ли вклад переменной `agepreg` быть нелинейным, добавим в датасет новый столбец со значениями `agepreg`, возведенными в квадрат:

```
valid["agepreg2"] = valid["agepreg"] ** 2
```

Теперь мы можем определить модель, включающую в себя и линейную, *и* квадратичную зависимости:

```
formula = "totalwgt_lb ~ agepreg + agepreg2"
```

Выполним подгонку модели как обычно:

```
result_age2 = smf.ols(formula, data=valid).fit()
display_summary(result_age2)
```

	coef	std err	t	P > |t|	[0.025	0.975]
Intercept	5.5720	0.275	20.226	0.000	5.032	6.112
agepreg	0.1186	0.022	5.485	0.000	0.076	0.161
agepreg2	–0.0019	0.000	–4.714	0.000	–0.003	–0.001

R-squared: 0.00718

Связанное с членом второй степени, `agepreg2`, *p*-значение очень мало, что свидетельствует о том, что его вклад в информацию о весе новорожденного больше, чем могла бы давать чистая случайность. А коэффициент R^2 этой модели, равный 0.0072, выше, чем у линейной модели (0.0047).

Оценивая коэффициенты для `agepreg` и `agepreg2`, мы фактически подгоняем параболу к данным. Чтобы в этом убедиться, можно воспользоваться объектом `RegressionResults` для генерации прогнозов для диапазона значений возраста матери.

Сначала создадим временный объект `DataFrame`, столбцы `agepreg` и `agepreg2` которого заполнены с использованием массива диапазона возраста `agepreg_range`:

```
df = pd.DataFrame({"agepreg": agepreg_range})
df["agepreg2"] = df["agepreg"] ** 2
```

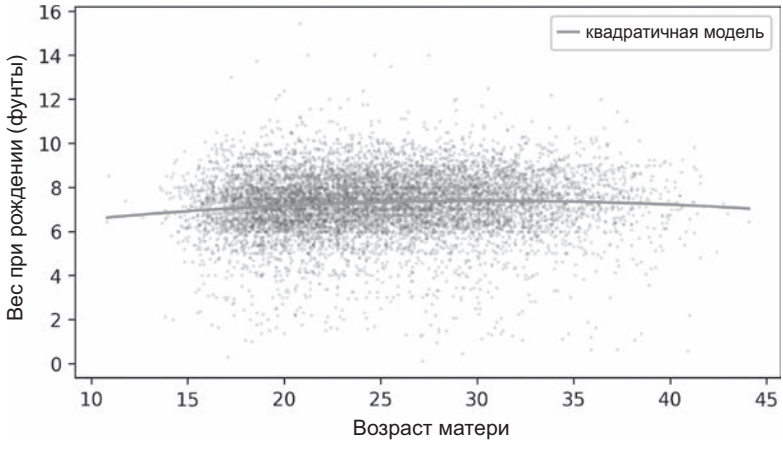
Теперь вызовем метод `predict`, передав ему этот датафрейм в качестве аргумента и получая на выходе объект `Series` с предсказаниями:

```
fit_ys = result_age2.predict(df)
```

Вот как выглядит подогнанная парабола вместе с диаграммой рассеяния данных:

```
plt.scatter(agepreg, totalwgt, marker=".", alpha=0.1, s=5)
plt.plot(agepreg_range, fit_ys, color="C1", label="квадратичная модель")

decorate(xlabel="Возраст матери", ylabel="Вес при рождении (фунты)")
```



Хотя кривизна едва различима, ее наличие говорит о том, что вес новорожденного ниже у самых юных и самых возрастных матерей и выше у рожениц из середины возрастного диапазона.

Квадратичная модель лучше, чем линейная, отражает взаимосвязь между этими переменными. Это говорит о том, что она может эффективнее учитывать разницу в весе при рождении в зависимости от возраста матери. Теперь посмотрим, что произойдет, если к квадратичной модели добавить переменную `is_first`:

```
formula = "totalwgt_lb ~ agepreg + agepreg2 + C(is_first)"
result = smf.ols(formula, data=valid).fit()
display_summary(result)
```

	coef	std err	t	P > |t|	[0.025	0.975]
Intercept	5.6923	0.286	19.937	0.000	5.133	6.252
C(is_first)[T.True]	–0.0504	0.031	–1.602	0.109	–0.112	0.011
agepreg	0.1124	0.022	5.113	0.000	0.069	0.155
agepreg2	–0.0018	0.000	–4.447	0.000	–0.003	–0.001

R-squared: 0.007462

При повышении эффективности контроля возраста матери оценочная разница между первыми и остальными детьми стала равна 0.0504 фунта — меньше, чем при использовании только линейной модели (0.0698 фунта). А *p*-значение, связанное с `is_first`, стало равно 0.109, то есть остаток разницы между этими группами может быть обусловлен случайностью.

Мы приходим к выводу, что разница в весе новорожденных объясняется — по крайней мере частично, а возможно и полностью — различием в возрасте матери.

Логистическая регрессия

Линейная регрессия по своей сути представляет модель, в которой ожидаемое значение зависимой переменной равно взвешенной сумме объясняющих переменных и свободного члена. Такая модель уместна, когда зависимая переменная является непрерывной величиной, такой как вес новорожденного или масса пингвина, но не когда она представляет собой дискретную величину, такую как натуральное число или категория.

Для такого типа зависимых переменных подходят **обобщенные линейные модели**, ОЛМ (generalized linear model, GLM). Например:

- Если зависимая переменная — натуральное число, можно использовать регрессию Пуассона.
- Если зависимая переменная категориальная и категорий только две, можно применить логистическую регрессию.
- Если зависимая переменная категориальная, а категорий больше двух, можно задействовать мультиномиальную логистическую регрессию.
- Если зависимая переменная категориальная и категории могут быть упорядочены, то подойдет порядковая логистическая регрессия.

В книге мы не будем разбирать все эти виды — только **логистическую регрессию**, которая используется наиболее широко. Для примера снова воспользуемся датасетом о пингвинах и посмотрим, можно ли на основе веса и размеров пингвина определить его пол.

В модуле `StatsModels` реализована функция, которая считает логистическую регрессию. Она называется `logit`, потому что математическая функция с таким названием фигурирует в определении логистической регрессии. Прежде чем мы сможем использовать функцию `logit`, необходимо преобразовать зависимую переменную так, чтобы ее значения были равны `0` и `1`[1]:

```
adelie["y"] = (adelie["Sex"] == "MALE").astype(int)
```

Начнем с простой модели с `y` в качестве зависимой переменной и `mass` в качестве объясняющей. Определим и подгоним ее под данные (аргумент `disp=False` подавляет сообщения о процессе подгонки):

```
model = smf.logit("y ~ mass", data=adelie)
result = model.fit(disp=False)
```

Вот что получается:

```
display_summary(result)
```

[1] Из кода далее ясно, что 1 соответствует самцам (male). — *Примеч. пер.*

	coef	std err	z	P > |z|	[0.025	0.975]
Intercept	–25.9871	4.221	–6.156	0.000	–34.261	–17.713
mass	0.0070	0.001	6.138	0.000	0.005	0.009

Pseudo R-squared: 0.5264

Коэффициент детерминации R^2 неприменим к логистической регрессии, но существует несколько альтернативных вариантов, используемых в качестве «значений псевдо-R^2». Величина псевдо-R^2 для этой модели составляет около 0.526. Оно само по себе сложно интерпретируемо, но в дальнейшем понадобится для сопоставления моделей.

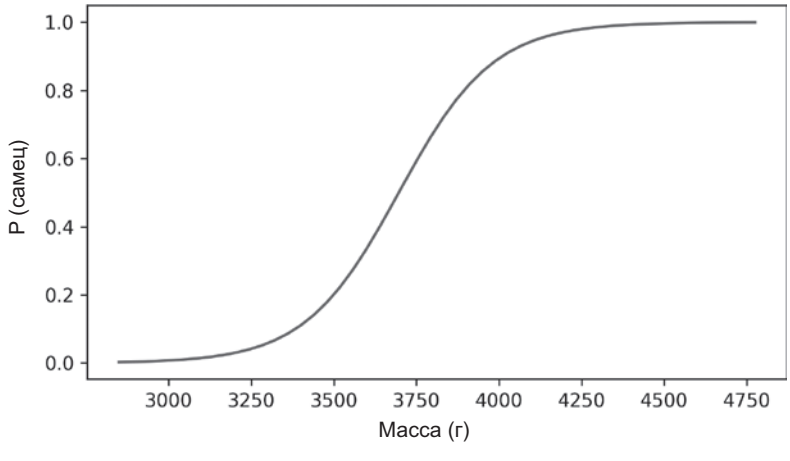
Коэффициент при переменной `mass` положителен, это означает, что более тяжелые пингвины, вероятнее всего, самцы. В остальном эти коэффициенты нелегко интерпретировать. Мы сможем лучше понять эту модель, если построим графики предсказанных значений. Для этого создадим объект `DataFrame` с диапазоном значений переменной `mass` и воспользуемся методом `predict`, чтобы получить объект `Series` с предсказаниями модели:

```
mass = adelie["mass"]
mass_range = np.linspace(mass.min(), mass.max())
df = pd.DataFrame({"mass": mass_range})
fit_ys = result.predict(df)
```

Каждое предсказанное значение как функция веса пингвина — вероятность того, что он самец. Вот график предсказанных значений:

```
plt.plot(mass_range, fit_ys)

decorate(xlabel="Масса (г)", ylabel="P(самец)")
```



Самые легкие пингвины почти наверняка самки, а самые тяжелые, скорее всего, самцы. Пингвин со средним весом 3750 г примерно равновероятно может быть самцом или самкой.

Теперь посмотрим, что нам даст добавление других измерений в качестве объясняющих переменных:

```
formula = "y ~ mass + flipper_length + culmen_length + culmen_depth"
model = smf.logit(formula, data=adelie)
result = model.fit(disp=False)
display_summary(result)
```

	coef	std err	z	P > |z|	[0.025	0.975]
Intercept	–60.6075	13.793	–4.394	0.000	–87.642	–33.573
mass	0.0059	0.001	4.153	0.000	0.003	0.009
flipper_length	–0.0209	0.052	–0.403	0.687	–0.123	0.081
culmen_length	0.6208	0.176	3.536	0.000	0.277	0.965
culmen_depth	1.0111	0.349	2.896	0.004	0.327	1.695

Pseudo R-squared: 0.6622

Значение псевдо-R^2 у этой модели равно 0.662 — выше, чем у предыдущей модели (0.526). Значит, в добавленных измерениях содержится дополнительная информация, помогающая отличить самцов от самок пингвинов.

Небольшая величина p-значений для длины и глубины клюва пингвина указывает на то, что их информационный вклад больше, чем если бы он был случайным. В то же время высокое p-значение для длины крыльев говорит, что если известен вес и размер клюва пингвина, то длина крыльев не добавляет информации.

Чтобы лучше понять эту модель, посмотрим на ее предсказания. Воспользуемся следующей функцией, принимающей на вход последовательность значений массы и одно конкретное значение переменной `culmen_length`. Установив остальные измерения в их средние значения, функция вычисляет предсказания вероятности в зависимости от массы тела и выводит результаты в виде графика:

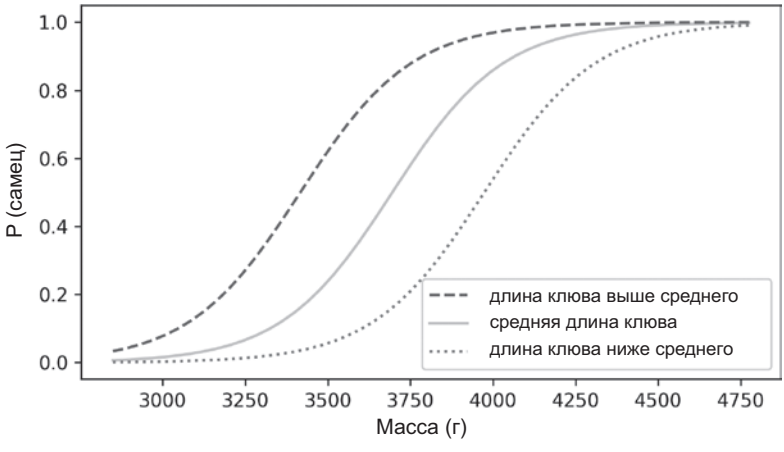
```
def plot_predictions(mass_range, culmen_length, **options):
    """Построить график предсказанной вероятности в зависимости от массы."""
    df = pd.DataFrame({"mass": mass_range})
    df["flipper_length"] = adelie["flipper_length"].mean()
    df["culmen_length"] = culmen_length
    df["culmen_depth"] = adelie["culmen_depth"].mean()
    fit_ys = result.predict(df)
    plt.plot(mass_range, fit_ys, **options)
```

Вот как выглядят результаты для трех значений `culmen_length` — одного стандартного отклонения выше среднего, среднего значения и одного стандартного отклонения ниже среднего:

```
culmen_length = adelie["culmen_length"]
m, s = culmen_length.mean(), culmen_length.std()
```

```
plot_predictions(mass_range, m + s, ls="--", label="Длина клюва выше среднего")
plot_predictions(mass_range, m, alpha=0.5, label="Средняя длина клюва")
plot_predictions(mass_range, m - s, ls=":", label="Длина клюва ниже среднего")

decorate(xlabel="Масса (г)", ylabel="P(самец)")
```



Как уже показала более простая модель, более тяжелые пингвины — с большей вероятностью самцы. Кроме того, у пингвина с более длинным клювом при любом весе вероятность оказаться самцом выше.

Эта модель оказывается более ценной, чем могло изначально показаться. В действительности она похожа на модели, описанные в оригинальной научной статье, для которой был собран этот датасет. Основной темой того исследования был половой диморфизм, то есть степень различия тел самцов и самок. Один из способов количественной оценки диморфизма — использовать измерения для классификации самцов и самок. Стоит ожидать, что у вида с более выраженным диморфизмом такая классификация даст больший процент верных результатов.

Чтобы опробовать эту методику, применим эту же модель к пингвинам другого вида. Помимо пингвина Адели, с которым мы работали до сих пор, датасет содержит измерения для 123 особей субантарктического пингвина (Gentoo penguin). Выделим их, воспользовавшись следующей функцией:

```
def get_species(penguins, species):
    df = penguins.query(f'Species.str.startswith("{species}")').copy()
    df["y"] = (df["Sex"] == "MALE").astype(int)
    return df
```

```
gentoo = get_species(penguins, "Gentoo")
len(gentoo)
```

```
123
```

Вот результаты работы модели логистической регрессии:

```
formula = "y ~ mass + flipper_length + culmen_length + culmen_depth"
model = smf.logit(formula, data=gentoo)
result = model.fit(disp=False)
display_summary(result)
```

	coef	std err	z	P > |z|	[0.025	0.975]
Intercept	–173.9123	62.326	–2.790	0.005	–296.069	–51.756
mass	0.0105	0.004	2.948	0.003	0.004	0.017
flipper_length	0.2839	0.183	1.549	0.121	–0.075	0.643
culmen_length	0.2734	0.285	0.958	0.338	–0.286	0.833
culmen_depth	3.0843	1.291	2.389	0.017	0.554	5.614

Pseudo R-squared: 0.848

Коэффициент псевдо-R^2 составляет 0.848 — больше, чем у пингвинов Адели (0.662). А значит, классификация пингвинов вида субантарктический пингвин с помощью физических измерений даст более верные результаты, чем для пингвинов Адели. Это в свою очередь говорит, что субантарктический пингвин обладает более выраженным половым диморфизмом.

Глоссарий

Регрессия (regression)

Метод оценки коэффициентов, которые подгоняют модель к данным.

Зависимая переменная (response variable)

Переменная, также известная как отклик, которую регрессионная модель пытается предсказать.

Объясняющая переменная (explanatory variable)

Переменная, также известная как независимая переменная, которую модель использует для предсказания зависимых переменных.

Простая регрессия (simple regression)

Регрессия с одной зависимой и одной объясняющей переменной.

Множественная регрессия (multiple regression)

Регрессия с несколькими объясняющими переменными, но только одной зависимой переменной.

Коэффициент (coefficient)

В регрессионной модели коэффициенты — это свободный член (intercept) и коэффициенты наклона (slope) при объясняющих переменных.

Категориальная переменная (categorical variable)

Переменная, которая может принимать одно из множества дискретных значений, обычно нечисловых.

Контрольная переменная (control variable)

Переменная, которая добавляется в модель регрессии, чтобы отделить непосредственный эффект объясняющей переменной от косвенного воздействия.

Обобщенная линейная модель (generalized linear model)

Семейство регрессионных моделей с различными математическими зависимостями между объясняющими переменными и зависимой переменной.

Логистическая регрессия (logistic regression)

Обобщенная линейная модель, используемая в случаях, когда переменная отклика имеет только два возможных значения.

Упражнения

Упражнение 11.1

Весят ли новорожденные мальчики больше девочек? Чтобы ответить на этот вопрос, снова воспользуемся данными NSFG.

Постройте модель линейной регрессии с `totalwgt_lb` в качестве зависимой переменной и `babysex` в качестве категориальной объясняющей переменной: используйте `1` для мальчиков и `2` для девочек. Какова оценочная разница в весе? Является ли она статистически значимой? Если проконтролировать возраст матери, объясняется ли наблюдаемая разница в весе полностью или частично возрастом матери?

Упражнение 11.2

Гипотеза Триверса — Уилларда предполагает, что у многих млекопитающих соотношение полов зависит от «состояния материнской особи», то есть от таких факторов, как возраст, физические параметры, состояние здоровья и социальный статус. Ряд исследований показал наличие этого эффекта у людей, но результаты неоднозначны.

Посмотрим, есть ли связь между возрастом матери и вероятностью рождения мальчика. Постройте модель логистической регрессии, указав пол ребенка в качестве зависимой переменной, а возраст матери — в качестве объясняющей. Шансы матери более старшего возраста родить мальчика повышаются или понижаются? Что получится, если воспользоваться квадратичной моделью возраста матери?

Упражнение 11.3

Для пингвинов Адели постройте модель линейной регрессии, которая предсказывает вес пингвина как функцию переменных `flipper_length`, `culmen_depth`, а также `Sex` в качестве категориальной переменной. Если контролируются длина крыльев и глубина клюва, насколько тяжелее будут самцы пингвинов? Сгенерируйте и постройте диаграмму предсказаний для диапазона длины крыльев у самцов и самок пингвинов, задав значение `culmen_depth` равным среднему значению.

Упражнение 11.4

Проверим, более или менее диморфен антарктический пингвин (Chinstrap penguin) по сравнению с другими видами, представленными в датасете. Об этом можно судить по коэффициенту псевдо-R^2 модели. Используйте функцию `get_species`, чтобы выбрать пингвинов вида антарктический пингвин, затем постройте модель логистической регрессии с полом в качестве зависимой переменной и всеми четырьмя измерениями в качестве объясняющих переменных. Как значение псевдо-R^2 соотносится с другими моделями?

ГЛАВА 12

Анализ временных рядов

Временной ряд — это последовательность измерений некоторой системы, меняющаяся во времени. Многие инструменты, которые мы использовали в предыдущих главах, такие как регрессия, также могут быть использованы для работы с временны́ми рядами. Но существуют и отдельные методы, особенно полезные для такого рода данных.

Для примера возьмем два датасета — с данными о производстве электроэнергии из возобновляемых источников в США с 2001 по 2024 год и с погодными данными за тот же период. Мы разработаем методы для разделения временного ряда на долгосрочный тренд и повторяющуюся сезонную составляющую. Для моделирования и прогнозирования трендов будем использовать линейную регрессию. Мы также опробуем широко используемую модель для анализа данных временных рядов, которая носит название «интегрированная модель авторегрессии скользящего среднего» (autoregressive integrated moving average), или сокращенно ARIMA.

Данные об электрогенерации

В качестве примера данных временных рядов используем датасет Управления энергетической информации (Energy Information Administration) США. В нем содержатся данные о месячном объеме выработки электроэнергии с использованием возобновляемых источников за период с 2001 по 2024 год. Инструкции по загрузке данных, как обычно, приведены в Jupyter-блокноте этой главы.

После загрузки данные необходимо преобразовать, чтобы привести в удобный для работы формат:

```
elec = (
    pd.read_csv("Net_generation_for_all_sectors.csv", skiprows=4)
    .drop(columns=["units", "source key"])
    .set_index("description")
    .replace("--", np.nan)
    .transpose()
    .astype(float)
)
```

В отформатированном датасете каждый столбец представляет собой последовательность суммарных месячных показателей выработки в гигаватт-часах (ГВт-ч). Вот названия столбцов для разных источников электроэнергии, или «секторов»:

```
elec.columns
```

```
Index(['Net generation for all sectors (Чистая выработка во всех секторах)',
           'United States (США)',
       'United States : all fuels (utility-scale) (США: все виды ископаемого
           топлива (промышленного масштаба))', 'United States : nuclear (США:
           атомная энергетика)',
       'United States : conventional hydroelectric (США: традиционная
           гидроэнергетика)',
       'United States : other renewables (США: другие возобновляемые источники
           энергии)', 'United States : wind (США: ветрогенерация)',
       'United States : all utility-scale solar (США: вся солнечная генерация
           промышленного масштаба)', 'United States : geothermal (США:
           геотермальная генерация)',
       'United States : biomass (США: биомасса)',
       'United States : hydro-electric pumped storage (США: гидроаккумулирующие
           электростанции)',
       'United States : all solar (США: вся солнечная генерация)',
       'United States : small-scale solar photovoltaic (США: малая солнечная
           фотоэлектрическая генерация'],
      dtype='object', name='description')
```

Метки индекса содержат месяц и год. Вот первые двенадцать из них:

```
elec.index[:12]
```

```
Index(['Jan (Январь) 2001', 'Feb (Февраль) 2001', 'Mar (Март) 2001',
       'Apr (Апрель) 2001', 'May (Май) 2001', 'Jun (Июнь) 2001',
              'Jul (Июль) 2001', 'Aug (Август) 2001', 'Sep (Сентябрь) 2001',
       'Oct (Октябрь) 2001', 'Nov (Ноябрь) 2001', 'Dec (Декабрь) 2001'],
      dtype='object')
```

Будет проще работать с этими данными, если заменить строки объектами `Timestamp` библиотеки Pandas. Сгенерируем с помощью функции `date_range` последовательность объектов `Timestamp` начиная с января 2001 года. Частотный код `"ME"`, расшифровывающийся как «конец месяца» (month end), указывает этой функции, что необходимо подставить последний день каждого месяца:

```
elec.index = pd.date_range(start="2001-01", periods=len(elec), freq="ME")
elec.index[:6]
```

```
DatetimeIndex(['2001-01-31', '2001-02-28', '2001-03-31', '2001-04-30',
               '2001-05-31', '2001-06-30'],
              dtype='datetime64[ns]', freq='ME')
```

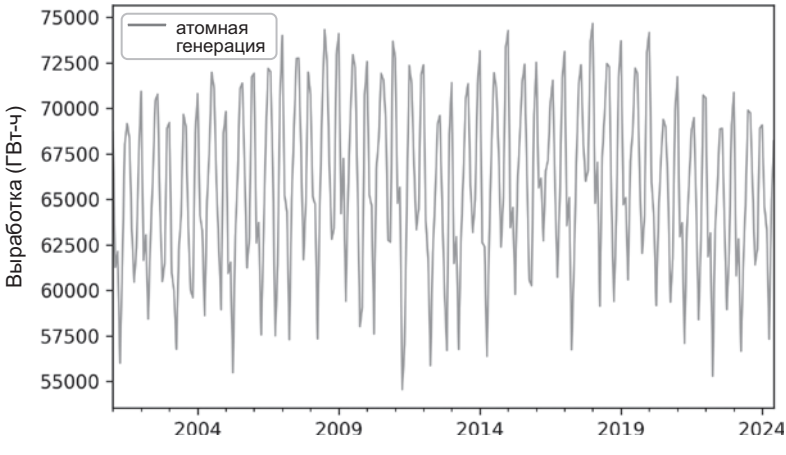
Теперь индекс представляет собой объект `DataTimeIndex` с типом данных `datetime64[ns]`, определенным в библиотеке NumPy. `64` означает, что каждая метка использует 64 бита, а `ns` — что она имеет наносекундную точность.

Декомпозиция

В качестве первого примера рассмотрим, как менялась выработка электроэнергии атомными электростанциями (АЭС) за период с января 2001 года по июнь 2024 года, и разложим временные ряды на долгосрочный тренд и периодическую составляющую. Ниже представлен график общего объема месячной выработки электроэнергии на АЭС в США:

```
nuclear = elec["United States : nuclear"]
nuclear.plot(label="атомная генерация", **actual_options)

decorate(ylabel="Выработка (ГВт-ч)")
```



Видно, что есть подъемы и спады, но их трудно рассмотреть, поскольку наблюдаются большие изменения от месяца к месяцу. Чтобы лучше понять долгосрочный тренд, можно воспользоваться методами `rolling` и `mean` для вычисления **скользящего среднего**:

```
trend = nuclear.rolling(window=12).mean()
```

Параметр размера окна `window=12` задает перекрывающиеся интервалы в 12 месяцев: первый интервал охватывает 12 измерений, начиная с первого, второй интервал — начиная со второго, и т. д. Для каждого интервала мы вычисляем средний объем выработки.

Вот как выглядит кривая тренда вместе с исходными данными:

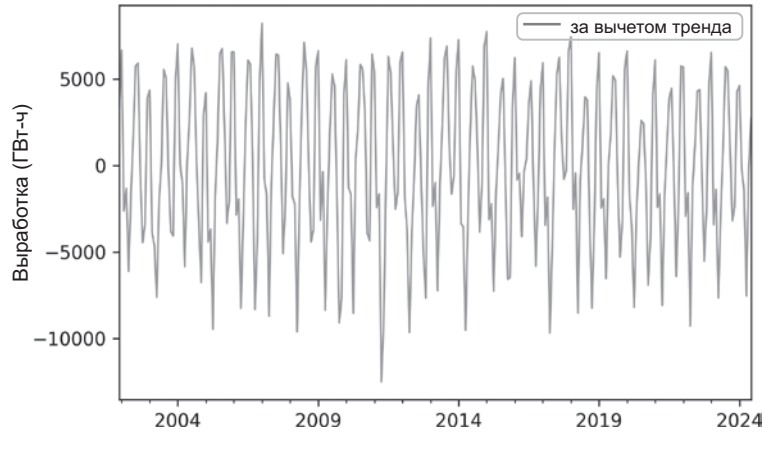
```
nuclear.plot(label="атомная генерация", **actual_options)
trend.plot(label="тренд", **trend_options)
decorate(ylabel="Выработка (ГВт-ч)")
```



Тренд все еще довольно изменчив. Мы могли бы сгладить его еще сильнее, увеличив параметр **window**, но пока оставим 12-месячное окно.

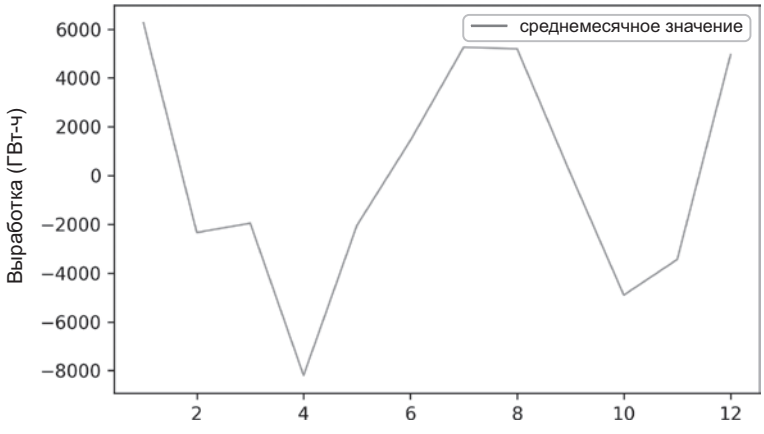
Если вычесть тренд из исходных данных, то в результате получится временной ряд с исключенным трендом (detrended), это подразумевает, что его долгосрочное среднее значение почти является константой. Вот как он выглядит:

```
detrended = (nuclear - trend).dropna()
detrended.plot(label="за вычетом тренда", **actual_options)
decorate(ylabel="Выработка (ГВт-ч)")
```



Похоже, существует повторяющаяся ежегодная закономерность. И в этом есть логика, поскольку спрос на электроэнергию зависит от сезона: зимой она используется для обогрева, а летом — для кондиционирования воздуха. Чтобы описать эту закономерность, можно взять месячную часть объектов `datetime` в индексе, сгруппировать данные по месяцам и вычислить средний объем производства. Вот как выглядят среднемесячные значения:

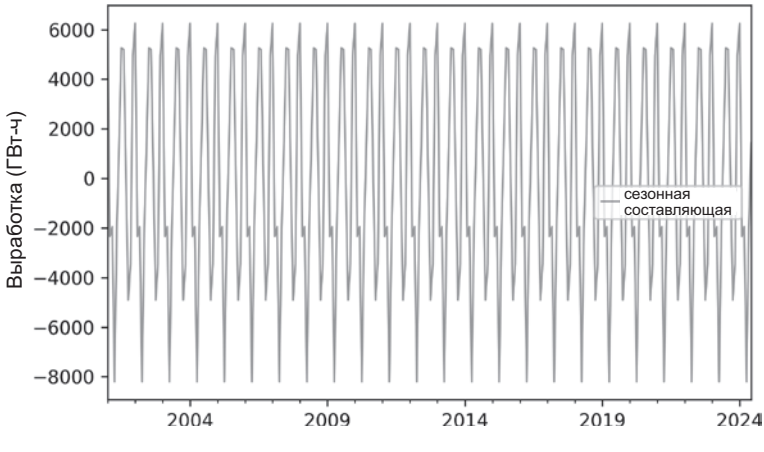
```
monthly_averages = detrended.groupby(detrended.index.month).mean()
monthly_averages.plot(label="среднемесячное значение", **actual_options)
decorate(ylabel="Выработка (ГВт-ч)")
```



На оси x отложены месяцы с января (1) по декабрь (12). Генерация электроэнергии максимальна в самые холодные и самые теплые месяцы года, минимальна в апреле и октябре.

Мы можем использовать среднемесячные значения из `monthly_averages` для вычисления сезонной составляющей данных. Она представляет собой ряд той же длины, что и в переменной `nuclear`. Элементы ряда для каждого месяца равны среднему значению за этот месяц. Вот как выглядит сезонная составляющая:

```
seasonal = monthly_averages[nuclear.index.month]
seasonal.index = nuclear.index
seasonal.plot(label="сезонная составляющая", **actual_options)
decorate(ylabel="Выработка (ГВт-ч)")
```



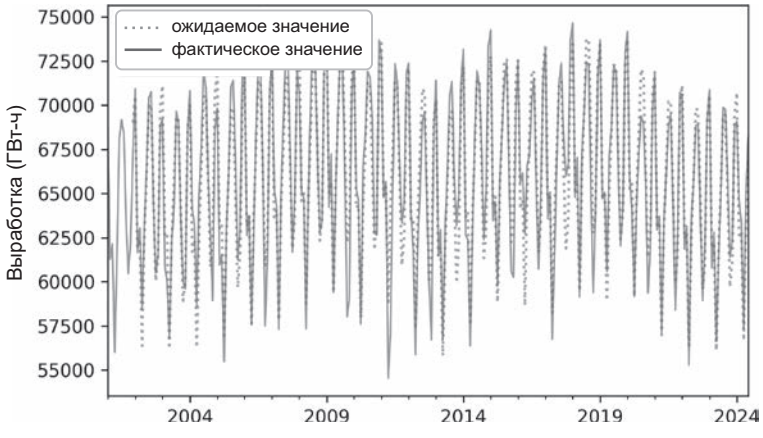
Каждый промежуток в 12 месяцев идентичен остальным.

Сумма тренда и сезонной составляющей представляет собой ожидаемое (expected) значение для каждого месяца:

```
expected = trend + seasonal
```

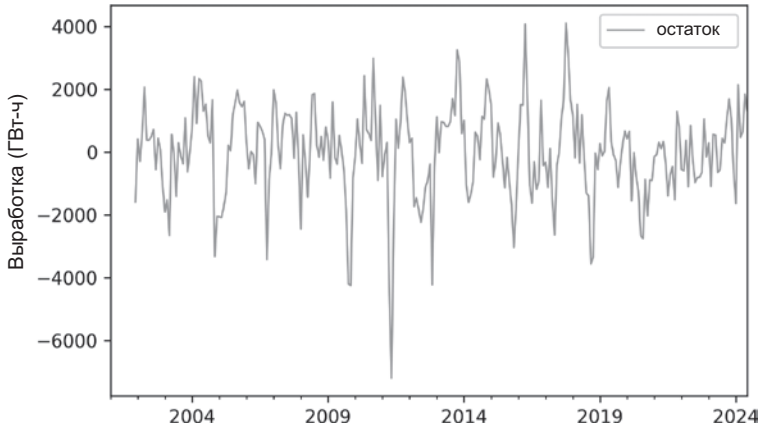
Вот как она выглядит в сравнении с оригинальным рядом:

```
expected.plot(label="ожидаемое значение", **pred_options)
nuclear.plot(label="фактическое значение", **actual_options)
decorate(ylabel="Выработка (ГВт-ч)")
```



Если вычесть эту сумму из исходного ряда, результатом будет остаток (residual), представляющий собой отклонение от ожидаемого значения для каждого месяца:

```
resid = nuclear - expected
resid.plot(label="остаток", **actual_options)
decorate(ylabel="Выработка (ГВт-ч)")
```

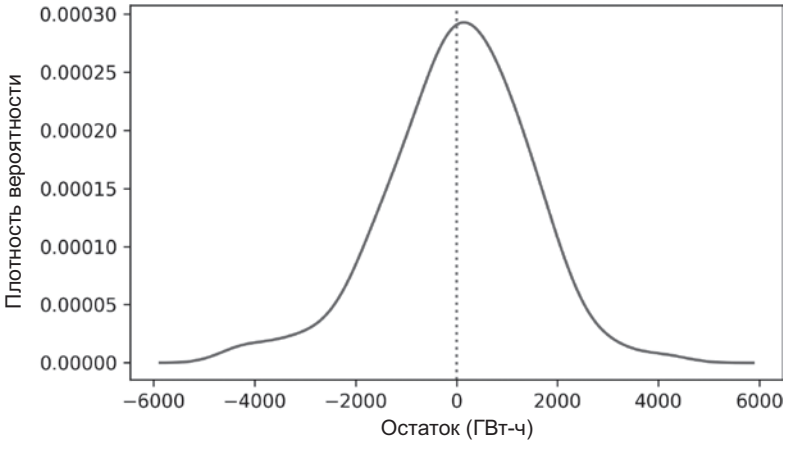


Остаток можно считать суммой всех возможных факторов, которые влияют на производство энергии, но не объясняются долгосрочным трендом или сезонной составляющей. Среди прочего, эта сумма включает погоду, оборудование, находящееся на техобслуживании, и изменение спроса в связи с конкретными событиями. Поскольку остаток — это сумма многих непредсказуемых, а иногда и неизвестных факторов, его часто рассматривают как случайную величину.

Вот как выглядит распределение остатков:

```
from thinkstats import plot_kde

plot_kde(resid.dropna())
decorate(xlabel="Остаток (ГВт-ч)")
```



Оно напоминает колоколообразную кривую нормального распределения, что согласуется с допущением, что остаток является суммой множества случайных вкладов.

Чтобы количественно оценить, насколько хорошо эта модель описывает исходный ряд, вычислим коэффициент детерминации, который показывает, насколько дисперсия остатков меньше по сравнению с дисперсией исходного ряда:

```
rsquared = 1 - resid.var() / nuclear.var()
rsquared
```

```
0.9054559977517084
```

Коэффициент R^2 около 0.91 означает, что на долгосрочный тренд и сезонную составляющую приходится 91 % изменчивости ряда. В данном случае R^2 существенно выше, чем значения, которые нам встретились в предыдущей главе. Это характерно для данных временных рядов, особенно в сценариях, подобных этому, когда мы построили модель так, чтобы она имела сходство с данными.

Процедура, которую мы только что рассмотрели, называется **сезонной декомпозицией**. В пакете StatsModels есть функция `seasonal_decompose`, которая выполняет эти действия:

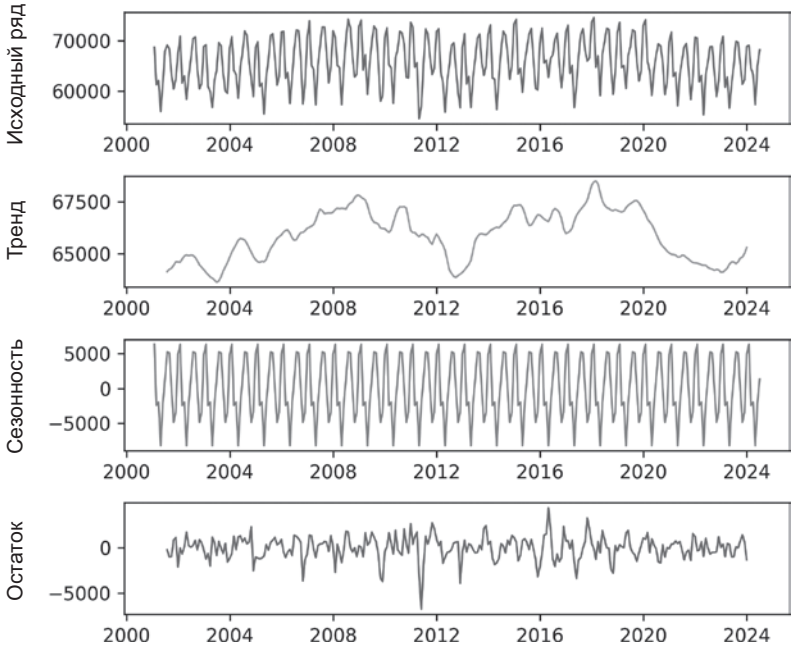
```
from statsmodels.tsa.seasonal import seasonal_decompose

decomposition = seasonal_decompose(nuclear, model="additive", period=12)
```

Аргумент `model="additive"` задает аддитивную модель, поэтому ряд разбивается на сумму тренда, сезонной составляющей и остатка. Вскоре вы также познакомитесь с мультипликативной моделью. Аргумент `period=12` задает продолжительность сезонного компонента в 12 месяцев.

Функция возвращает содержащий эти три составляющих объект. В Jupyter-блокноте этой главы реализована функция, которая строит их графики:

```
plot_decomposition(nuclear, decomposition)
```



Результаты аналогичны тем, что мы получили самостоятельно, а небольшие отличия можно объяснить особенностями реализации.

Такая сезонная декомпозиция позволяет получить представление о структуре временного ряда. Как мы увидим в следующем разделе, она также полезна в построении прогнозов.

Предсказание данных

Результаты сезонной декомпозиции можно использовать для предсказания будущего. В демонстративных целях с помощью функции ниже разделим временной ряд на **обучающий** (training series) — его мы будем использовать для генерации предсказаний — и **тестовый** (test series), который будем использовать для проверки точности предсказаний:

```
def split_series(series, n=60):
    training = series.iloc[:-n]
    test = series.iloc[-n:]
    return training, test
```

При `n=60` длина тестового ряда составляет пять лет, начиная с июля 2019 года:

```
training, test = split_series(nuclear)
test.index[0]
```

```
Timestamp('2019-07-31 00:00:00')
```

Теперь представьте, что сейчас июнь 2019 года и вас просят составить пятилетний прогноз производства электроэнергии на АЭС. Чтобы ответить на этот вопрос, сначала с помощью обучающих данных построим модель, а затем воспользуемся ею для генерации предсказаний. Начнем с сезонной декомпозиции обучающих данных:

```
decomposition = seasonal_decompose(training, model="additive", period=12)
trend = decomposition.trend
```

Теперь применим к тренду линейную модель. Объясняющая переменная `months` равна количеству месяцев от начала ряда:

```
import statsmodels.formula.api as smf

months = np.arange(len(trend))
data = pd.DataFrame({"trend": trend, "months": months}).dropna()
results = smf.ols("trend ~ months", data=data).fit()
```

Вот сводная таблица результатов:

```
from thinkstats import display_summary

display_summary(results)
```

	coef	std err	t	P > | t |	[0.025	0.975]
Intercept	6.482e+04	131.524	492.869	0.000	6.46e+04	6.51e+04
months	10.9886	1.044	10.530	0.000	8.931	13.046

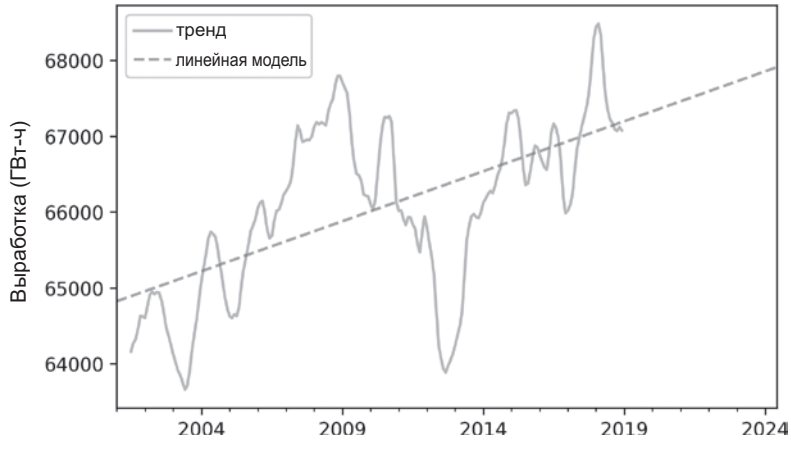
R-squared: 0.3477

Величина коэффициента R^2 около 0.35 свидетельствует, что модель не очень хорошо соответствует данным. Мы сможем лучше в этом разобраться, если построим линию регрессии. Воспользуемся методом `predict` для вычисления ожидаемых значений для обучающих и тестовых данных:

```
months = np.arange(len(training) + len(test))
df = pd.DataFrame({"months": months})
pred_trend = results.predict(df)
pred_trend.index = nuclear.index
```

Вот график тренда и линия регрессии:

```
trend.plot(**trend_options)
pred_trend.plot(label="линейная модель", **model_options)
decorate(ylabel="Выработка (ГВт-ч)")
```



Линейная модель не учитывает многих деталей, но, похоже, в целом наблюдается тенденция к повышению выработки.

Далее воспользуемся сезонной составляющей из результатов декомпозиции, чтобы получить объект `Series` со среднемесячными значениями (monthly averages):

```
seasonal = decomposition.seasonal
monthly_averages = seasonal.groupby(seasonal.index.month).mean()
```

Сезонную составляющую можно предсказать, если по датам линии регрессии выбрать элементы из последовательности `monthly_averages`:

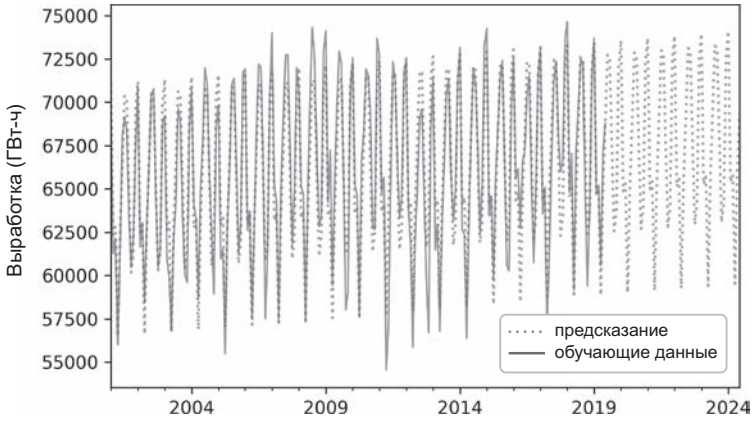
```
pred_seasonal = monthly_averages[pred_trend.index.month]
pred_seasonal.index = pred_trend.index
```

Наконец, чтобы сгенерировать предсказания, добавим к тренду сезонную составляющую:

```
pred = pred_trend + pred_seasonal
```

Вот обучающие данные и полученные предсказания:

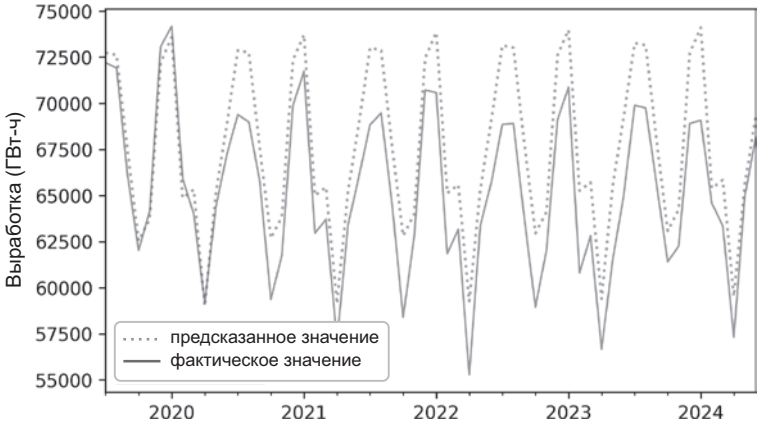
```
pred.plot(label="предсказание", **pred_options)
training.plot(label="обучающие данные", **actual_options)
decorate(ylabel="Выработка (ГВт-ч)")
```



Предсказанные значения достаточно хорошо согласуются с обучающими данными, и прогноз выглядит обоснованным в предположении, что долгосрочный тренд сохранится.

Теперь из позиции в будущем посмотрим, насколько точным оказался наш прогноз. Вот предсказанные и фактические значения за пятилетний период с июля 2019 года:

```
forecast = pred[test.index]
forecast.plot(label="предсказанное значение", **pred_options)
test.plot(label="фактическое значение", **actual_options)
decorate(ylabel="Выработка (ГВт-ч)")
```



Прогноз для первого года довольно неплох, но производство на АЭС в 2020 году оказалось ниже, чем ожидалось — возможно, из-за пандемии COVID-19, — и оно так и не вернулось к уровню долгосрочного тренда.

Чтобы количественно оценить точность предсказаний, используем среднюю абсолютную процентную ошибку (mean absolute percentage error, MAPE), которую вычисляет следующая функция:

```
def MAPE(predicted, actual):
    ape = np.abs(predicted - actual) / actual
    return np.mean(ape) * 100
```

В этом примере предсказания в среднем смещены на 3.81 %:

```
MAPE(forecast, test)
```

```
3.811940747879257
```

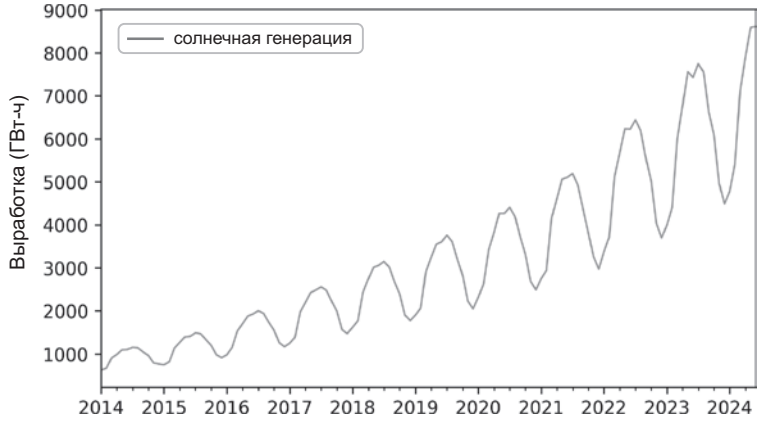
Мы вернемся к этому примеру позже в этой главе и посмотрим, можно ли улучшить результаты с помощью другой модели.

Мультипликативная модель

В аддитивной модели, которую мы использовали в предыдущем разделе, предполагается, что временной ряд представляет собой *сумму* долгосрочного тренда, сезонной составляющей и остатка. Это подразумевает, в свою очередь, что величина сезонной составляющей и остатков не меняется с течением времени.

В качестве примера того, как нарушается это предположение, рассмотрим объемы малой (small-scale) солнечной генерации электроэнергии начиная с 2014 года:

```
solar = elec["United States : small-scale solar photovoltaic"].dropna()
solar.plot(label="солнечная генерация", **actual_options)
decorate(ylabel="Выработка (ГВт-ч)")
```



За этот период общий объем выработки увеличился в несколько раз. Амплитуда сезонных колебаний, очевидно, также возросла.

Предположение о том, что амплитуда сезонных колебаний и случайной изменчивости пропорциональна величине тренда, наводит на мысль о модели, альтернативной аддитивной, в которой временной ряд является *произведением* трех составляющих.

Чтобы опробовать эту мультипликативную модель, разделим этот ряд на обучающий и тестовый наборы:

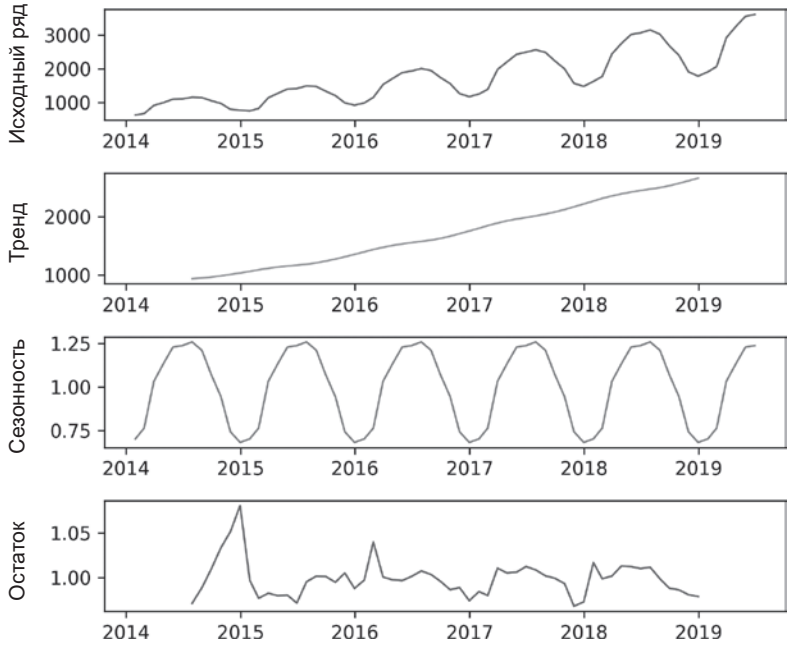
```
training, test = split_series(solar)
```

И вызовем функцию `seasonal_decompose`, указывая аргумент `model="multiplicative"`:

```
decomposition = seasonal_decompose(training, model="multiplicative", period=12)
```

Вот как выглядят полученные составляющие:

```
plot_decomposition(training, decomposition)
```



Теперь сезонная составляющая и остаток являются мультипликативными множителями. И картина такова, что сезонная составляющая колеблется примерно от уровня 25 % ниже тренда до 25 % выше. Независимо от этого остаток обычно

составляет менее 5 %, за исключением небольшого временного отрезка в начальный период. Составляющие модели можно извлечь следующим образом:

```
trend = decomposition.trend
seasonal = decomposition.seasonal
resid = decomposition.resid
```

Коэффициент R^2 у этой модели очень высок:

```
rsquared = 1 - resid.var() / training.var()
rsquared
```

```
0.9999999992978134
```

Солнечная генерация главным образом зависит от количества падающего на панели солнечного света, поэтому вполне логично, что выработка так строго следует годовому циклу. Для предсказания долгосрочного тренда используем квадратичную модель:

```
months = range(len(training))
data = pd.DataFrame({"trend": trend, "months": months}).dropna()
results = smf.ols("trend ~ months + I(months**2)", data=data).fit()
```

В формульной записи пакета Patsy подстрока `I(months**2)` добавляет к модели квадратичный член, поэтому его не нужно вычислять явно. Вот результаты подгонки модели:

```
display_summary(results)
```

	coef	std err	t	P > | t |	[0.025	0.975]
Intercept	766.1962	13.494	56.782	0.000	739.106	793.286
months	22.2153	0.938	23.673	0.000	20.331	24.099
I(months ** 2)	0.1762	0.014	12.480	0.000	0.148	0.205

R-squared: 0.9983

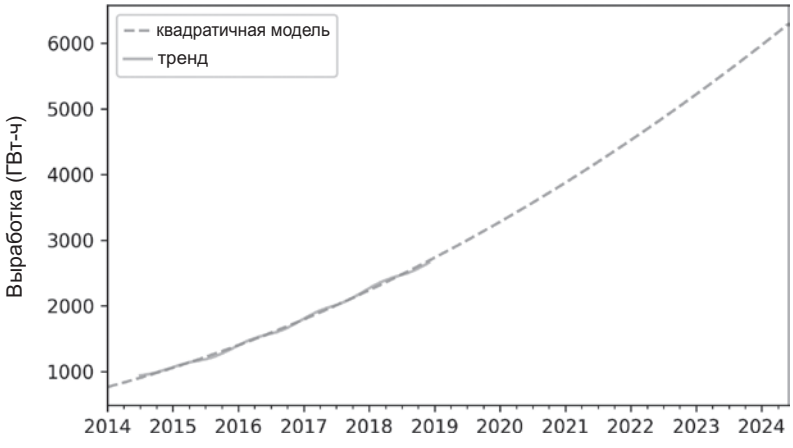
У линейного и квадратичного коэффициентов *p*-значения очень малы, что позволяет предположить, что квадратичная модель отражает больше информации о тренде, чем линейная. Коэффициент R^2 также очень велик.

Теперь с помощью этой модели можно посчитать ожидаемые значения тренда для моментов времени в прошлом и будущем:

```
months = range(len(solar))
df = pd.DataFrame({"months": months})
pred_trend = results.predict(df)
pred_trend.index = solar.index
```

Вот как выглядят полученные результаты:

```
pred_trend.plot(label="квадратичная модель", **model_options)
trend.plot(**trend_options)
decorate(ylabel="Выработка (ГВт-ч)")
```



Квадратичная модель хорошо передает тенденцию в прошлом. Теперь можно использовать сезонную составляющую для предсказания сезонной изменчивости в будущем:

```
monthly_averages = seasonal.groupby(seasonal.index.month).mean()
pred_seasonal = monthly_averages[pred_trend.index.month]
pred_seasonal.index = pred_trend.index
```

Наконец, чтобы рассчитать **ретроспективные оценки** (retrodiction) для значений в прошлом и предсказания для будущего, перемножим тренд и сезонную составляющую:

```
pred = pred_trend * pred_seasonal
```

Вот полученные результаты на одном графике с обучающими данными:

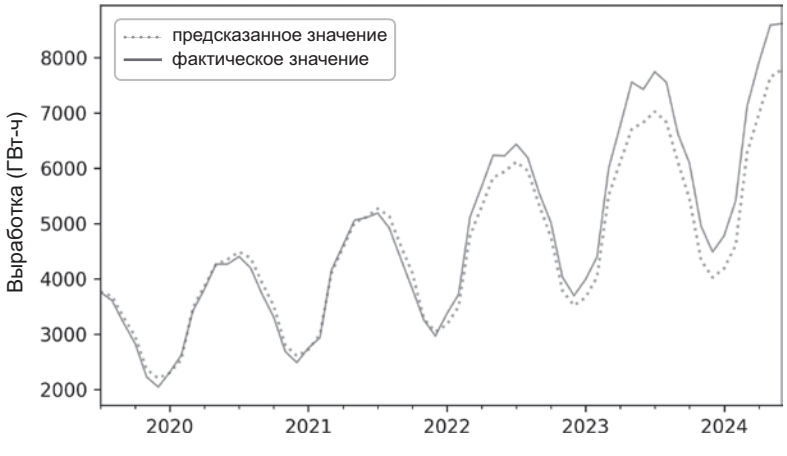
```
training.plot(label="обучающие данные", **actual_options)
pred.plot(label="предсказание", **pred_options)
decorate(ylabel="Выработка (ГВт-ч)")
```



Ретроспективные оценки хорошо согласуются с обучающими данными, а прогноз кажется правдоподобным. Теперь посмотрим, насколько он оказался верным. Вот предсказанные значения вместе с тестовыми данными:

```
future = pred[test.index]

future.plot(label="предсказанное значение", **pred_options)
test.plot(label="фактическое значение", **actual_options)
decorate(ylabel="Выработка (ГВт-ч)")
```



Предсказания для первых трех лет очень хорошо соответствуют данным. В дальнейшем, как видно, фактический рост превысил ожидания.

В этом примере сезонная декомпозиция хорошо показала себя при моделировании и прогнозировании солнечной генерации, но в предыдущем примере с АЭС

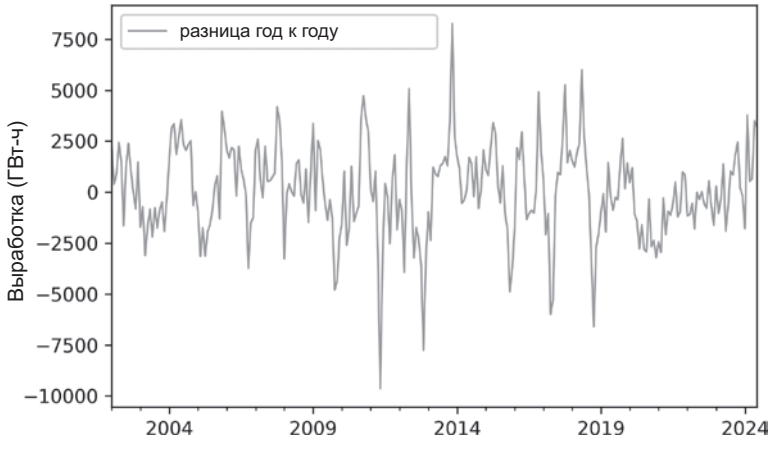
она была не очень эффективна. В следующем разделе мы опробуем другой подход — авторегрессию.

Авторегрессия

Авторегрессия основана на том принципе, что будущее похоже на прошлое. Например, во временных рядах, которые мы рассматривали до сих пор, присутствует четкий годовой цикл. Поэтому при составлении прогноза на июнь следующего года можно ориентироваться на июнь прошлого.

Чтобы проверить, что этот принцип работает, вернемся к ряду `nuclear`, содержащему ежемесячный объем выработки электроэнергии на АЭС США, и посчитаем разность значений за один и тот же месяц в следующих один за другим годах, которая называется разностью «год к году» (year-over-year difference):

```
diff = (nuclear - nuclear.shift(12)).dropna()
diff.plot(label="разница год к году", **actual_options)
decorate(ylabel="Выработка (ГВт-ч)")
```



Величина этой разности существенно меньше значений исходного ряда, что приводит нас к следующему принципу авторегрессии: возможно, будет проще предсказать эту разность, чем исходные значения.

С этой целью посмотрим, существует ли корреляция между последовательными элементами в ряду разностей. Если да, можно использовать эту корреляцию для предсказания последующих значений на основе предыдущих.

Сначала создадим объект `DataFrame`, поместив исходные разности в первый столбец и их же со сдвигом на 1, 2 и 3 месяца — в последующие столбцы. Назовем

последние `lag1`, `lag2` и `lag3` соответственно, поскольку в них содержатся ряды с **лагом** (lag), или временно́й задержкой:

```
df_ar = pd.DataFrame({"diff": diff})
for lag in [1, 2, 3]:
    df_ar[f"lag{lag}"] = diff.shift(lag)

df_ar = df_ar.dropna()
```

Вот значения корреляции между столбцами:

```
df_ar.corr()[["diff"]]
```

	diff
diff	1.000000
lag1	0.562212
lag2	0.292454
lag3	0.222228

Эти корреляции называются запаздывающими корреляциями (lagged correlations) или **автокорреляциями**. Префикс «авто» указывает на то, что мы считаем корреляцию ряда с самим собой. Как частный случай, корреляция между `diff` и `lag1` называется **последовательной корреляцией** (serial correlation), поскольку это корреляция между следующими один за другим элементами в ряду.

Сила этих корреляций достаточна, чтобы предположить, что они могут оказаться полезными для предсказаний. Поэтому включим их в модель множественной регрессии. Функция ниже использует столбцы объекта `DataFrame` для создания формулы в формате Patsy: первый столбец используется в качестве зависимой переменной, а остальные столбцы — в качестве объясняющих переменных:

```
def make_formula(df):
    """Составить формулу Patsy из заголовков столбцов."""
    y = df.columns[0]
    xs = " + ".join(df.columns[1:])
    return f"{y} ~ {xs}"
```

Вот результаты работы линейной модели, предсказывающей следующее значение последовательности на основе трех предыдущих значений:

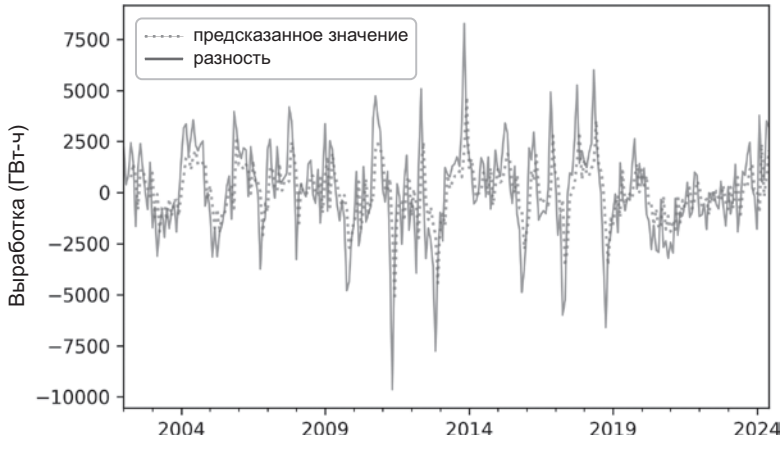
```
formula = make_formula(df_ar)
results_ar = smf.ols(formula=formula, data=df_ar).fit()
display_summary(results_ar)
```

	coef	std err	t	P > |t|	[0.025	0.975]
Intercept	24.2674	114.674	0.212	0.833	–201.528	250.063
lag1	0.5847	0.061	9.528	0.000	0.464	0.706
lag2	–0.0908	0.071	–1.277	0.203	–0.231	0.049
lag3	0.1026	0.062	1.666	0.097	–0.019	0.224

R-squared: 0.3239

Теперь воспользуемся методом `predict` для генерации предсказаний значений ряда в прошлом. Вот как выглядит эта ретроспективная оценка в сравнении с фактическими данными:

```
pred_ar = results_ar.predict(df_ar)
pred_ar.plot(label="предсказанное значение", **pred_options)
diff.plot(label="разность", **actual_options)
decorate(ylabel="Выработка (ГВт-ч)")
```



Предсказанные значения местами близки к фактическим, но коэффициент R^2 всего около 0.319, так что модель оставляет желать лучшего:

```
resid_ar = (diff - pred_ar).dropna()
R2 = 1 - resid_ar.var() / diff.var()
R2
```

```
0.3190252265690783
```

Один из способов улучшить предсказания — вычислить остатки от предсказаний этой модели и использовать другую модель для предсказания этих остатков, что является еще одним принципом авторегрессии.

Скользящее среднее

Представьте, что сейчас июнь 2019 года и вас просят сделать предсказание на июнь 2020-го. В качестве начального приближения можно принять, что значение этого года повторится в следующем году.

Теперь предположим, что сейчас май 2020 года и вас просят пересмотреть ваш прогноз на июнь 2020-го. Вы могли бы использовать результаты за последние три месяца и автокорреляционную модель из предыдущего раздела, чтобы предсказать разность «год к году».

Наконец, допустим, что вы проверяете предсказания последних нескольких месяцев и видите, что все они занижены. Это означает, что предсказание на следующий месяц также может быть слишком низким, поэтому вы, скорее всего, пересмотрите его в сторону повышения. Основная предпосылка здесь в том, что близкие по времени ошибки предсказаний предсказывают будущие ошибки предсказаний.

Чтобы убедиться в этом, создадим объект `DataFrame`, где остатки из модели авторегрессии занимают первый столбец, а запаздывающие версии остатков — остальные столбцы. В этом примере я буду использовать лаг в 1 и 6 месяцев:

```
df_ma = pd.DataFrame({"resid": resid_ar})

for lag in [1, 6]:
    df_ma[f"lag{lag}"] = resid_ar.shift(lag)

df_ma = df_ma.dropna()
```

Воспользуемся классом `ols` для построения модели авторегрессии остатков. Эта часть модели называется **скользящим средним** (moving average), поскольку она уменьшает изменчивость предсказаний способом, аналогичным эффекту скользящего среднего. Мне этот термин не кажется информативным, тем не менее он общеупотребим.

Как бы там ни было, вот сводная таблица модели авторегрессии остатков:

```
formula = make_formula(df_ma)
results_ma = smf.ols(formula=formula, data=df_ma).fit()
display_summary(results_ma)
```

	coef	std err	t	P > |t|	[0.025	0.975]
Intercept	–14.0016	114.697	–0.122	0.903	–239.863	211.860
lag1	0.0014	0.062	0.023	0.982	–0.120	0.123
lag6	–0.1592	0.063	–2.547	0.011	–0.282	–0.036

R-squared: 0.0247

Коэффициент R^2 довольно мал, так что, похоже, эта часть модели не очень подходит. Но малость p-значения для лага в шесть месяцев означает, что эта переменная содержит больше информации, чем случайный вклад.

Теперь с помощью этой модели можно сгенерировать ретроспективные оценки для остатков:

```
pred_ma = results_ma.predict(df_ma)
```

А сейчас, чтобы сгенерировать ретроспективные оценки для разностей год к году, добавим поправку из второй модели к ретроспективным оценкам первой:

```
pred_diff = pred_ar + pred_ma
```

Коэффициент R^2 для суммы двух моделей, равный примерно 0.332, совсем ненамного лучше результата без поправки скользящего среднего (0.319):

```
resid_ma = (diff - pred_diff).dropna()
R2 = 1 - resid_ma.var() / diff.var()
R2
```

```
0.3315101001391231
```

Далее мы с помощью этих разностей год к году будем создавать ретроспективные оценки исходных значений.

Ретроспективная оценка с авторегрессией

Начнем генерировать ретроспективные оценки с того, что поместим значения разности год к году в объект `Series`, в котором индекс соответствует индексу исходных данных:

```
pred_diff = pd.Series(pred_diff, index=nuclear.index)
```

Используя метод `isna` для проверки значений `NaN`, обнаруживаем, что первый 21 элемент нового объекта `Series` отсутствует:

```
n_missing = pred_diff.isna().sum()
n_missing
```

```
21
```

Это объясняется тем, что мы сначала сдвинули временной ряд на 12 месяцев, чтобы рассчитать разность год к году, затем эти разности сдвинули на 3 месяца для первой модели авторегрессии и остатки первой модели — еще на 6 месяцев для второй модели. Каждый раз, когда мы сдвигаем ряд подобным образом, мы теряем несколько значений в начале, а сумма всех этих сдвигов равна 21.

Поэтому, прежде чем мы сможем генерировать ретроспективные оценки, мы должны подготовить наш ряд, скопировав первый 21 элемент из исходного в новый объект `Series`:

```
pred_series = pd.Series(index=nuclear.index, dtype=float)
pred_series.iloc[:n_missing] = nuclear.iloc[:n_missing]
```

Теперь можно исполнить следующий цикл, который заполняет элементы с индекса 21 (22-го элемента) до конца последовательности. Каждый элемент — это сумма значения из предыдущего года и предсказанной разности год к году:

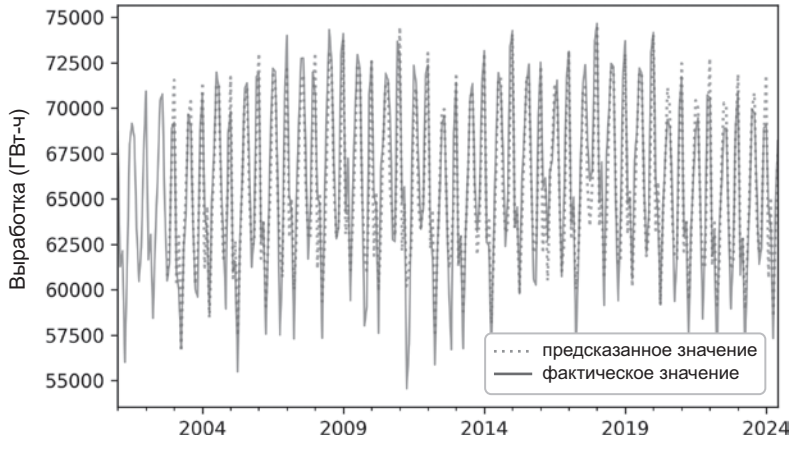
```
for i in range(n_missing, len(pred_series)):
    pred_series.iloc[i] = pred_series.iloc[i - 12] + pred_diff.iloc[i]
```

Теперь заменим скопированные элементы на `NaN`, чтобы не ставить себе в заслугу то, что мы идеально «предсказали» первое 21 значение:

```
pred_series[:n_missing] = np.nan
```

Вот как выглядит ретроспективная оценка в сравнении с оригинальными данными:

```
pred_series.plot(метка="предсказанное значение", **pred_options)
nuclear.plot(label="фактическое значение", **actual_options)
decorate(ylabel="Выработка (ГВт-ч)")
```



Она выглядит довольно неплохо, а коэффициент R^2 равен примерно 0.86:

```
resid = (nuclear - pred_series).dropna()
R2 = 1 - resid.var() / nuclear.var()
R2
```

```
0.8586566911201015
```

Модель, которую мы использовали для расчета этих ретроспективных оценок, называется SARIMA и является одной из моделей семейства ARIMA. Каждая составляющая этих аббревиатур соотносится с определенным элементом модели:

- *S* означает сезонный (seasonal), поскольку первым шагом было вычисление разностей между значениями, разделенными одним периодом сезонности.
- *AR* расшифровывается как авторегрессия (autoregression), которую мы использовали для моделирования запаздывающих корреляций в разностях.
- *I* означает интегрированный (integrated), так как итеративный процесс, который мы использовали для вычисления элементов `pred_series`, аналогичен интегрированию в математическом анализе.
- *MA* расшифровывается как скользящее среднее (moving average) — общепринятое название для второй модели авторегрессии, которую мы применяли к остаткам первой.

Модели ARIMA — это мощные и универсальные инструменты для моделирования данных временных рядов.

ARIMA

Модуль StatsModel содержит библиотеку `tsa`, название которой является сокращением от «анализ временных рядов» (time series analysis). В ней реализована функция `ARIMA`, которая подбирает модели ARIMA и генерирует прогнозы.

Чтобы подогнать модель SARIMA, которую мы строили в предыдущих разделах, вызовем эту функцию с двумя кортежами — `order` и `seasonal_order` — в качестве аргументов. Вот значения в кортеже `order`, соответствующие модели из предыдущих разделов:

```
order = ([1, 2, 3], 0, [1, 6])
```

Значения параметра порядка (`order`) определяют:

- Какие лаги следует включить в модель авторегрессии (AR): первые три числа в данном примере.
- Сколько раз модель должна считать разность между последовательными элементами. В нашем примере это 0, потому что взамен мы вычисляли сезонную разницу, — мы к этому еще вернемся.
- Какие лаги должны быть включены в модель скользящего среднего (MA): в данном примере это один и шесть.

А вот значения параметра сезонного порядка (`seasonal_order`):

```
seasonal_order = (0, 1, 0, 12)
```

Первый и третий элементы равны 0, так как в этой модели отсутствуют и сезонная авторегрессия (AR), и сезонное скользящее среднее (MA). Второй элемент, равный 1, указывает модели считать сезонные разности, а последний элемент задает период сезонности.

Вот как вызвать функцию `ARIMA` для создания и подгонки этой модели:

```
import statsmodels.tsa.api as tsa

model = tsa.ARIMA(nuclear, order=order, seasonal_order=seasonal_order)
results_arima = model.fit()
display_summary(results_arima)
```

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.0458	0.379	0.121	0.904	–0.697	0.788
ar.L2	–0.0035	0.116	–0.030	0.976	–0.230	0.223
ar.L3	0.0375	0.049	0.769	0.442	–0.058	0.133
ma.L1	0.2154	0.382	0.564	0.573	–0.533	0.964
ma.L6	–0.0672	0.019	–3.500	0.000	–0.105	-0.030
sigma2	3.473e+06	1.9e-07	1.83e+13	0.000	3.47e+06	3.47e+06

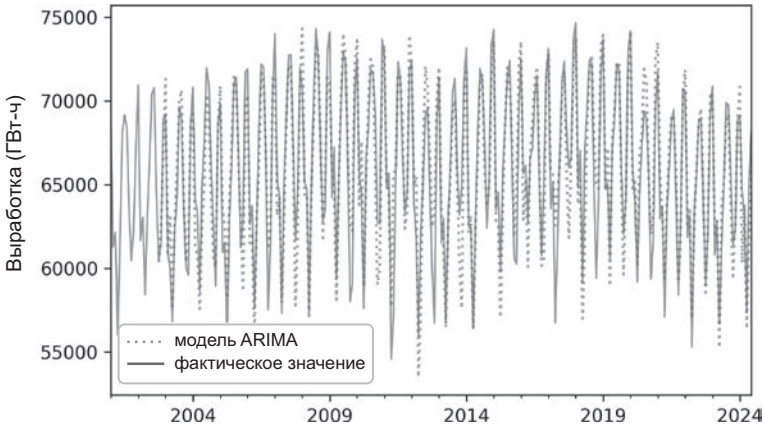
В таблице результатов приведены оценки коэффициентов для трех лагов в модели авторегрессии (AR), двух лагов в модели скользящего среднего (MA) и дисперсия остатков `sigma2`.

Из объекта `results_arima` можно извлечь значение поля `fittedvalues`, в котором содержатся ретроспективные оценки. По той же причине, по которой в начале вычисленного нами ряда отсутствовали значения, в начале `fittedvalues` есть некорректные значения. Удалим их:

```
fittedvalues = results_arima.fittedvalues[n_missing:]
```

Подогнанные (fitted) значения похожи на те, которые мы уже вычислили, но не совпадают в точности — вероятно, из-за того что данная реализация `ARIMA` по-другому обрабатывает начальные условия:

```
fittedvalues.plot(label="модель ARIMA", **pred_options)
nuclear.plot(label="фактическое значение", **actual_options)
decorate(ylabel="Выработка (ГВт-ч)")
```



Коэффициент R^2 тоже близок, но не идентичен:

```
resid = fittedvalues - nuclear
R2 = 1 - resid.var() / nuclear.var()
R2
```

```
0.8262717330784233
```

Функция `ARIMA` позволяет легко экспериментировать с различными вариантами данной модели.

Предсказание с помощью ARIMA

В объекте, возвращаемом функцией `ARIMA`, имеется метод `get_forecast`, который генерирует предсказания. В демонстрационных целях разделим временной ряд на обучающую и тестовую части и применим ту же модель к обучающему ряду:

```
training, test = split_series(nuclear)
model = tsa.ARIMA(training, order=order, seasonal_order=seasonal_order)
results_training = model.fit()
```

Полученный результат можно использовать, чтобы создать прогноз для тестового ряда:

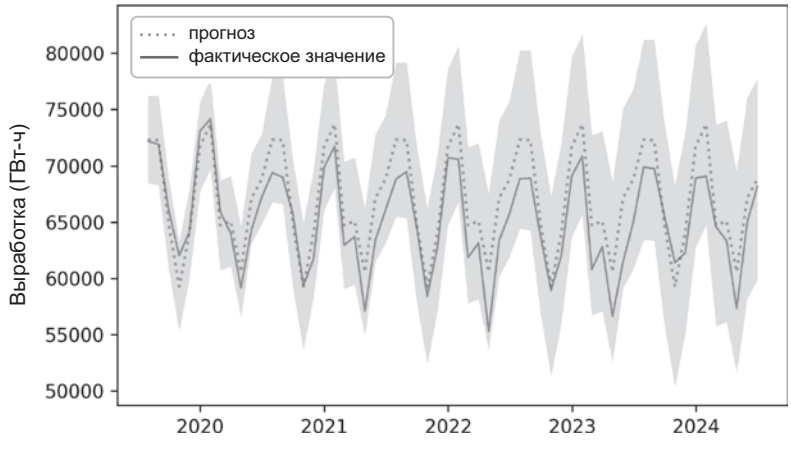
```
forecast = results_training.get_forecast(steps=len(test))
```

У полученного объекта имеется атрибут `forecast_mean`, содержащий массив средних значений, и функция, возвращающая доверительный интервал (confidence interval, ci):

```
forecast_mean = forecast.predicted_mean
forecast_ci = forecast.conf_int()
forecast_ci.columns = ["lower", "upper"]
```

Чтобы представить результаты графически и сравнить их с фактическими данными временного ряда, выполним следующий код:

```
plt.fill_between(
    forecast_ci.index,
    forecast_ci.lower,
    forecast_ci.upper,
    lw=0,
    color="gray",
    alpha=0.2,
)
plt.plot(forecast_mean.index, forecast_mean, label="прогноз", **pred_options)
plt.plot(test.index, test, label="фактическое значение", **actual_options)
decorate(ylabel="Выработка (ГВт-ч)")
```



Фактические значения почти полностью укладываются в доверительный интервал прогнозных значений. Вот величина метрики MAPE для предсказаний модели:

```
MAPE(forecast_mean, test)
```

```
3.381754924564627
```

Предсказания в среднем смещены на 3.38 %, что несколько лучше результатов, полученных в случае сезонной декомпозиции (3.81 %).

Модель ARIMA универсальнее, чем сезонная декомпозиция, и часто позволяет делать более точные предсказания. В данном временном ряду автокорреляция не очень сильна, поэтому выигрыш от использования ARIMA умеренный.

Глоссарий

Временной ряд (time series)

Датасет, в котором каждое значение связано с определенным временем и часто представляет собой измерения, проводимые через регулярные промежутки времени.

Сезонная декомпозиция (seasonal decomposition)

Метод разделения временного ряда на долгосрочный тренд, повторяющуюся сезонную составляющую и остаток.

Обучающий ряд (training series)

Часть временного ряда, используемая для подгонки модели.

Тестовый ряд (test series)

Часть временного ряда, используемая для проверки точности сгенерированных моделью предсказаний.

Ретроспективная оценка (retrodiction)

Предсказание значения, наблюдавшегося в прошлом, часто используемое для тестирования или валидации модели.

Окно (window)

Набор следующих друг за другом значений во временном ряду, используемый для вычисления скользящего среднего.

Скользящее среднее (moving average)

Метод сглаживания колебаний временных рядов, состоящий в усреднении значений в перекрывающихся окнах.

Последовательная корреляция[1] (serial correlation)

Корреляция между следующими друг за другом элементами временного ряда.

Автокорреляция (autocorrelation)

Корреляция между временным рядом и его сдвинутой, или запаздывающей (lagged), версией.

Лаг (lag)

Величина сдвига в последовательной корреляции[2] или автокорреляции.

---

[1] Известна также как автокорреляция первого порядка. — *Примеч. пер.*

[2] В случае последовательной корреляции лаг всегда равен 1. — *Примеч. пер.*

Упражнения

Упражнение 12.1

В качестве примера сезонной декомпозиции смоделируем среднемесячные значения температуры поверхности в США. Используем датасет проекта «Наш мир в данных» (Our World in Data), который содержит «значения температуры [в градусах Цельсия] воздуха, измеренные на высоте двух метров над уровнем земли, охватывающие поверхности суши, моря, а также внутриматериковые водные поверхности» для большинства стран мира за период с 1950 по 2024 год. Инструкции по загрузке данных приведены в Jupyter-блокноте этой главы.

Данные можно считать так:

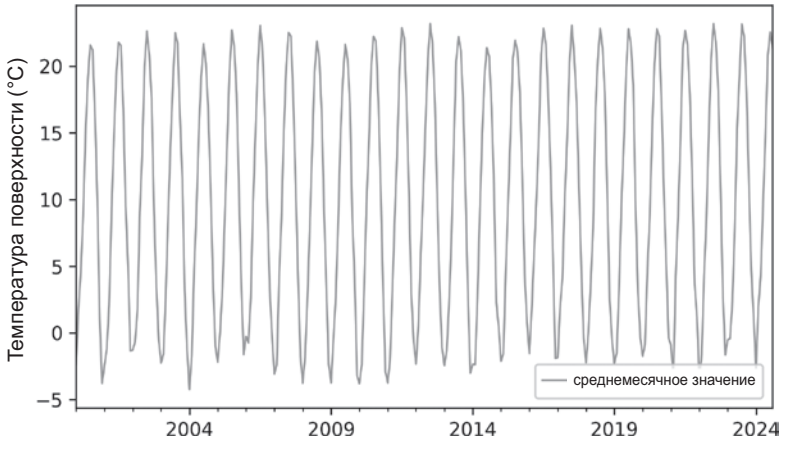
```
temp = pd.read_csv("monthly-average-surface-temperatures-by-year.csv")
```

В следующей ячейке выбираются данные по США с 2001 года до конца измерений и сохраняются в объекте `Series` библиотеки Pandas:

```
temp_us = temp.query("Code == 'USA'")
columns = [str(year) for year in range(2000, 2025)]
temp_series = temp_us.loc[:, columns].transpose().stack()
temp_series.index = pd.date_range(start="2000-01", periods=len(temp_series),
                                  freq="ME")
```

Вот как выглядят полученные результаты:

```
temp_series.plot(label="среднемесячное значение", **actual_options)
decorate(ylabel="Температура поверхности (°C)")
```



Вполне ожидаемо наблюдается четкая сезонная закономерность. Проведите аддитивную сезонную декомпозицию с периодом в 12 месяцев. Подгоните линейную модель к линии тренда. Каково среднегодовое повышение температуры поверхности за этот временной интервал? При желании повторите тот же анализ для других интервалов или данных из других стран.

Упражнение 12.2

Ранее в этой главе мы использовали мультипликативную сезонную декомпозицию для моделирования малой солнечной генерации электроэнергии с 2014 по 2019 год и прогнозирования производства на период с 2019 по 2024 год. Теперь проделаем то же самое для солнечной генерации промышленного масштаба (utility-scale solar power). Вот как выглядит ее временной ряд:

```
util_solar = elec["United States : all utility-scale solar"].dropna()
util_solar = util_solar[util_solar.index.year >= 2014]
util_solar.plot(**actual_options)
decorate(ylabel="Выработка (ГВт-ч)")
```



С помощью функции `split_series` разделите эти данные на обучающий и тестовый ряды. Выполните мультипликативную декомпозицию обучающего ряда за 12-месячный период. Подгоните линейную или квадратичную модель под тренд и сгенерируйте пятилетний прогноз, включая сезонную составляющую. Постройте график прогноза вместе с тестовым рядом и вычислите среднюю абсолютную процентную ошибку (MAPE).

Упражнение 12.3

Посмотрим, насколько хорошо модель ARIMA подходит для моделирования объема электроэнергии, произведенной гидроэлектростанциями в США. Вот как выглядит временной ряд с 2001 по 2024 год:

```
hydro = elec["United States : conventional hydroelectric"]
hydro.plot(**actual_options)
decorate(ylabel="Выработка (ГВт-ч)")
```



Подгоните модель SARIMA с периодом сезонности в 12 месяцев к этим данным. Поэкспериментируйте с разными величинами лага в авторегрессии и скользящим средним модели — проверьте, удастся ли вам найти комбинацию, которая максимизирует коэффициент R^2 модели. Создайте пятилетний прогноз и постройте его график вместе с доверительным интервалом.

ГЛАВА 13

Анализ выживаемости

Анализ выживаемости (survival analysis) — это способ описания того, как долго что-либо длится. Он часто используется для изучения продолжительности жизни человека, но также подходит для моделирования «выживаемости» механических и электронных компонентов, или, в более общем плане, интервала времени до наступления некоторого события, или даже расстояния в пространстве.

Начнем с простого примера — срока службы электрических лампочек, а затем рассмотрим другой, более существенный пример — возраст вступления в первый брак и то, как он изменился в США за последние 50 лет.

Функция выживания

Фундаментальным понятием в анализе выживаемости является **функция выживания** (survival function). Она представляет собой долю генеральной совокупности, которая сохраняется во времени дольше заданной продолжительности. В качестве первого примера мы рассчитаем функцию выживания для времени работы лампы накаливания.

Используем данные эксперимента, проведенного в 2007 году. Исследователи установили 50 новых ламп накаливания и оставили их постоянно включенными. Они проверяли лампочки каждые 12 часов и записывали срок службы каждой перегоревшей. Эксперимент продолжался, пока из строя не вышли все 50 лампочек. Инструкции по загрузке данных приведены в Jupyter-блокноте этой главы.

Считаем данные:

```
df = pd.read_csv("lamps.csv", index_col=0)
df.tail()
```

i	h	f	K
28	1812	1	4
29	1836	1	3
30	1860	1	2
31	1980	1	1
32	2568	1	0

В столбце h приведена продолжительность работы в часах (hours). В столбце f указано количество перегоревших лампочек для каждого значения h. Чтобы представить распределение срока службы, поместим эти значения в объект Pmf и нормализуем его:

```
from empiricaldist import Pmf

pmf_bulblife = Pmf(df["f"].values, index=df["h"])
pmf_bulblife.normalize()
```

```
50
```

С помощью метода make_cdf построим интегральную функцию распределения (ИФР), показывающую долю лампочек, вышедших из строя до или в момент времени h. Например, к моменту времени 1656 часов перегорает 78 % ламп:

```
cdf_bulblife = pmf_bulblife.make_cdf()
cdf_bulblife[1656]
```

```
0.7800000000000002
```

Функция выживания — это доля лампочек, перегорающих *после* какого-либо момента времени h, являющаяся дополнением (complement) к ИФР. Ее можно вычислить следующим образом:

```
complementary_cdf = 1 - cdf_bulblife
complementary_cdf[1656]
```

```
0.21999999999999975
```

После 1656 часов выходит из строя 22 % лампочек.

В библиотеке empiricaldist имеется класс Surv, реализующий функцию выживания. Объекты этого класса можно создавать при помощи метода make_surv:

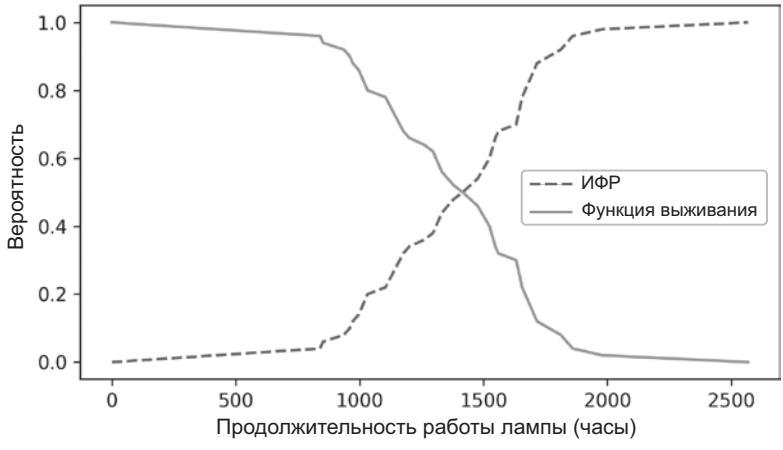
```
surv_bulblife = cdf_bulblife.make_surv()
surv_bulblife[1656]
```

```
0.21999999999999997
```

Если построить графики ИФР и функции выживания, то станет видно, что они дополняют друг друга, то есть их сумма равна 1 при любых значениях h:

```
cdf_bulblife.plot(ls="--", label="ИФР")
surv_bulblife.plot(label="Функция выживания")

decorate(xlabel="Продолжительность работы лампы (часы)", ylabel="Вероятность")
```



В этом смысле ИФР и функция выживания эквивалентны: если дана любая из них, можно вычислить другую. Но в рамках анализа выживаемости чаще всего приходится работать с кривыми выживания. А вычисление кривой выживания — это шаг к следующему важному понятию, функции риска (hazard function).

Функция риска

В датасете о лампах накаливания каждое значение `h` представляет собой 12-часовой интервал, заканчивающийся в час `h`. Его я буду называть «интервал `h`». Предположим, мы знаем, что лампочка проработала вплоть до интервала `h`, и хотели бы узнать вероятность того, что она перегорит в течение интервала `h`. Для ответа на этот вопрос можно использовать функцию выживания, показывающую долю лампочек, «переживших» интервал `h`, и функцию вероятности (ФВ), которая дает долю лампочек, перегоревших в течение интервала `h`. Сумма этих показателей равна доле ламп, которые *могут* перегореть в течение интервала `h` и поэтому называются находящимися «в группе риска» (at risk). В данном примере 26 % лампочек находились в группе риска в течение интервала 1656:

```
at_risk = pmf_bulblife + surv_bulblife
at_risk[1656]
```

```
0.25999999999999995
```

И 4 % всех лампочек перегорело в течение интервала 1656:

```
pmf_bulblife[1656]
```

```
0.04
```

Риск (hazard), или интенсивность отказов, в данном случае — это отношение `pmf_bulblife` и `at_risk`:

```
hazard = pmf_bulblife / at_risk
hazard[1656]
```

```
0.15384615384615388
```

Из всех ламп, которые остались в строю до интервала 1656, около 15 % перегорели в течение этого интервала.

Функцию риска можно вычислить самостоятельно или же воспользоваться библиотекой `empiricaldist`. В ней представлен объект `Hazard`, реализующий функцию риска, и метод `make_hazard`, который ее вычисляет:

```
hazard_bulblife = surv_bulblife.make_hazard()
hazard_bulblife[1656]
```

```
0.15384615384615397
```

Вот как выглядит функция риска для ламп накаливания:

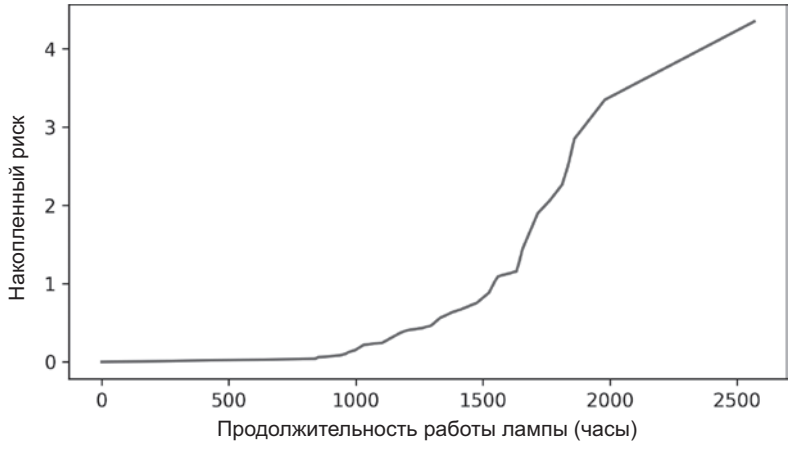
```
hazard_bulblife.plot()
decorate(xlabel="Продолжительность работы лампы (часы)", ylabel="Риск")
```



Видны всплески уровня риска в некоторых местах графика, но такой способ представления функции риска может вводить в заблуждение, особенно в тех частях диапазона, где мало данных. Лучше всего построить график **функции накопленного риска** (cumulative hazard function), которая представляет собой накопленную сумму (cumulative sum) риска:

```
cumulative_hazard = hazard_bulblife.cumsum()
cumulative_hazard.plot()

decorate(xlabel="Продолжительность работы лампы (часы)", ylabel="Накопленный
         риск")
```



Там, где вероятность выхода из строя высока, функция накопленного риска показывает крутой подъем. Там же, где эта вероятность мала, кумулятивная функция риска растет полого. В нашем примере можно заметить, что наивысший уровень риска наблюдается между 1500 и 2000 часами. После этого риск снижается — хотя это только благодаря одной необычно долгоживущей лампочке, поэтому в другом датасете он может выглядеть иначе.

Теперь, когда у вас появилось общее представление о функциях выживания и риска, применим их к более объемному датасету.

Данные о браке

Во многих странах люди сегодня вступают в брак позже, чем это было в прежние годы, и все больше людей остаются одинокими. Чтобы изучить эти тенденции на примере США, воспользуемся инструментами анализа выживаемости и данными Национального исследования роста семьи (NSFG) США.

Использованный в предыдущих главах датасет NSFG был посвящен беременностям и содержал по одной строке для каждой упомянутой респонденткой беременности. В этой главе мы будем работать с файлом, который содержит информацию о самих респондентках.

Я собрал ответы, полученные в ходе девяти этапов опроса, проводившихся в период с 1982 по 2019 год, и отобрал данные, касающиеся брака. Инструкции по загрузке этой выборки приведены в блокноте этой главы.

Считаем данные:

```
resp = pd.read_csv("marriage_nsfg_female.csv.gz")
resp.shape
```

```
(70183, 34)
```

В выборке каждая строка соответствует одной из более чем 70 000 респонденток и содержит следующие переменные, связанные с возрастом и браком:

`cmbirth`

Известная для всех респонденток дата рождения.

`cmintvw`

Известная для всех респонденток дата проведения опроса.

`cmmarrhx`

Дата вступления респондентки в первый брак, если брак имел место и дата известна.

`evrmarry`

1 — если респондентка состояла в браке до даты проведения опроса, 0 — в противном случае.

Первые три переменные записаны в кодировке «век — месяц» (Century-Month Code, CMC), то есть как целое число месяцев, прошедших с декабря 1899 года. Так что 1 в этом формате записи — это январь 1900 года.

Чтобы изучить эффект смены поколений, сгруппируем респонденток по десятилетию их даты рождения. В этом нам поможет следующая функция, которая принимает значение переменной `cmbirth` и вычисляет десятилетие даты рождения. В ней использован оператор целочисленного деления `//` для деления на 10 и округления в меньшую сторону:

```
month0 = pd.to_datetime("1899-12-31")

def decade_of_birth(cmbirth):
    date = month0 + pd.DateOffset(months=cmbirth)
    return date.year // 10 * 10
```

При помощи метода `apply` применим эту функцию и вычислим десятилетие даты рождения всех респонденток, а затем поместим полученные значения в новый столбец `cohort`. В данном контексте **когорта** (cohort) — это группа людей, объединенных чем-то общим (например, десятилетием своего рождения), для целей анализа рассматриваемая как единое целое.

Результатом функции `value_counts` является количество женщин в каждой когорте:

```
from thinkstats import value_counts

resp["cohort"] = resp["cmbirth"].apply(decade_of_birth)
value_counts(resp["cohort"])
```

```
cohort
1930      325
1940     3608
```

```
1950    10631
1960    14953
1970    16438
1980    14271
1990     8552
2000     1405
Name: count, dtype: int64
```

В этом датасете содержится информация о более 10 000 женщин, родившихся с 1950-х по 1980-е годы, и меньшее количество женщин из десятилетий до и после этого промежутка времени.

Далее рассчитаем возраст каждой респондентки на момент вступления в брак (если он имел место) и на дату проведения опроса:

```
resp["agemarr"] = (resp["cmmarrhx"] - resp["cmbirth"]) / 12
resp["age"] = (resp["cmintvw"] - resp["cmbirth"]) / 12
```

Прежде чем приступить к работе с этими данными, определим следующую функцию, которая принимает в качестве аргументов объект `DataFrame` и список когорт, а возвращает словарь, сопоставляющий каждую когорту с объектом `Surv`. Для каждой когорты функция выбирает возраст респондентки на момент вступления в первый брак и использует `Surv.from_seq` для вычисления функции выживания. Аргумент `dropna=False` позволяет учесть значения `NaN` в функции выживания, поэтому результат включает женщин, которые не выходили замуж:

```
from empiricaldist import Surv

def make_survival_map(resp, cohorts):
    surv_map = {}

    grouped = resp.groupby("cohort")
    for cohort in cohorts:
        group = grouped.get_group(cohort)
        surv_map[cohort] = Surv.from_seq(group["agemarr"], dropna=False)

    return surv_map
```

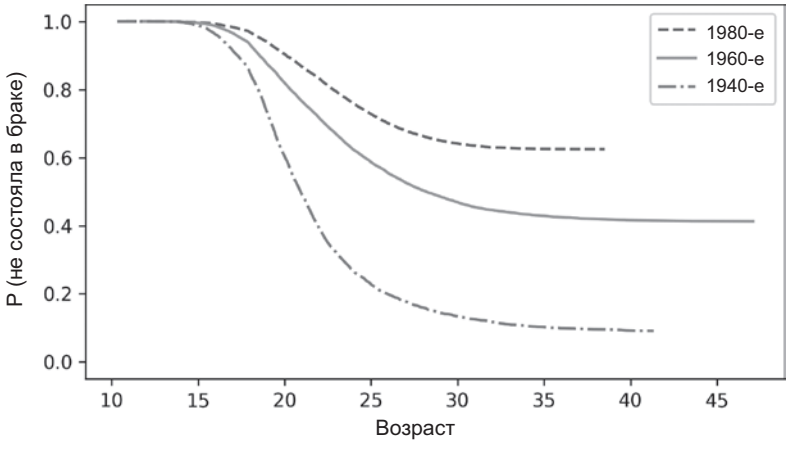
Вот пример ее использования:

```
cohorts = [1980, 1960, 1940]
surv_map = make_survival_map(resp, cohorts)
```

А вот результаты для женщин, родившихся в 1940-х, 1960-х и 1980-х годах:

```
for cohort, surv in surv_map.items():
    surv.plot(label=f"{cohort}-e")

ylim = [-0.05, 1.05]
decorate(xlabel="Возраст", ylabel="P(не состояла в браке)", ylim=ylim)
```



Если принять эти результаты за истину, то они показывают, что женщины предыдущих поколений вступали в брак в более молодом возрасте и доля хотя бы раз выходивших замуж была больше. Однако мы не будем заниматься их интерпретацией, поскольку они некорректны по двум причинам:

- Как указано в отчете «Национального исследования роста семьи» на с. 3, NSFG использует стратифицированную выборку, это означает, что в нем намеренно избыточно представлены некоторые социальные группы.
- Кроме того, наш способ вычисления функции выживания не учитывает должным образом людей, которые пока не состоят в браке.

Для решения первой проблемы воспользуемся методом ресемплинга, называемым **взвешенным бутстрепингом** (weighted bootstrap resampling). Для решения второй применим метод, называемый оценкой Каплана — Мейера. Начнем с ресемплинга.

Взвешенный бутстрепинг

В переменной `finalwgt` датасета NSFG содержится выборочный вес (sampling weight) каждого респондента, то есть количество людей из генеральной совокупности, которое он представляет. Эти веса можно использовать в процессе ресемплинга, чтобы сделать поправку на стратифицированность выборки. Функция ниже принимает на вход объект `DataFrame` и имя столбца, содержащего выборочные веса. Она выполняет ресемплинг строк датафрейма с учетом выборочных весов и возвращает новый объект `DataFrame`:

```
def resample_rows_weighted(df, column="finalwgt"):
    n = len(df)
    weights = df[column]
    return df.sample(n, weights=weights, replace=True)
```

Текущий датасет включает респондентов из нескольких итераций опроса, называемых циклами. Поэтому для ресемплинга нужно сначала сгруппировать респондентов по циклам, провести ресемплинг каждой группы, а затем снова объединить все группы. Эти шаги выполняет следующая функция:

```
def resample_cycles(resp):
    grouped = resp.groupby("cycle")
    samples = [resample_rows_weighted(group) for _, group in grouped]
    return pd.concat(samples)
```

Сначала проведем ресемплинг данных один раз:

```
sample = resample_cycles(resp)
```

Позже мы проведем множественный ресемплинг, чтобы посмотреть, насколько велика изменчивость, обусловленная случайной выборкой.

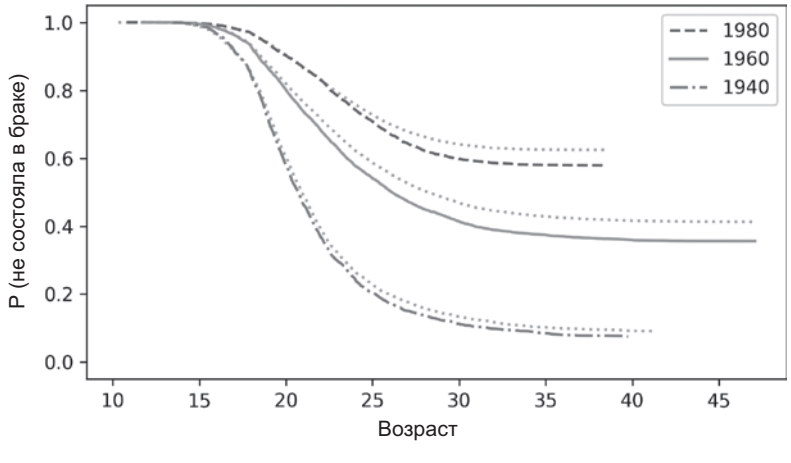
На рисунке ниже показаны результаты после проведения ресемплинга в сравнении с результатами из предыдущего раздела, где он не проводился, обозначенными пунктиром:

```
for label, surv in surv_map.items():
    surv.plot(ls=":", color="gray", alpha=0.6)

survs_resampled = make_survival_map(sample, cohorts)

for label, surv in survs_resampled.items():
    surv.plot(label=label)

decorate(xlabel="Возраст", ylabel="P(не состояла в браке)", ylim=ylim)
```



Разница результатов с проведением ресемплинга и без него существенна, и это доказывает, что для получения верных результатов нужно вводить поправку на стратифицированность выборки.

Теперь перейдем ко второй проблеме, связанной с неполнотой данных.

Оценка функции риска

В примере с лампами накаливания мы знаем продолжительность работы всех 50 ламп, поэтому можем напрямую вычислить функцию выживания, которую затем использовать для вычисления функции риска.

В примере со статистикой браков мы знаем возраст вступления в первый брак для тех респонденток, которые в него вступали к моменту опроса. Но для респонденток, которые никогда не выходили замуж, мы не знаем, в каком возрасте они сделают это в будущем и вступят ли в брак вообще.

Такого рода недостающие данные называются **цензурированными** (censored). Этот термин может показаться странным, поскольку информация, подвергшаяся цензуре, обычно скрывается преднамеренно, но в данном случае она отсутствует только потому, что мы не знаем будущего.

Однако имеется неполная информация, с которой можно работать: если на момент опроса женщина не состоит в браке, то мы знаем, что возраст ее вступления в брак (если это вообще произойдет) должен превышать ее текущий возраст.

Эту неполную информацию можно использовать для оценки функции риска, а затем с помощью функции риска вычислить функцию выживания. Этот процесс называется **оценкой Каплана — Мейера** (Kaplan-Meier estimation).

В демонстративных целях я выберу только одну когорту из повторной выборки данных:

```
resp60 = sample.query("cohort == 1960")
```

Для респонденток, которые состояли в браке на момент проведения опроса, выберем их возраст на момент вступления в первый брак. Всего имеется 9921 такой случай, назовем их «завершенными» (complete):

```
complete = resp60.query("evrmarry == 1")["agemarr"]
complete.count()
```

```
9921
```

Для респонденток, которые не состояли в браке, выберем их возраст на момент опроса. Всего есть 5468 таких случаев; их мы назовем «незавершенными» (ongoing):

```
ongoing = resp60.query("evrmarry == 0")["age"]
ongoing.count()
```

```
5468
```

Теперь, чтобы оценить функцию риска, вычислим общее количество случаев, которые были «в группе риска» для всех значений возраста, включая всех тех, кто не состоял в браке до этого возраста. Будет удобно создать объект `FreqTab`, который подсчитывает количество завершенных и незавершенных случаев для всех возрастов:

```
from empiricaldist import FreqTab

ft_complete = FreqTab.from_seq(complete)
ft_ongoing = FreqTab.from_seq(ongoing)
```

Например, 58 респонденток сообщили, что они впервые вышли замуж в возрасте 25 лет:

```
ft_complete[25]
```

```
58
```

А 5 респонденток, которые были опрошены в возрасте 25 лет, сообщили, что никогда не состояли в браке:

```
ft_ongoing[25]
```

```
5
```

Из этих объектов класса `FreqTab` можно вычислить ненормированные объекты класса `Surv`, которые содержат количество завершенных и незавершенных случаев для возраста, превышающего заданный:

```
surv_complete = ft_complete.make_surv()
surv_ongoing = ft_ongoing.make_surv()
```

Например, 2848 женщин сообщили, что вступили в брак после 25 лет:

```
surv_complete[25]
```

```
2848
```

И 2273 опрошенных в возрасте после 25 лет сообщили, что никогда не состояли в браке:

```
surv_ongoing[25]
```

```
2273
```

Сумма только что вычисленных четырех чисел представляет собой число респондентов, которые находились в группе риска, то есть могли бы вступить в брак в возрасте 25 лет. Термин «группа риска» является наследием анализа выживаемости в медицине, где он зачастую связан с угрозой заболевания или смерти. Он может показаться неуместным применительно к браку, обычно считающемуся позитивным этапом жизни. Как бы то ни было, вот наши выкладки:

```
at_risk = ft_complete[25] + ft_ongoing[25] + surv_complete[25] +
          surv_ongoing[25]
at_risk
```

```
5184
```

Из этого числа количество тех, кто действительно вступил в брак в возрасте 25 лет, равно `ft_complete[25]`. И теперь мы можем вычислить функцию риска в возрасте 25 лет:

```
hazard = ft_complete[25] / at_risk
hazard
```

```
0.011188271604938271
```

Мы показали, как вычислить функцию риска для определенного возраста. Теперь вычислим ее целиком, для всех возрастов. Воспользуемся методом `union` класса `Index` библиотеки Pandas, чтобы получить индекс, содержащий все значения возраста из объектов `ft_complete` и `ft_ongoing` по порядку:

```
ts = pd.Index.union(ft_complete.index, ft_ongoing.index)
```

Теперь посчитаем количество женщин группы риска для всех возрастов, сделав выборку значений в каждом из объектов `FreqTab` и `Surv` по возрастам из переменной `ts`:

```
at_risk = ft_complete(ts) + ft_ongoing(ts) + surv_complete(ts) +
          surv_ongoing(ts)
```

Наконец, вычислим функцию риска для всех значений возраста и поместим результаты в объект `Hazard`:

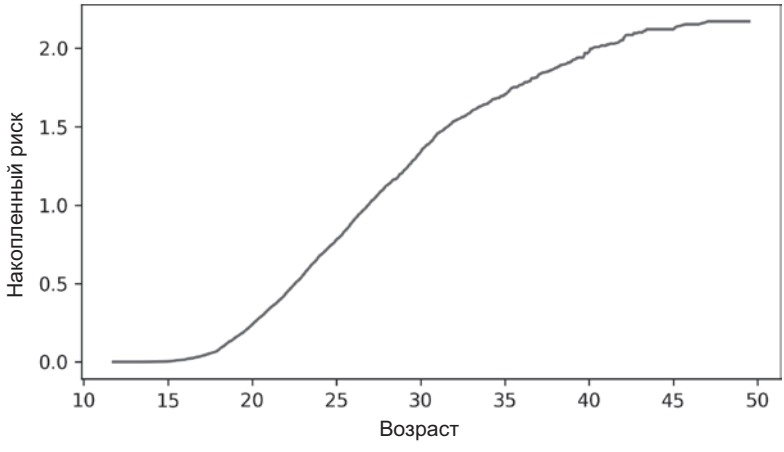
```
from empiricaldist import Hazard

hs = ft_complete(ts) / at_risk
hazard = Hazard(hs, ts)
```

Вот как выглядит функция накопленного риска:

```
hazard.cumsum().plot()

decorate(xlabel="Возраст", ylabel="Накопленный риск")
```



В диапазоне от 20 до 30 лет наблюдается наибольшая крутизна графика, это означает, что в этом возрасте одинокая женщина подвергается наибольшему «риску» вступления в брак. После этого накопленный риск выходит на плато, что означает постепенное снижение риска.

Оценка функции выживания

Зная функцию выживания, можно вычислить функцию риска. А если наоборот?

Вот один из способов. Функция риска показывает вероятность вступить в брак в зависимости от возраста, если брака раньше не было. Следовательно, комплементарной к функции риска является вероятность остаться не замужем как функция возраста.

Чтобы «пережить» определенный возраст `t`, нужно оставаться не замужем в любом возрасте до `t` включительно. И вероятность этого события является накопленным произведением (cumulative product) функции, комплементарной к функции риска, которую можно вычислить следующим образом:

```
ps = (1 - hazard).cumprod()
```

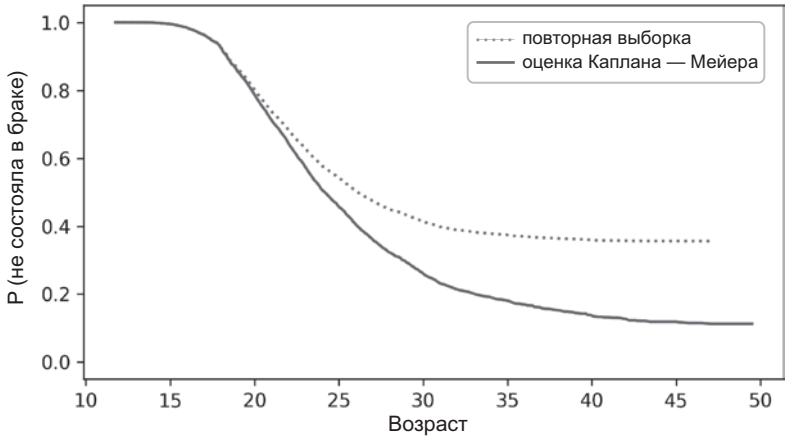
У объекта `Hazard` есть метод `make_surv`, который выполняет этот подсчет:

```
surv = hazard.make_surv()
```

Вот как выглядит полученный результат по сравнению с предыдущим (пунктирная линия), в котором делалась поправка на стратифицированность выборки с помощью ресемплинга, но не учитывались цензурированные данные:

```
survs_resampled[1960].plot(ls=":", color="gray", label="повторная выборка")
surv.plot(label="оценка Каплана – Мейера")

decorate(xlabel="Возраст", ylabel="P(не состояла в браке)", ylim=ylim)
```



Это показывает, насколько важно правильно обрабатывать цензурированные данные.

Подобная функция выживания легла в основу известной статьи 1986 года. Журнал *Newsweek* сообщал, что у 40-летней незамужней женщины «больше шансов быть убитой террористом», чем выйти замуж. Это утверждение получило широкую огласку и стало частью массовой культуры, но оно изначально было неверным (потому что основывалось на ошибочном анализе) и оказалось еще более неверным (из-за культурных изменений, которые уже начались тогда). В 2006 году *Newsweek* опубликовал другую статью, в которой признал свою ошибку.

Я рекомендую узнать больше об этой статье, статистических данных, на которых она была основана, и реакции на нее. Она напоминает о наших моральных обязательствах тщательно проводить статистический анализ, интерпретировать результаты с должным уровнем скептицизма, а также аккуратно и честно представлять их на суд общественности.

Следующая функция реализует все эти этапы оценки Каплана — Мейера. Она принимает на вход последовательности значений времени выживания для завершенных и незавершенных случаев, а затем возвращает объект `Hazard`:

```
def estimate_hazard(complete, ongoing):
    """Оценка Каплана — Мейера."""
    ft_complete = FreqTab.from_seq(complete)
    ft_ongoing = FreqTab.from_seq(ongoing)

    surv_complete = ft_complete.make_surv()
    surv_ongoing = ft_ongoing.make_surv()

    ts = pd.Index.union(ft_complete.index, ft_ongoing.index)
    at_risk = (
        ft_complete(ts) + ft_ongoing(ts) +
        surv_complete(ts) + surv_ongoing(ts)
```

```
    )

    hs = ft_complete(ts) / at_risk
    return Hazard(hs, ts)
```

А функция ниже принимает на вход данные группы респондентов, извлекает значения времени выживания, получает функцию риска в результате вызова `estimate_hazard`, а затем вычисляет соответствующую функцию выживания:

```
def estimate_survival(group):
    """Оценка функции выживания."""
    complete = group.query("evrmarry == 1")["agemarr"]
    ongoing = group.query("evrmarry == 0")["age"]
    hf = estimate_hazard(complete, ongoing)
    sf = hf.make_surv()
return sf
```

Нам вскоре потребуются эти функции для вычисления доверительного интервала для функции выживания. Но сначала рассмотрим еще один способ вычисления оценок Каплана — Мейера.

Библиотека Lifelines

Python-пакет `lifelines` предоставляет инструменты для анализа выживаемости, в том числе функции для вычисления оценок Каплана — Мейера.

Воспроизведем с его помощью результат, полученный в предыдущем разделе, тем самым доказав его правильность. Сначала вычислим функцию выживания, вызвав `estimate_survival`:

```
surv = estimate_survival(resp60)
```

Далее получим ее с помощью инструментов `lifelines`. Сначала преобразуем данные в формат, совместимый с `lifelines`:

```
complete = complete.dropna()
durations = np.concatenate([complete, ongoing])
event_observed = np.concatenate([np.ones(len(complete)),
np.zeros(len(ongoing))])
```

Теперь можно создать объект `KaplanMeierFitter` и подогнать модель под данные:

```
from lifelines import KaplanMeierFitter

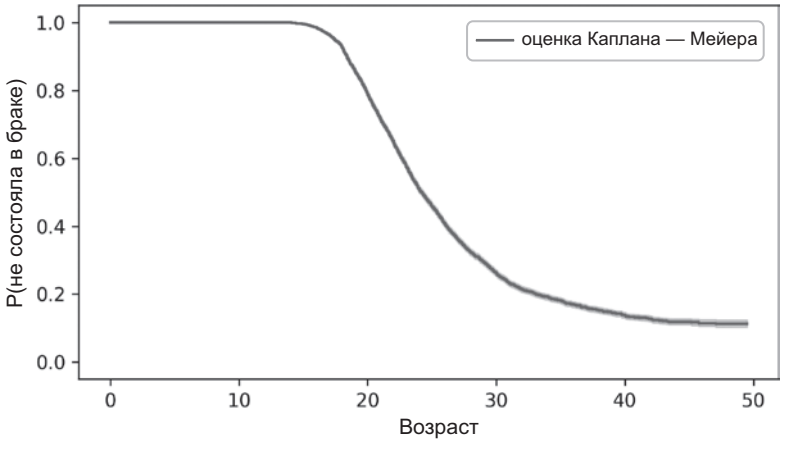
kmf = KaplanMeierFitter()
kmf.fit(durations=durations, event_observed=event_observed)
```

```
<lifelines.KaplanMeierFitter:"KM_estimate", fitted with 15389 total
          observations[1],
5468 right-censored observations[2]>
```

После подгонки к данным можно отобразить результаты с помощью функции `plot`. Результаты содержат оценку функции выживания и доверительный интервал, хотя в данном случае последний неверен, поскольку не учитывает стратифицированность выборки:

```
kmf.plot()

decorate(xlabel="Возраст", ylabel="P(не состояла в браке)", ylim=ylim)
```



В отличие от функции выживания, которую мы вычислили, функция из библиотеки `lifelines` определена, начиная с 0. Но остальная часть значений функции совпадает с нашими — в пределах погрешности арифметики с плавающей точкой:

```
ps = kmf.survival_function_["KM_estimate"].drop(0)
np.allclose(ps, surv)
```

```
True
```

В следующем разделе воспользуемся взвешенным ресемплингом для вычисления доверительного интервала, который учитывает стратифицированность выборки.

[1] fitted with 15389 total observations — подогнано к 15 389 наблюдениям. — *Примеч. пер.*

[2] 5468 right-censored observations — 5468 цензурированных справа наблюдений. — *Примеч. пер.*

Доверительный интервал

Вычисленная нами оценка Каплана — Мейера основана на однократном ресемплинге датасета. Чтобы получить представление о том, насколько велика вариативность из-за случайной выборки, проведем расчеты с несколькими повторными выборками и построим график результатов.

Применим следующую функцию, которая принимает объект `DataFrame` и список когорт, оценивает функцию выживания для каждой когорты и возвращает словарь, сопоставляющий каждую целочисленную когорту с объектом класса `Surv`.

Эта функция идентична функции `make_survival_map`, за исключением того что она вызывает функцию `estimate _survival`, использующую оценку Каплана — Мейера, вместо метода `Surv.from_seq`, который подходит только при отсутствии цензурированных данных:

```
def estimate_survival_map(resp, cohorts):
    """Создать словарь, сопоставляющий когортам объекты Surv."""
    surv_map = {}

    grouped = resp.groupby("cohort")
    for cohort in cohorts:
        group = grouped.get_group(cohort)
        surv_map[cohort] = estimate_survival(group)

    return surv_map
```

В цикле ниже случайно генерируется 101 повторная выборка нашего датасета и создается список из 101 словаря, содержащего оценки функции выживания:

```
cohorts = [1940, 1950, 1960, 1970, 1980, 1990]

surv_maps = [estimate_survival_map(resample_cycles(resp), cohorts)
             for i in range(101)]
```

Чтобы отобразить результаты, воспользуемся функцией ниже, которая принимает полученный список словарей, целочисленное значение когорты и строку с названием цвета графика (color). Функция перебирает словари, выбирает функцию выживаемости для указанной когорты и отображает ее полупрозрачной линией — это один из способов визуализировать изменчивость в повторных выборках:

```
def plot_cohort(surv_maps, cohort, color):
    """Построить диаграмму результатов для одной когорты."""
    survs = [surv_map[cohort] for surv_map in surv_maps]
    for surv in survs:
        surv.plot(color=color, alpha=0.05)

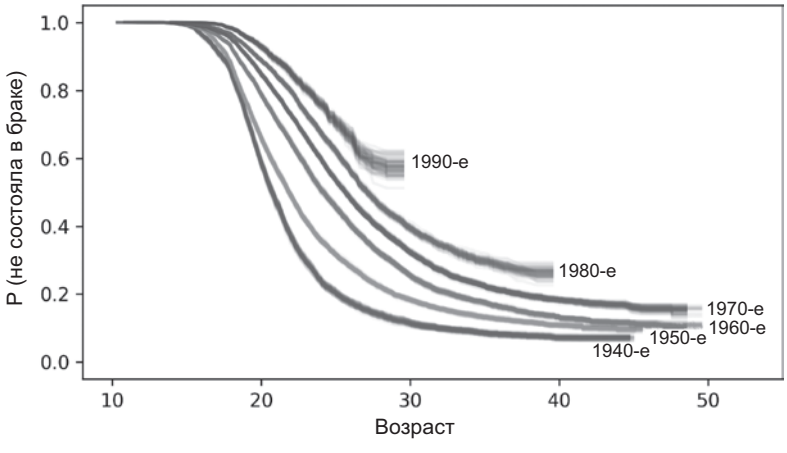
    x, y = surv.index[-1], surv.iloc[-1]
    plt.text(x + 1, y, f"{cohort}-e", ha="left", va="center")
```

Вот результаты по когортам рождения с 1940-х по 1990-е годы:

```
colors = [f"C{i}" for i in range(len(cohorts))]

for cohort, color in zip(cohorts, colors):
    plot_cohort(surv_maps, cohort, color)

xlim = [8, 55]
decorate(xlabel="Возраст", ylabel="P(не состояла в браке)",
         xlim=xlim, ylim=ylim)
```



Полученная диаграмма вполне подходит для изучения, но линии выглядят размытыми, а некоторые надписи перекрывают друг друга. Для получения рисунка, подходящего для публикации, потребуется больше усилий, но мы пока не будем ничего усложнять.

На диаграмме видны несколько закономерностей:

- Женщины, родившиеся в 1940-х годах, выходили замуж раньше всех, а женщины, родившиеся в 1950-х и 1960-х годах, хотя и выходили замуж позже, но среди них доля оставшихся незамужними примерно одинакова.
- Женщины, родившиеся в 1970-х годах, позже выходили замуж *и* чаще оставались незамужними по сравнению с женщинами в предыдущих когортах.
- Родившиеся в 1980-х и 1990-х годах вступают в брак еще позже и еще чаще остаются незамужними, хотя в будущем эти тенденции могут измениться.

Придется дождаться следующего выпуска данных NSFG, чтобы продолжить наши исследования.

Ожидаемое остаточное время жизни

Имея некоторое распределение, можно рассчитать ожидаемое остаточное время жизни (expected remaining lifetime) как функцию от прошедшего времени. Например, если дано распределение продолжительности беременности, можно рассчитать ожидаемое время до наступления родов. В демонстрационных целях будем использовать данные о беременностях из NSFG.

С помощью функции `get_nsfg_groups` считаем данные и разделим их на соответствующие первым (firsts) и последующим (others) детям:

```
from nsfg import get_nsfg_groups

live, firsts, others = get_nsfg_groups()
```

Начнем с однократного ресемплинга данных:

```
sample = resample_rows_weighted(live, "finalwgt")
```

Вот функция вероятности продолжительности беременности:

```
pmf_durations = Pmf.from_seq(sample["prglngth"])
```

Теперь предположим, что сейчас начало 36-й недели беременности. Зная, что чаще всего срок беременности составляет 39 недель, естественно ожидать, что остаточное время составит 3–4 недели. Чтобы уточнить эту оценку, отберем значения в распределении, которые больше или равны 36 неделям:

```
t = 36
is_remaining = pmf_durations.qs >= t
```

Далее создадим новый объект `Pmf`, содержащий только эти значения со сдвигом влево — чтобы текущее время было равно 0:

```
ps = pmf_durations.ps[is_remaining]
qs = pmf_durations.qs[is_remaining] - t

pmf_remaining = Pmf(ps, qs)
```

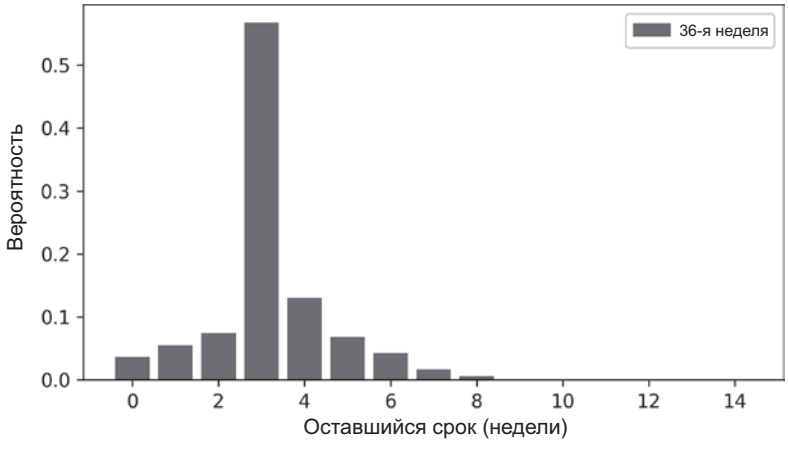
Поскольку было выбрано подмножество значений исходного объекта `Pmf`, вероятности больше не суммируются в 1, но это можно исправить при помощи нормировки `Pmf`:

```
pmf_remaining.normalize()
```

```
0.9155006558810669
```

Вот диаграмма полученного распределения остаточного времени на начало 36-й недели:

```
pmf_remaining.bar(label="36-я неделя")
decorate(xlabel="Оставшийся срок (недели)", ylabel="Вероятность")
```



Среднее значение этого распределения — ожидаемое остаточное время:

```
pmf_remaining.mean()
```

```
3.2145671641791043
```

Следующая функция объединяет эти шаги и вычисляет распределение остаточного времени для заданного объекта `Pmf` в данный момент времени `t`:

```
def compute_pmf_remaining(pmf, t):
    """Распределение остаточного времени."""
    is_remaining = pmf.qs >= t
    ps = pmf.ps[is_remaining]
    qs = pmf.qs[is_remaining] - t
    pmf_remaining = Pmf(ps, qs)
    pmf_remaining.normalize()
    return pmf_remaining
```

Следующая функция принимает на вход объект `Pmf` с продолжительностями беременности и вычисляет ожидаемое остаточное время в начале каждой недели с 36-й по 43-ю:

```
def expected_remaining(pmf):
    index = range(36, 44)
    expected = pd.Series(index=index)

    for t in index:
```

```
        pmf_remaining = compute_pmf_remaining(pmf, t)
        expected[t] = pmf_remaining.mean()

    return expected
```

Вот результаты однократного ресемплинга данных:

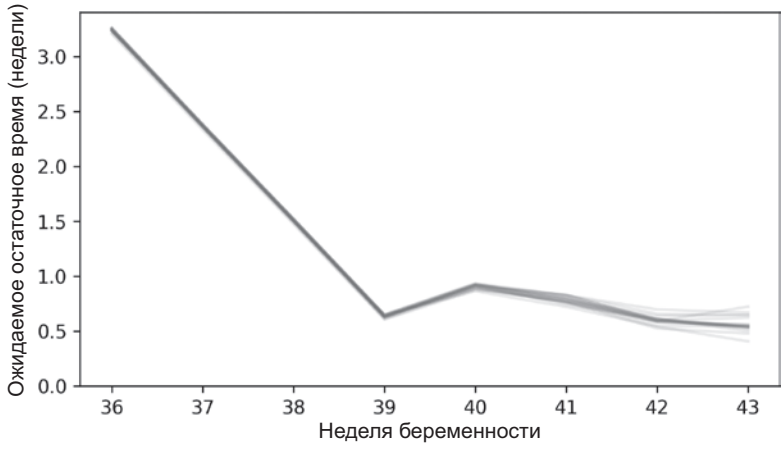
```
expected = expected_remaining(pmf_durations)
expected
```

```
36    3.214567
37    2.337714
38    1.479095
39    0.610133
40    0.912517
41    0.784211
42    0.582301
43    0.589372
dtype: float64
```

Чтобы оценить, насколько велика вариативность, обусловленная случайной выборкой, проведем эксперимент с несколькими ресемплингами и построим график результатов:

```
for i in range(21):
    sample = resample_rows_weighted(live, "finalwgt")
    pmf_durations = Pmf.from_seq(sample["prglngth"])
    expected = expected_remaining(pmf_durations)
    expected.plot(color="C0", alpha=0.1)

decorate(
    xlabel="Неделя беременности",
    ylabel="Ожидаемое остаточное время (недели)", ylim=[0, 3.4]
)
```



С 36-й по 39-ю неделю ожидаемое остаточное время уменьшается и в начале 39-й недели достигает уровня около 0.6 недели. Но далее кривая стабилизируется. В начале 40-й недели ожидаемое остаточное время все еще близко к 0.6 недели, как и в начале 41-й, 42-й и 43-й недель. Для родителей, с нетерпением ожидающих появления на свет ребенка, такая закономерность может показаться довольно мучительной.

Глоссарий

Анализ выживаемости (survival analysis)

Набор методов для описания и предсказания времени до наступления определенного события, часто касающийся анализа продолжительности жизни или срока службы.

Функция выживания (survival function)

Функция, которая сопоставляет времени t вероятность «пережить» момент t.

Функция риска (hazard function)

Функция, которая сопоставляет времени t долю случаев, в которых событие наступило в момент t, среди всех случаев, «доживших» до момента t.

Функция накопленного риска (cumulative hazard function)

Накопленная сумма функции риска, часто используемая для визуализации.

Взвешенный бутстрепинг (weighted bootstrap resampling)

Форма ресемплинга, которая использует выборочные веса для корректировки стратифицированной выборки с целью моделирования репрезентативной выборки.

Цензурированные данные (censored data)

Данные, которые известны лишь частично, поскольку определенное событие еще не произошло или осталось ненаблюдаемым.

Оценка Каплана — Мейера (Kaplan-Meier estimation)

Метод оценки функции выживания и функции риска для датасетов с цензурированными наблюдениями.

Когорта (cohort)

Группа испытуемых с общими характеристиками, например диагноз или десятилетие даты рождения.

Упражнения

Упражнение 13.1

Методы, описанные в этой главе, можно использовать для оценки функции риска и функции выживания применительно к продолжительности брака. Для простоты мы рассмотрим только первые браки и ограничимся разводом в качестве конечной точки, не рассматривая такие сценарии, как раздельное проживание или смерть.

В данных NSFG переменная `cmdivorcx` содержит дату развода для первого брака каждого респондента, если таковой имел место, записанную в кодировке «век — месяц». Подсчитайте продолжительность браков, завершившихся разводом, и продолжительность текущих браков.

- Для завершенных случаев вычислите время, прошедшее между `cmdivorcx` и `cmmarrhx`. Если оба значения корректны, то есть не равны `NaN`, это означает, что первый брак респондента закончился разводом.
- Чтобы выявить незавершенные случаи, выберите женщин, которые были замужем только один раз и все еще состоят в браке. Используйте переменную `fmarno`, в которой записано количество браков каждой респондентки, и переменную `fmarital`, кодирующую ее семейное положение, — значение 1 указывает на то, что респондентка замужем.

Иногда значения этих переменных приблизительные, поэтому вы можете встретить небольшое количество случаев с отрицательной разницей, но она не должна превышать одного года.

Проведите оценку функции риска и функции выживания для продолжительности брака. Постройте график функции накопленного риска — когда риск развода наиболее высок? Постройте график функции выживания — какая доля браков заканчивается разводом?

Упражнение 13.2

В 2012 году группа демографов из Университета Южной Калифорнии подсчитала ожидаемую продолжительность жизни людей, родившихся в Швеции в начале XIX и XX веков. Для людей в возрасте от 0 до 91 года они рассчитали повозрастной коэффициент смертности (age-specific mortality rate), представляющий собой долю людей, умирающих в определенном возрасте, из всех, кто доживает до этого возраста, — зависимость, в которой вы могли узнать функцию риска.

Я оцифровал онлайн-данные из статьи и сохранил их в CSV-файле. Инструкции по загрузке данных приведены в Jupyter-блокноте этой главы.

Данные можно загрузить так:

```
mortality = pd.read_csv("mortality_rates_beltran2012.csv",
            header=[0, 1]).dropna()
```

Следующая функция с помощью интерполяции данных вычисляет функцию риска с приближенными значениями коэффициента смертности для всех возрастов от 0 до 99 лет:

```
from scipy.interpolate import interp1d
from empiricaldist import Hazard

def make_hazard(ages, rates):
    interp = interp1d(ages, rates, fill_value="extrapolate")
    xs = np.arange(0, 100)
    ys = np.exp(interp(xs))
    return Hazard(ys, xs)
```

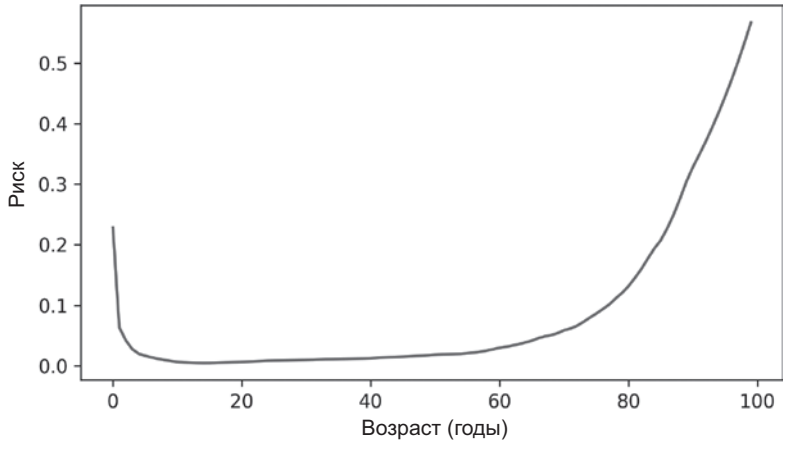
Теперь можно создать объект `Hazard`:

```
ages = mortality["1800", "X"].values
rates = mortality["1800", "Y"].values
hazard = make_hazard(ages, rates)
```

Вот график коэффициента смертности:

```
hazard.plot()

decorate(xlabel="Возраст (годы), ylabel="Риск")
```



На основе этих данных с помощью метода `make_surv` создайте функцию выживания и вызовите `make_cdf`, чтобы вычислить соответствующую ИФР. Постройте график результатов.

Затем с помощью метода `make_pmf` создайте объект `Pmf`, представляющий распределение продолжительности жизни, и изобразите его графически. Наконец, используйте функцию `compute_pmf_remaining` для вычисления среднего остаточного

времени жизни для всех возрастов от 0 до 99 лет. Постройте диаграмму полученных результатов.

На кривой остаточного времени жизни вы должны заметить парадоксальную закономерность: в течение первых нескольких лет жизни остаточное время жизни увеличивается. Так как в начале XIX века детская смертность была очень высокой, можно было ожидать, что дети более старшего возраста проживут дольше, чем дети младшего возраста. Примерно после 5 лет ожидаемая продолжительность жизни возвращается к предсказуемому поведению — ожидается, что молодые люди проживут дольше, чем пожилые.

Если вас заинтересовала эта тема, рекомендую ознакомиться с главой 5 моей книги *Probably Overthinking It* (University of Chicago Press, 2023). В ней вы найдете другие столь же непредсказуемые результаты из разных областей статистики.

ГЛАВА 14

Аналитические методы

В этой книге основное внимание уделяется вычислительным методам, таким как моделирование и ресемплинг, но некоторые из рассмотренных нами задач имеют аналитическое решение, которое можно вывести гораздо быстрее.

В этой главе представлены некоторые из таких методов, а также объясняются принципы их работы. В конце главы я даю советы, как при анализе данных объединить вычислительные и аналитические методы.

График нормальной вероятности

В основе многих аналитических методов лежат свойства нормального распределения. Этому есть два объяснения: распределения многих показателей физического мира хорошо аппроксимируются нормальным распределением, к тому же некоторые математические свойства нормального распределения делают его удобным для анализа.

Чтобы продемонстрировать первое обстоятельство, посмотрим на некоторые переменные из датасета о пингвинах. Затем мы исследуем математические свойства нормального распределения. Инструкции по загрузке данных приведены в Jupyter-блокноте этой главы.

Загрузим данные:

```
penguins = pd.read_csv("penguins_raw.csv")
penguins.shape
```

```
(344, 17)
```

Датасет содержит результаты измерений трех видов пингвинов. В этом примере выберем пингвинов вида Адели:

```
adelie = penguins.query('Species.str.startswith("Adelie")').copy()
len(adelie)
```

```
152
```

Чтобы проверить, соответствует ли вес пингвинов нормальному распределению, вычислим эмпирическую интегральную функцию распределения (ИФР) результатов измерений:

```
from empiricaldist import Cdf

weights = adelie["Body Mass (g)"].dropna()
cdf_weights = Cdf.from_seq(weights)
```

Также вычислим аналитическую ИФР нормального распределения с теми же средним значением и стандартным отклонением:

```
m, s = weights.mean(), weights.std()
m, s
```

```
(3700.662251655629, 458.5661259101348)
```

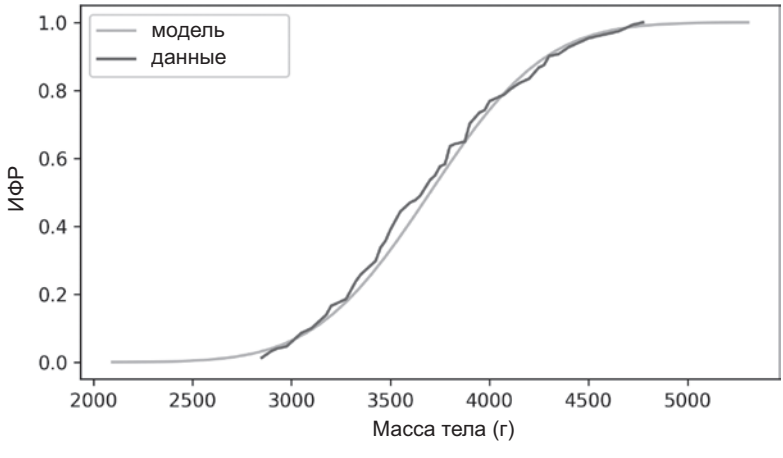
```
from scipy.stats import norm

dist = norm(m, s)
qs = np.linspace(m - 3.5 * s, m + 3.5 * s)
ps = dist.cdf(qs)
```

Вот как выглядит ИФР фактических данных в сравнении с нормальной моделью:

```
model_options = dict(color="gray", alpha=0.5, label="модель")
plt.plot(qs, ps, **model_options)
cdf_weights.plot(label="данные")

decorate(ylabel="ИФР")
```



Нормальное распределение можно считать достаточно хорошей моделью этих данных, но оно, безусловно, не идеально согласуется с ними.

В целом построение ИФР данных и ИФР модели — хороший способ проверить, насколько модель соответствует данным. Однако этот метод зависит от того, хорошо ли мы оценили параметры модели — в данном примере среднее значение и стандартное отклонение.

Альтернативой ему является **график нормальной вероятности** (normal probability plot), который не зависит от нашего умения оценивать параметры. На графике нормальной вероятности значения y — отсортированные выборочные данные:

```
ys = np.sort(weights)
```

А значения x — это соответствующие квантили нормального распределения, вычисленные с использованием метода `ppf` объекта `norm`. Этот метод считает квантильную функцию (percent point function, PPF), которая является обратной к ИФР:

```
n = len(weights)
ps = (np.arange(n) + 0.5) / n
xs = norm.ppf(ps)
```

Если результаты измерений на самом деле подчиняются нормальному распределению, то координаты y и x должны лежать на прямой линии. Чтобы посмотреть, так ли это, можно с помощью функции `linregress` подобрать линию регрессии:

```
from scipy.stats import linregress

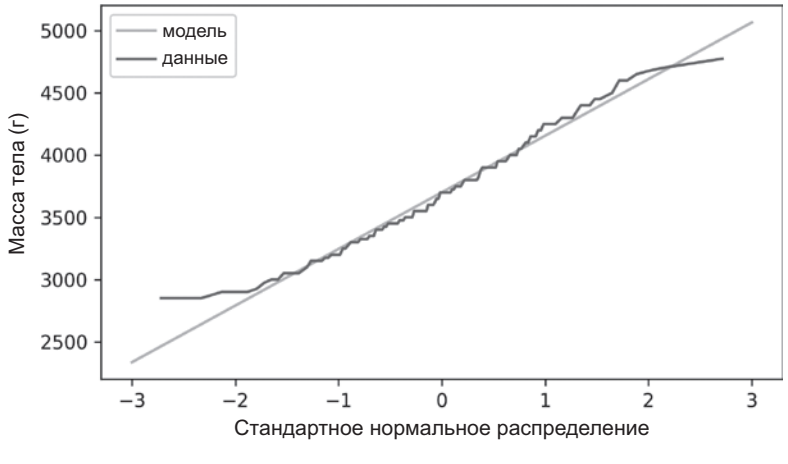
results = linregress(xs, ys)
intercept, slope = results.intercept, results.slope

fit_xs = np.linspace(-3, 3)
fit_ys = intercept + slope * fit_xs
```

Следующий рисунок демонстрирует величины x и y вместе с подогнанной линией:

```
plt.plot(fit_xs, fit_ys, **model_options)
plt.plot(xs, ys, label="данные")

decorate(xlabel="Стандартное нормальное распределение", ylabel="Масса тела (г)")
```



Данный график нормальной вероятности — не безукоризненно прямая линия, что указывает на то, что нормальное распределение не лучшая модель для этих данных.

Одна из причин в том, что данные описывают как самцов, так и самок пингвинов, а у этих двух групп разные средние значения. Посмотрим, что изменится, если мы построим графики этих групп по отдельности. Следующая функция аккумулирует все шаги, которые мы совершили для построения графика нормальной вероятности:

```
def normal_probability_plot(sample, **options):
    """Построить график нормальной вероятности с подогнанной линией."""
    n = len(sample)
    ps = (np.arange(n) + 0.5) / n
    xs = norm.ppf(ps)
    ys = np.sort(sample)

    results = linregress(xs, ys)
    intercept, slope = results.intercept, results.slope

    fit_xs = np.linspace(-3, 3)
    fit_ys = intercept + slope * fit_xs

    plt.plot(fit_xs, fit_ys, color="gray", alpha=0.5)
    plt.plot(xs, ys, **options)
    decorate(xlabel="Стандартное нормальное распределение")
```

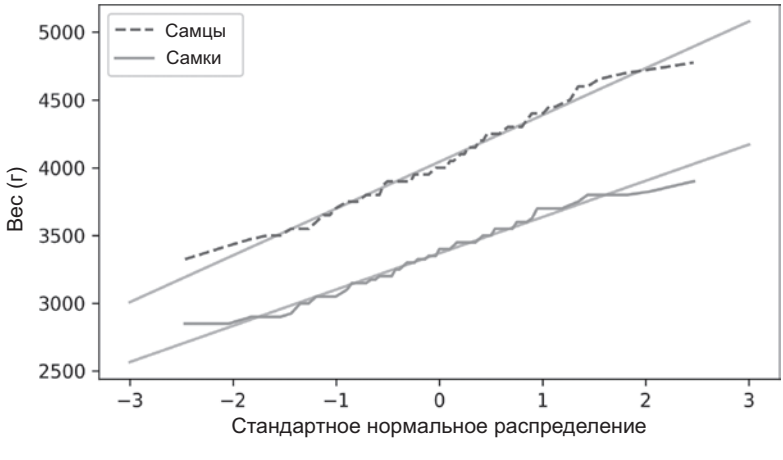
Вот как выглядят результаты для самцов и самок пингвинов по отдельности:

```
grouped = adelie.groupby("Sex")

weights_male = grouped.get_group("MALE")["Body Mass (g)"]
normal_probability_plot(weights_male, ls="--", label="Самцы")

weights_female = grouped.get_group("FEMALE")["Body Mass (g)"]
normal_probability_plot(weights_female, label="Самки")

decorate(ylabel="Вес (г)")
```



Графики нормальных вероятностей для обеих групп близки к прямой линии, это указывает на то, что распределение веса соответствует нормальному распределению. Когда мы объединяем группы, распределение их весов представляет собой смесь двух нормальных распределений с разными средними, а она не всегда хорошо моделируется нормальным распределением.

Теперь рассмотрим, какие математические свойства нормального распределения делают его удобным для анализа.

Нормальное распределение

Следующий класс описывает объект, представляющий нормальное распределение. В качестве атрибутов используются параметры `mu` и `sigma2` — среднее значение и дисперсия распределения. Название `sigma2` напоминает, что дисперсия — это квадрат стандартного отклонения, которое обычно обозначается как `sigma`:

```
class Normal:
    """Представляет нормальное распределение"""

    def __init__(self, mu, sigma2):
        self.mu = mu
        self.sigma2 = sigma2

    def __repr__(self):
        return f"Normal({self.mu}, {self.sigma2})"

    __str__ = __repr__
```

Для примера создадим объект `Normal`, представляющий нормальное распределение с тем же средним значением и дисперсией, что и вес самцов пингвинов:

```
m, s = weights_male.mean(), weights_male.std()
dist_male = Normal(m, s**2)
dist_male
```

```
Normal(4043.4931506849316, 120278.25342465754)
```

И еще один объект `Normal` со средним значением и дисперсией как у веса самок пингвинов:

```
m, s = weights_female.mean(), weights_female.std()
dist_female = Normal(m, s**2)
dist_female
```

```
Normal(3368.8356164383563, 72565.63926940637)
```

Далее в класс `Normal` добавим метод, который генерирует случайную выборку из нормального распределения. Новые методы к существующему классу мы будем добавлять с помощью «магической» команды Jupyter `add_method_to`, которая

определена в модуле `thinkstats`. Эта команда не является функционалом языка Python — ее можно использовать только в Jupyter-блокнотах:

```
%%add_method_to Normal

def sample(self, n):
    sigma = np.sqrt(self.sigma2)
    return np.random.normal(self.mu, sigma, n)
```

Воспользуемся методом `sample`, чтобы продемонстрировать первое полезное свойство нормального распределения: если извлекать значения из двух нормальных распределений и складывать их, то распределение полученной суммы также будет нормальным.

Для примера с помощью только что созданных объектов `Normal` сгенерируем две выборки, сложим их и построим график нормальной вероятности сумм:

```
sample_sum = dist_male.sample(1000) + dist_female.sample(1000)
normal_probability_plot(sample_sum)

decorate(ylabel="Суммарный вес (г)")
```

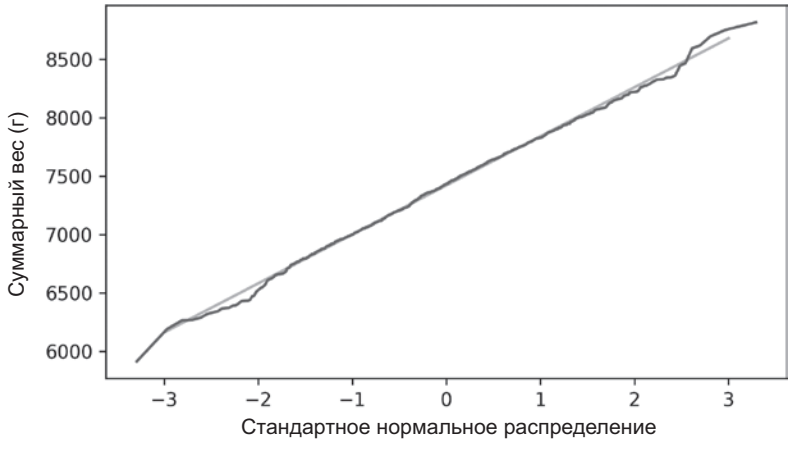


График нормальной вероятности выглядит как прямая линия, а значит, суммы следуют нормальному распределению. И это еще не все: если нам известны параметры двух распределений, мы можем вычислить параметры распределения суммы. Следующий метод показывает, как это сделать:

```
%%add_method_to Normal

def __add__(self, other):
    """Распределение суммы двух нормальных распределений."""
    return Normal(self.mu + other.mu, self.sigma2 + other.sigma2)
```

У распределения суммы среднее значение представляет собой сумму средних значений, а дисперсия — сумму дисперсий. Теперь, после того как определен специальный метод `__add__`, мы можем использовать оператор `+` для «сложения» двух распределений, то есть для вычисления распределения их суммы:

```
dist_sum = dist_male + dist_female
dist_sum
```

```
Normal(7412.328767123288, 192843.8926940639)
```

Чтобы подтвердить правильность полученного результата, воспользуемся следующим методом, который строит график аналитической ИФР нормального распределения:

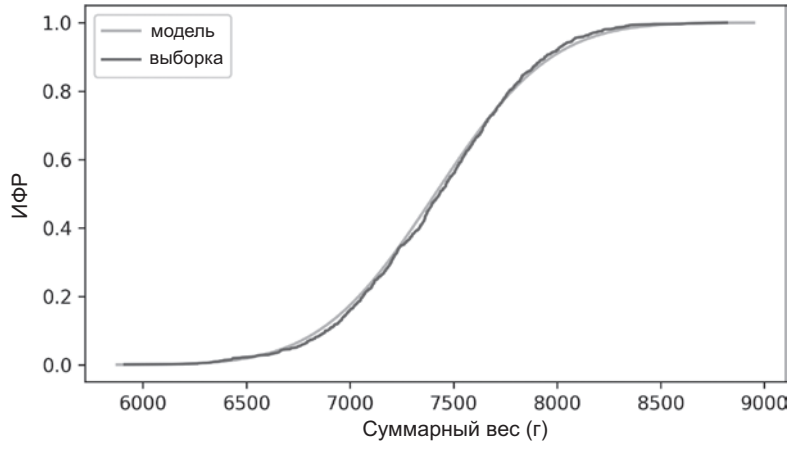
```
%%add_method_to Normal

def plot_cdf(self, n_sigmas=3.5, **options):
    mu, sigma = self.mu, np.sqrt(self.sigma2)
    low, high = mu - n_sigmas * sigma, mu + n_sigmas * sigma
    xs = np.linspace(low, high, 101)
    ys = norm.cdf(xs, mu, sigma)
    plt.plot(xs, ys, **options)
```

Вот графики аналитической и эмпирической ИФР суммы случайных выборок:

```
dist_sum.plot_cdf(**model_options)
Cdf.from_seq(sample_sum).plot(label="выборка")

decorate(xlabel="Суммарный вес (г)", ylabel="ИФР")
```



Выглядит так, что параметры, которые мы вычислили, верны. Это подтверждает, что можно сложить два нормальных распределения, суммировав их средние значения и дисперсии.

Как следствие, если извлечь n значений из нормального распределения и сложить их, то распределение суммы также будет нормальным. Чтобы показать это, начнем с того, что сгенерируем 73 значения из распределения веса самцов и сложим их. В следующем цикле эта процедура повторяется 1001 раз, так что результатом является выборка из распределения сумм:

```
n = len(weights_male)
sample_sums_male = [dist_male.sample(n).sum() for i in range(1001)]
n
```

```
73
```

Следующий метод создает объект `Normal`, представляющий распределение сумм. Чтобы вычислить его параметры, умножим среднее значение и дисперсию на n:

```
%%add_method_to Normal

def sum(self, n):
    """Распределение суммы n значений."""
    return Normal(n * self.mu, n * self.sigma2)
```

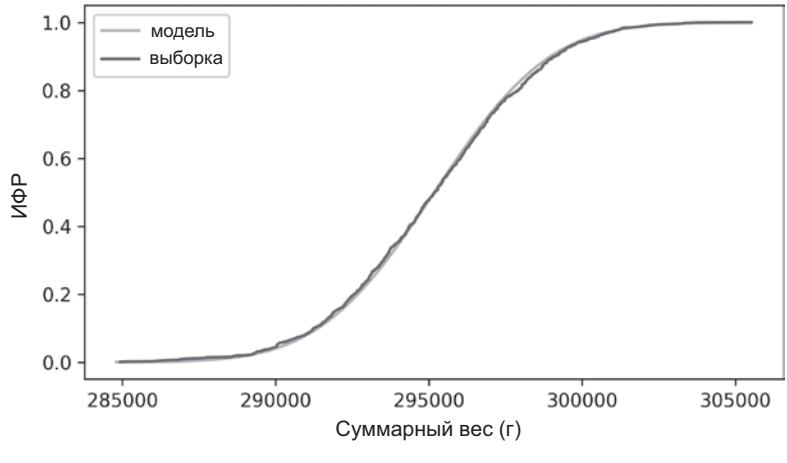
Вот распределение суммы n значений веса:

```
dist_sums_male = dist_male.sum(n)
```

А вот как оно соотносится с эмпирическим распределением случайной выборки:

```
dist_sums_male.plot_cdf(**model_options)
Cdf.from_seq(sample_sums_male).plot(label="выборка")

decorate(xlabel="Суммарный вес (г)", ylabel="ИФР")
```



Аналитическое распределение соответствует распределению выборки, что подтверждает верность метода `sum`. Таким образом, если составить выборку из `n` измерений, можно вычислить распределение их суммы.

Распределение выборочных средних

Если можно вычислить распределение выборочной суммы, можно вычислить и распределение выборочного среднего. Для этого воспользуемся третьим свойством нормального распределения: в результате умножения или деления на константу получается нормальное распределение. Следующие методы показывают, как можно вычислить параметры распределения произведения или частного:

```
%%add_method_to Normal

def __mul__(self, factor):
    """Умножение на константу."""
    return Normal(factor * self.mu, factor**2 * self.sigma2)
```

```
%%add_method_to Normal

def __truediv__(self, factor):
    """Деление на константу."""
    return self * (1/factor)
```

Чтобы вычислить распределение произведения, мы умножаем среднее значение на значение `factor`, а дисперсию — на квадрат `factor`. Это свойство можно использовать для вычисления распределения выборочных средних:

```
dist_mean_male = dist_sums_male / n
```

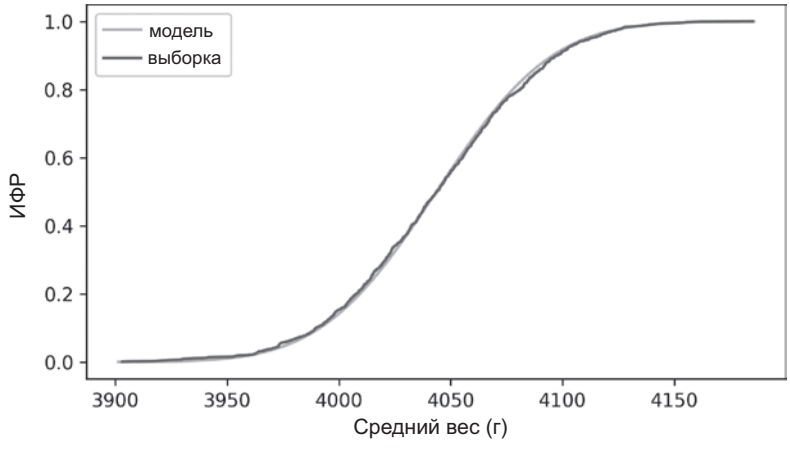
Чтобы убедиться в правильности полученного результата, вычислим также средние значения случайных выборок:

```
sample_means_male = np.array(sample_sums_male) / n
```

И сравним нормальную модель с эмпирической ИФР выборочных средних:

```
dist_mean_male.plot_cdf(**model_options)
Cdf.from_seq(sample_means_male).plot(label="выборка")

decorate(xlabel="Средний вес (г)", ylabel="ИФР")
```



Нормальная модель и результаты ресемплинга согласуются, это показывает, что вычислить распределение выборочных средних значений аналитически гораздо быстрее, чем проводить ресемплинг.

Теперь, зная выборочное распределение среднего, используем его для вычисления стандартной ошибки, которая представляет собой стандартное отклонение выборочного распределения:

```
standard_error = np.sqrt(dist_mean_male.sigma2)
standard_error
```

```
40.591222045992765
```

Полученный результат наводит на мысль о простом способе непосредственного вычисления стандартной ошибки — без предварительного вычисления выборочного распределения. Ранее мы сначала умножали дисперсию на `n`, а затем делили ее на `n**2`, что свелось в итоге к делению дисперсии на `n`, а значит, стандартного отклонения на квадратный корень из `n`.

Таким образом, рассчитаем стандартную ошибку выборочного среднего:

```
standard_error = weights_male.std() / np.sqrt(n) standard_error
```

```
40.59122204599277
```

Теперь рассмотрим еще один результат, который можно получить при помощи нормальных распределений, — распределение разностей.

Распределение разностей

Объединив все описанные в предыдущем разделе шаги, можно рассчитать распределение выборочных средних для веса самок пингвинов:

```
n = len(weights_female)
dist_mean_female = dist_female.sum(n) / n
dist_mean_female
```

```
Normal(3368.835616438356, 994.0498530055667)
```

Получив таким образом выборочные распределения для среднего веса самцов и самок пингвинов, вычислим распределение разностей. Следующий метод вычисляет распределение разностей выборочных значений из двух нормальных распределений:

```
%%add_method_to Normal

def __sub__(self, other):
    """Распределение разности."""
    return Normal(self.mu - other.mu, self.sigma2 + other.sigma2)
```

Как вы могли догадаться, среднее значение разностей равно разности средних значений. Но как вы, возможно, и не ожидали, дисперсия разностей — это не разность дисперсий, а их сумма! Чтобы понять почему, представьте, что мы выполняем вычитание в два этапа.

- Если изменить знак второго распределения, среднее значение также сменит знак, но дисперсия останется такой же.
- Далее, если добавить первое распределение, дисперсия суммы будет равна сумме дисперсий.

Если у вас еще остались сомнения, давайте их развеем. Вот аналитическое распределение разностей:

```
dist_diff_means = dist_mean_male - dist_mean_female
dist_diff_means
```

```
Normal(674.6575342465753, 2641.697160192656)
```

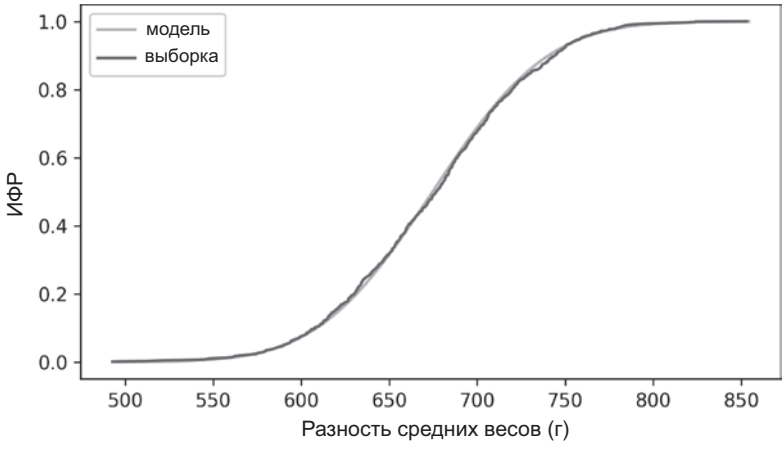
А вот случайная выборка разностей:

```
sample_sums_female = [dist_female.sample(n).sum() for i in range(1001)]
sample_means_female = np.array(sample_sums_female) / n
sample_diff_means = sample_means_male - sample_means_female
```

Следующий рисунок демонстрирует эмпирическую ИФР случайной выборки и аналитическую ИФР нормального распределения:

```
dist_diff_means.plot_cdf(**model_options)
Cdf.from_seq(sample_diff_means).plot(label="выборка")

decorate(xlabel="Разность средних весов (г)", ylabel="ИФР")
```



Графики совпадают, что подтверждает правильность найденного нами распределения разностей. С помощью полученного распределения можно вычислить доверительный интервал разности весов. Найдем функцию, обратную ИФР:

```
%%add_method_to Normal

def ppf(self, xs):
    sigma = np.sqrt(self.sigma2)
    return norm.ppf(xs, self.mu, sigma)
```

5-й и 95-й процентили образуют 90%-ный доверительный интервал:

```
ci90 = dist_diff_means.ppf([0.05, 0.95])
ci90
```

```
array([590.1162635, 759.19880499])
```

С помощью случайной выборки приходим к похожим результатам:

```
np.percentile(sample_diff_means, [5, 95])
```

```
array([589.01470284, 760.1276391 ])
```

Аналитический метод быстрее, чем ресемплинг, и он является детерминированным, то есть не подверженным случайности.

Однако все, что мы делали до сих пор, основывалось на допущении, что распределение результатов измерений нормальное. Однако оно не всегда такое — на самом деле распределение реальных данных никогда не бывает полностью нормальным. Но даже когда оно таким не является, для суммы большого количества выборочных значений оно часто близко к нормальному. В этом сила центральной предельной теоремы.

Центральная предельная теорема

Как вы узнали из предыдущих разделов, если сложить значения, извлеченные из нормальных распределений, то распределение их суммы также будет нормальным. Большинство других распределений не обладает этим свойством. Например, если суммировать значения, полученные из экспоненциального распределения, распределение полученной суммы не будет экспоненциальным.

Но многие распределения обладают следующим свойством. Если сгенерировать `n` значений, а затем сложить их, распределение суммы будет сходиться к нормальному по мере увеличения `n`. Точнее, если исходное распределение имеет среднее значение `m` и дисперсию `s2`, то распределение суммы сходится к нормальному распределению со средним значением `n * m` и дисперсией `n * s2`.

Такая закономерность называется центральной предельной теоремой, ЦПТ (Central Limit Theorem, CLT). ЦПТ — один из наиболее полезных инструментов статистического анализа, но с некоторыми оговорками.

- Значения должны быть получены из одного и того же распределения (хотя это требование может быть ослаблено).
- Значения должны быть получены независимо друг от друга. Если они коррелируют, ЦПТ неприменима (хотя и может работать, если корреляция не слишком сильна).
- Значения должны быть получены из распределения с конечными средним значением и дисперсией. Таким образом, ЦПТ неприменима к некоторым распределениям с «длинным хвостом».

Центральная предельная теорема объясняет широкое распространение нормального распределения в природе. Многие характеристики живых организмов подвержены влиянию генетики и факторов окружающей среды, воздействие которых аддитивно. Характеристики, которые мы измеряем, являются суммой большого числа малых воздействий, поэтому их распределение, как правило, является нормальным.

Чтобы познакомиться с тем, как работает центральная предельная теорема, а также с обстоятельствами, в которых она не работает, проведем несколько экспериментов, начав с экспоненциального распределения. В цикле ниже генерируются выборки из экспоненциального распределения, которые затем суммируются и помещаются в словарь. В нем каждому размеру выборки `n` сопоставляется список из 1001 суммы:

```
lam = 1
df_sample_expo = pd.DataFrame()
for n in [1, 10, 100]:
    df_sample_expo[n] = [np.sum(np.random.exponential(lam, n))
                         for _ in range(1001)]
```

Вот средние значения для каждого списка сумм:

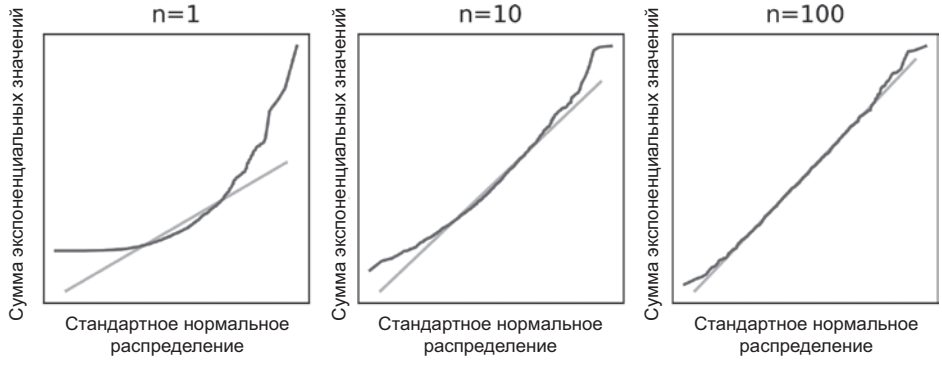
```
df_sample_expo.mean()
```

```
1        1.010255
10       9.993695
100    100.416396
dtype: float64
```

У этого распределения среднее значение равно 1. Поэтому если сложить 10 значений, среднее значение суммы будет близко к 10, а если сложить 100 значений, то среднее значение суммы будет близко к 100.

На следующем рисунке показаны графики нормальной вероятности для трех списков сумм (определение функции `normal_plot_samples` приведено в Jupyter-блокноте этой главы):

```
normal_plot_samples(df_sample_expo, ylabel="Сумма экспоненциальных значений")
```



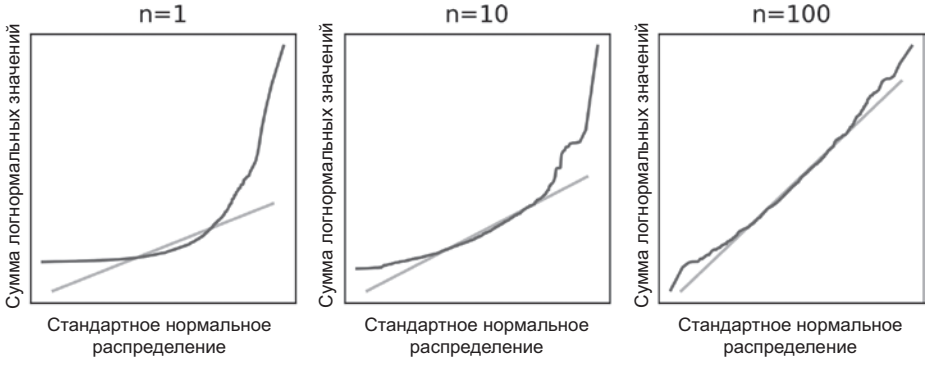
При `n=1` распределение суммы является экспоненциальным, поэтому график нормальной вероятности не выглядит как прямая линия. Но при `n=10` распределение суммы становится близким к нормальному, а при `n=100` оно уже почти неотличимо от нормального.

Для распределений не таких скошенных, как экспоненциальное, распределение суммы приближается к нормальному быстрее, то есть при меньших значениях `n`. Для более асимметричных распределений это происходит медленнее. В качестве примера рассмотрим суммы выборочных значений логнормального распределения:

```
mu, sigma = 3.0, 1.0
df_sample_lognormal = pd.DataFrame()
for n in [1, 10, 100]:
    df_sample_lognormal[n] = [
        np.sum(np.random.lognormal(mu, sigma, n)) for _ in range(1001)
    ]
```

Вот графики нормальной вероятности для тех же размеров выборки:

```
normal_plot_samples(df_sample_lognormal, ylabel="Сумма логнормальных значений")
```



При `n=1` нормальная модель плохо описывает распределение, и при `n=10` ситуация ненамного лучше. Даже при `n=100` хвосты распределения явно отклоняются от модельной прямой.

Среднее значение и дисперсия логнормального распределения конечны, а потому распределение суммы в итоге стремится к нормальному. Но для отдельных сильно скошенных распределений оно может не сходиться ни при каком доступном на практике размере выборки. А в некоторых случаях сходимости не происходит даже в теории.

Ограничения центральной предельной теоремы

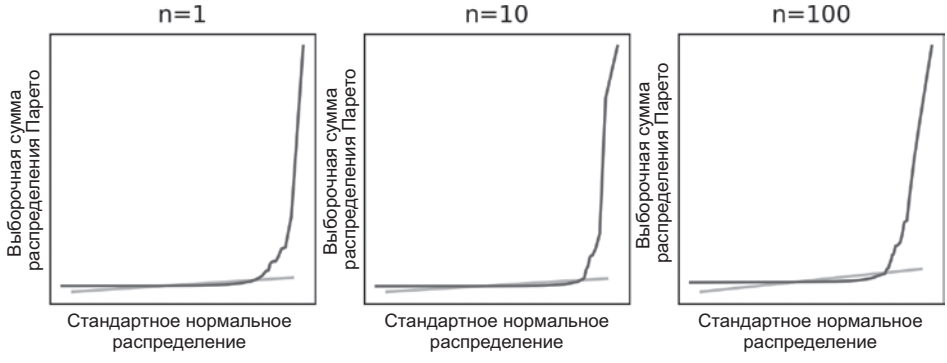
Распределение Парето еще более скошено, чем логнормальное. При некоторых значениях параметра распределение Парето не имеет конечного среднего значения или дисперсии. В подобных случаях центральная предельная теорема неприменима.

Чтобы это показать, сгенерируем выборку из распределения Парето с параметром `alpha=1`, имеющего бесконечные среднее значение и дисперсию:

```
alpha = 1.0
df_sample = pd.DataFrame()
for n in [1, 10, 100]:
    df_sample[n] = [np.sum(np.random.pareto(alpha, n)) for _ in range(1001)]
```

Вот как выглядят графики нормальной вероятности при разных размерах выборки:

```
normal_plot_samples(df_sample, ylabel="Выборочная сумма распределения Парето")
```



Даже при `n=100` распределение суммы совсем не похоже на нормальное.

Я уже упоминал, что центральная предельная теорема неприменима, если значения коррелируют. Чтобы в этом убедиться, воспользуемся функцией `generate_expo_correlated`. Она генерирует значения согласно экспоненциальному распределению, при этом последовательная корреляция, то есть корреляция между следующими один за другим элементами в полученной выборке, равна заданному значению `rho`. Функция определена в Jupyter-блокноте данной главы.

В следующем цикле создается объект `DataFrame` со столбцом для каждого размера выборки и 1001 суммой в каждом столбце:

```
rho = 0.8
df_sample = pd.DataFrame()
for n in [1, 10, 100]:
    df_sample[n] = [np.sum(generate_expo_correlated(n, rho))
                    for _ in range(1001)]
```

Вот графики нормальной вероятности для распределения этих сумм:

```
normal_plot_samples(df_sample, ylabel="Сумма коррелированных значений")
```



При `rho=0.8` существует сильная корреляция между последовательными элементами, и распределение суммы сходится медленно. Если также присутствует сильная корреляция между отдаленными элементами последовательности, оно может вообще не сойтись.

В предыдущем разделе было показано, что центральная предельная теорема работает, а в этом разделе — что происходит, когда она не работает. Теперь посмотрим, как ее можно использовать.

Применение центральной предельной теоремы

Чтобы понять, почему центральная предельная теорема очень полезна, вернемся к примеру из раздела «Проверка разницы средних» главы 9 на с. 166 — проверке наблюдаемой разницы в средней продолжительности беременности для первых и последующих детей. Снова воспользуемся данными NSFG — инструкции по их загрузке приведены в Jupyter-блокноте этой главы.

Воспользуемся функцией `get_nsfg_groups`, чтобы считать данные и разделить их на соответствующие первым (firsts) и последующим (others) детям:

```
from nsfg import get_nsfg_groups

live, firsts, others = get_nsfg_groups()
```

Как вы уже видели, первые дети в среднем рождаются немного позже: наблюдаемая разница составляет около 0.078 недели:

```
delta = firsts["prglngth"].mean() - others["prglngth"].mean()
delta
```

```
0.07803726677754952
```

Чтобы понять, можно ли появление такого различия объяснить случайностью, примем в качестве нулевой гипотезы, что среднее значение и дисперсия продолжительности беременности на самом деле одинаковы в обеих группах. Тогда их оценку можно получить, используя все случаи рождения живого ребенка (live):

```
all_lengths = live["prglngth"]
m, s2 = all_lengths.mean(), all_lengths.var()
```

Продолжительность беременности не описывается нормальным распределением, тем не менее с его помощью можно аппроксимировать выборочное распределение среднего.

Следующая функция принимает на вход последовательность значений и возвращает объект `Normal`, представляющий выборочное распределение среднего

некоторой выборки заданного размера n, полученной из нормального распределения с тем же средним и дисперсией, что и входные данные:

```
def sampling_dist_mean(data, n):
    mean, var = data.mean(), data.var()
    dist = Normal(mean, var)
    return dist.sum(n) / n
```

Вот нормальная аппроксимация выборочного распределения среднего веса первого ребенка в условиях нулевой гипотезы:

```
n1 = firsts["totalwgt_lb"].count()
dist_firsts = sampling_dist_mean(all_lengths, n1)
```

А вот выборочное распределение для второго и последующих детей:

```
n2 = others["totalwgt_lb"].count()
dist_others = sampling_dist_mean(all_lengths, n2)
```

Вычислим выборочное распределение их разности:

```
dist_diff = dist_firsts - dist_others
dist_diff
```

```
Normal(0.0, 0.003235837567930557)
```

Среднее значение равно 0, и это понятно: ведь если извлечь две выборки из одного и того же распределения, то разница в средних значениях в среднем ожидаемо будет равна 0. Дисперсия выборочного распределения, показывающая ожидаемую величину случайной изменчивости разности весов, равна 0.0032.

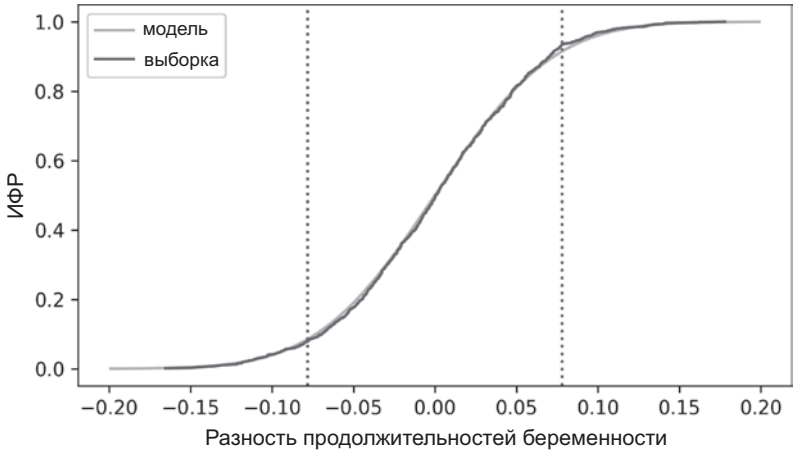
Чтобы убедиться, что это распределение аппроксимирует выборочное распределение, оценим его при помощи ресемплинга:

```
sample_firsts = [np.random.choice(all_lengths, n1).mean() for i in range(1001)]
sample_others = [np.random.choice(all_lengths, n2).mean() for i in range(1001)]
sample_diffs = np.subtract(sample_firsts, sample_others)
```

Вот эмпирическая ИФР разностей двух повторных выборок в сравнении с нормальной моделью. Вертикальные пунктирные линии обозначают наблюдаемую разность, взятую с положительным и отрицательным знаком:

```
dist_diff.plot_cdf(**model_options)
Cdf.from_seq(sample_diffs).plot(label="выборка")
plt.axvline(delta, ls=":")
plt.axvline(-delta, ls=":")

decorate(xlabel="Разность продолжительностей беременности", ylabel="ИФР")
```



В этом примере размеры выборок велики, а асимметрия данных умеренная, поэтому выборочное распределение хорошо аппроксимируется нормальным распределением. Следовательно, можно использовать нормальную ИФР для вычисления *p*-значения. Следующий метод высчитывает ИФР нормального распределения:

```
%%add_method_to Normal

def cdf(self, xs):
    sigma = np.sqrt(self.sigma2)
    return norm.cdf(xs, self.mu, sigma)
```

Вот вероятность разности величиной с `delta`, то есть площадь под правым хвостом выборочного распределения, в условиях нулевой гипотезы:

```
right = 1 - dist_diff.cdf(delta)
right
```

```
0.08505405315526993
```

А вот вероятность отрицательной разности размером с `-delta`, то есть площадь под левым хвостом:

```
left = dist_diff.cdf(-delta)
left
```

```
0.08505405315526993
```

Значения переменных `left` и `right` одинаковы, поскольку нормальное распределение симметрично. Сумма этих двух значений — вероятность, что разность по модулю равна величине `delta`:

```
left + right
```

```
0.17010810631053985
```

Итоговое *p*-значение, равное 0.170, согласуется с оценкой, вычисленной при помощи ресемплинга в разделе «Проверка разницы средних» главы 9 на с. 166.

Способ, которым мы вычислили это *p*-значение, аналогичен ***t*-критерию Стьюдента для независимых выборок** (independent sample t-test). В библиотеке SciPy определена функция `ttest_ind`, которая принимает на вход две выборки и вычисляет *p*-значение для различия в их средних:

```
from scipy.stats import ttest_ind

result = ttest_ind(firsts["prglngth"], others["prglngth"])
result.pvalue
```

```
0.16755412639415004
```

При большом объеме выборок результат *t*-критерия близок к тому, который мы рассчитали при помощи нормальных распределений. *t*-критерий называется так потому, что в нем вместо нормального распределения используется ***t*-распределение**. *t*-распределение, как вы узнаете в следующем разделе, также пригодно для проверки статистической значимости корреляций.

Критерий корреляции

В разделе «Проверка значимости корреляции» главы 9 на с. 170 для проверки значимости корреляции между весом новорожденного и возрастом матери мы использовали критерий перестановок (permutation test) и обнаружили, что она статистически значима, а *p*-значение меньше 0.001.

Теперь то же самое можно проделать аналитически. Для этого нужно выполнить следующие шаги: сначала сгенерировать две выборки размером `n` из нормальных распределений, далее вычислить коэффициент корреляции Пирсона `r`, а затем преобразовать (transform) его с помощью функции ниже:

```
def transform_correlation(r, n):
    return r * np.sqrt((n - 2) / (1 - r**2))
```

Преобразованный коэффициент корреляции следует *t*-распределению с параметром `n-2`. Чтобы посмотреть, как оно выглядит, нам потребуется следующая функция, которая генерирует некоррелированные выборки из стандартного нормального распределения:

```
def generate_data(n):
    """Некоррелированные последовательности из стандартного нормального
       распределения."""
    xs = np.random.normal(0, 1, n)
    ys = np.random.normal(0, 1, n)
    return xs, ys
```

А следующая функция вычисляет их корреляцию:

```
def correlation(data):
    xs, ys = data
    return np.corrcoef(xs, ys)[0, 1]
```

В следующем цикле генерируется множество пар выборок, вычисляется их коэффициент корреляции и полученный результат сохраняется в списке:

```
n = 100
rs = [correlation(generate_data(n)) for i in range(1001)]
```

Далее вычислим преобразованные коэффициенты корреляции:

```
ts = transform_correlation(np.array(rs), n)
```

Чтобы проверить, соответствуют ли значения в массиве `ts` *t*-распределению, воспользуемся следующей функцией. В ней создается объект, представляющий ИФР *t*-распределения:

```
from scipy.stats import t as student_t

def make_student_cdf(df):
    """Расчет ИФР для t-распределения Стьюдента."""
    ts = np.linspace(-3, 3, 101)
    ps = student_t.cdf(ts, df=df)
    return Cdf(ps, ts)
```

Параметр *t*-распределения обозначается как `df`, то есть «степени свободы» (degrees of freedom). На следующем рисунке показана ИФР *t*-распределения с параметром `n-2`, а также эмпирическая ИФР преобразованных коэффициентов корреляции:

```
make_student_cdf(df=n - 2).plot(**model_options)

cdf_ts = Cdf.from_seq(ts)
cdf_ts.plot(метка="случайные выборки")

decorate(xlabel="Преобразованный коэффициент корреляции", ylabel="ИФР")
```

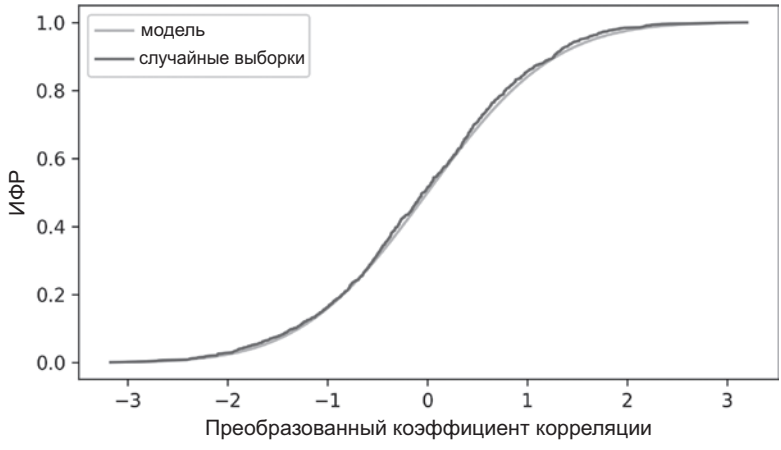


График показывает, что если сгенерировать некоррелированные выборки из нормального распределения, то преобразованное значение их коэффициента корреляции будет следовать *t*-распределению.

Если извлечь выборки из других распределений, то их преобразованные коэффициенты корреляции не будут в точности соответствовать *t*-распределению, но будут сходиться к *t*-распределению по мере увеличения размера выборки. Посмотрим, применимо ли это утверждение к корреляции возраста матери и веса новорожденного. Из объекта `DataFrame` данных о живых новорожденных выберем строки с действительными (valid) значениями:

```
valid = live.dropna(subset=["agepreg", "totalwgt_lb"])
n = len(valid)
n
```

```
9038
```

Фактический коэффициент корреляции составляет около 0,07:

```
data = valid["agepreg"].values, valid["totalwgt_lb"].values
r_actual = correlation(data)
r_actual
```

```
0.0688339703541091
```

Как вы узнали из раздела «Проверка значимости корреляции», нулевую гипотезу можно смоделировать при помощи перестановки выборочных значений:

```
def permute(data):
    """Перетасовка значений x."""
    xs, ys = data
    new_xs = xs.copy()
    np.random.shuffle(new_xs)
    return new_xs, ys
```

Если сгенерировать множество перестановок и посчитать их корреляции, мы получим выборку из распределения корреляций в условиях нулевой гипотезы:

```
permuted_corrs = [correlation(permute(data)) for i in range(1001)]
```

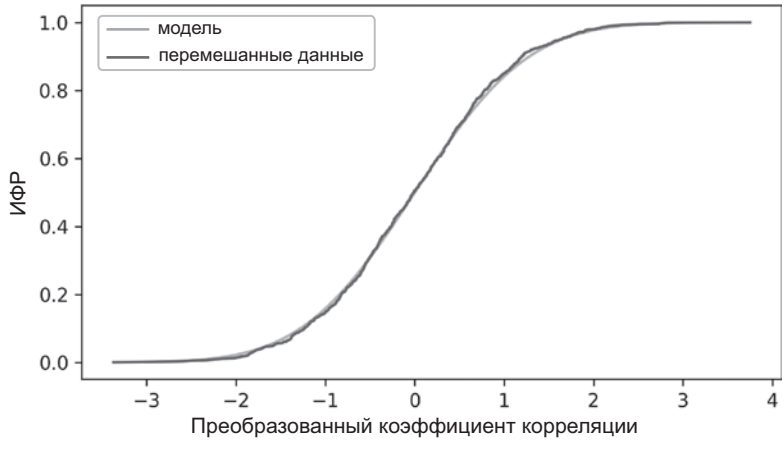
Далее можно вычислить преобразованные корреляции:

```
ts = transform_correlation(np.array(permuted_corrs), n)
```

На следующем рисунке показана эмпирическая ИФР, полученная для последовательности `ts`, вместе с ИФР *t*-распределения с параметром `n-2`:

```
make_student_cdf(n - 2).plot(**model_options)
Cdf.from_seq(ts).plot(label="перемешанные данные")

decorate(xlabel="Преобразованный коэффициент корреляции", ylabel="ИФР")
```



Модель хорошо согласуется с эмпирическим распределением, это означает, что ее можно использовать для вычисления *p*-значения для наблюдаемой корреляции. Сначала преобразуем коэффициент корреляции:

```
t_actual = transform_correlation(r_actual, n)
```

Теперь используем ИФР *t*-распределения для вычисления вероятности того, что при верности нулевой гипотезы *p*-значение будет равно величине `t_actual`:

```
right = 1 - student_t.cdf(t_actual, df=n - 2)
right
```

```
2.861466619208386e-11
```

Также рассчитаем вероятность получить отрицательное значение, равное `-t_actual`:

```
left = student_t.cdf(-t_actual, df=n – 2)
left
```

```
2.8614735536574016e-11
```

Сумма этих двух вероятностей равна вероятности того, что коэффициент корреляции равен величине `r_actual`, независимо от знака:

```
left + right
```

```
5.722940172865787e-11
```

Библиотека SciPy предоставляет функцию, которая выполняет те же вычисления и возвращает *p*-значение наблюдаемой корреляции:

```
from scipy.stats import pearsonr

corr, p_value = pearsonr(*data)
p_value
```

```
5.722947107314431e-11
```

Результаты практически одинаковые.

Используя ресемплинг, мы пришли к выводу, что *p*-значение меньше 0.001, но без очень большого количества повторных выборок не могли сказать, насколько меньше. Аналитические методы позволяют быстро вычислять такие небольшие *p*-значения.

Однако на практике это может и не требоваться. Как правило, *p*-значение меньше 0.001 говорит о том, что наблюдаемый эффект вряд ли является случайным. Но знать, насколько именно эта случайность маловероятна, вовсе не обязательно.

Критерий хи-квадрат

В разделе «Проверка пропорций» главы 9 на с. 173 мы проверяли, является ли игральная кость поддельной, отталкиваясь от множества наблюдаемых (observed) исходов:

```
from empiricaldist import Hist

qs = np.arange(1, 7)
freqs = [8, 9, 19, 5, 8, 11]
observed = Hist(freqs, qs)
observed.index.name = "исход"
observed
```

исход	частота
1	8
2	9
3	19
4	5
5	8
6	11

Сначала мы вычисляли ожидаемую (expected) частоту всех исходов (outcomes):

```
num_rolls = observed.sum()
outcomes = observed.qs
expected = Hist(num_rolls / 6, outcomes)
```

Затем использовали следующую функцию для вычисления статистики хи-квадрат:

```
def chi_squared_stat(observed, expected):
    diffs = (observed - expected) ** 2
    ratios = diffs / expected
    return np.sum(ratios.values.flatten())
```

```
observed_chi2 = chi_squared_stat(observed, expected)
```

Статистика хи-квадрат широко используется для такого рода данных, поскольку их выборочное распределение при нулевой гипотезе сходится к распределению, которое легко вычислить, — неслучайно оно называется **распределением хи-квадрат**. Чтобы понять, как оно выглядит, воспользуемся следующей функцией, которая моделирует (симулирует) бросание неподдельной игральной кости:

```
def simulate_dice(observed):
    n = np.sum(observed)
    rolls = np.random.choice(observed.qs, n, replace=True)
    hist = Hist.from_seq(rolls)
    return hist
```

В цикле ниже моделирование и последующее вычисление статистики хи-квадрат результатов повторяются существенное количество раз:

```
simulated_chi_squared = [
    chi_squared_stat(simulate_dice(observed), expected) for i in range(1001)
]
cdf_simulated = Cdf.from_seq(simulated_chi_squared)
```

Чтобы проверить, следуют ли полученные результаты распределению хи-квадрат, применим следующую функцию, вычисляющую ИФР распределения хи-квадрат с параметром `df`:

```
from scipy.stats import chi2 as chi2_dist

def chi_squared_cdf(df):
    """Дискретная аппроксимация ИФР распределения хи-квадрат."""
    xs = np.linspace(0, 21, 101)
    ps = chi2_dist.cdf(xs, df=df)
    return Cdf(ps, xs)
```

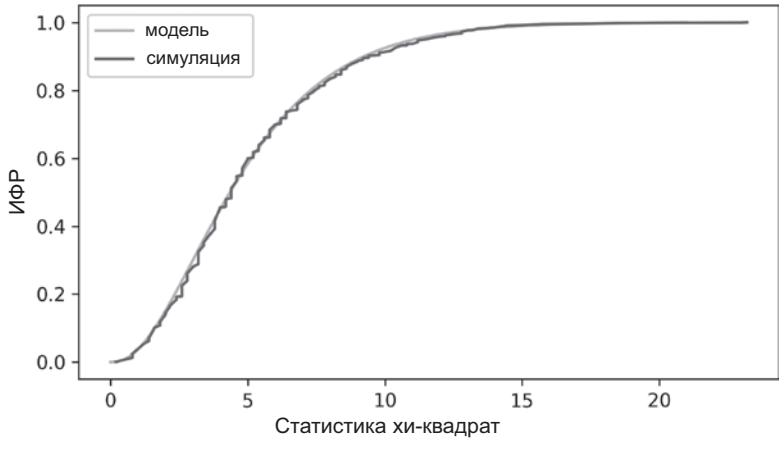
При `n` возможных исходах статистика хи-квадрат результатов симуляции должна соответствовать распределению хи-квадрат с параметром `n-1`:

```
n = len(observed)
cdf_model = chi_squared_cdf(df=n - 1)
```

Вот эмпирическая ИФР статистики хи-квадрат результатов симуляции, а также ИФР распределения хи-квадрат:

```
cdf_model.plot(**model_options)
cdf_simulated.plot(label="симуляция")

decorate(xlabel="Статистика хи-квадрат", ylabel="ИФР")
```



Модель хорошо соответствует результатам симуляции, поэтому ее можно использовать для вычисления вероятности того, что при условии верности нулевой гипотезы статистика хи-квадрат будет равна величине `observed_chi2`:

```
p_value = 1 - chi2_dist.cdf(observed_chi2, df=n - 1)
p_value
```

```
0.04069938850404997
```

В библиотеке SciPy есть функция, которая выполняет эти же вычисления:

```
from scipy.stats import chisquare

chi2_stat, p_value = chisquare(f_obs=observed, f_exp=expected)
```

Результат очень близок к вычисленному нами *p*-значению:

```
p_value
0.040699388504049985
```

Преимущество статистики хи-квадрат заключается в том, что ее распределение в условиях нулевой гипотезы можно легко посчитать. Но в данном случае, возможно, она не самым лучшим образом оценивает разницу между наблюдаемыми и ожидаемыми результатами.

Вычислительные и аналитические методы

В этой книге основное внимание уделяется численным методам, таким как ресемплинг и перестановка. Они имеют ряд преимуществ перед аналитическими:

Простота объяснения и понимания

Например, одной из самых сложных тем на вводных занятиях по статистике является проверка гипотез. Многие студенты на самом деле не понимают, что такое *p*-значение. Мне кажется, что подход, который мы использовали в главе 9 — моделирование нулевой гипотезы и вычисление тестовой статистики, — лучше объясняет эту базовую идею.

Надежность и универсальность

Аналитические методы часто основываются на допущениях, которые на практике не соответствуют действительности. Вычислительные методы требуют меньшего количества допущений, и их легко адаптировать и расширять.

Возможность отладки

Аналитические методы часто похожи на черный ящик: вы вводите в них цифры, а они «выплевывают» результаты. Но в них легко допустить почти незаметные ошибки; правильность их результатов трудно проверить и сложно распознать проблему, когда результаты ошибочны. Вычислительные методы можно разрабатывать и тестировать пошагово, что облегчает проверку правильности результатов.

Но у них есть и свои недостатки:

- Они могут требовать больше вычислительных ресурсов.
- Методы случайной выборки, такие как ресемплинг, не всегда выдают одинаковые результаты, что затрудняет проверку их правильности.

Принимая во внимание все эти плюсы и минусы, я рекомендую придерживаться следующей процедуры:

1. Используйте вычислительные методы для разведочного анализа данных. Если ответ вас удовлетворяет, а время выполнения приемлемо, то на этом можно остановиться.
2. Если время выполнения оказывается неприемлемым, поищите способы оптимизации. Использование аналитических методов — один из таких возможных способов.
3. Если замена вычислительного метода аналитическим вас устраивает, используйте вычислительный метод в качестве основы для сравнения, проводя взаимопроверку результатов, полученных методами обоих типов.

Для многих практических задач время выполнения вычислительных алгоритмов не представляет проблемы, поэтому дальше первого шага идти не приходится.

Глоссарий

График нормальной вероятности (normal probability plot)

График сравнения наблюдаемых значений с квантилями нормального распределения, позволяющий увидеть, насколько точно данные соответствуют нормальному распределению.

t-критерий Стьюдента для независимых выборок (independent sample t-test)

Метод вычисления *p*-значения наблюдаемой разницы между средними значениями двух независимых групп.

t-распределение (t-distribution)

Распределение, используемое для моделирования выборочного распределения разности в средних значениях в условиях нулевой гипотезы равенства разности нулю, а также для моделирования выборочного распределения преобразованных коэффициентов корреляции.

Распределение хи-квадрат (chi-squared distribution)

Распределение, используемое для моделирования выборочного распределения статистики хи-квадрат.

Статистика хи-квадрат (chi-squared statistic)

Тестовая статистика, оценивающая величину различия между двумя дискретными распределениями.

Упражнения

Упражнение 14.1

В этой главе мы сравнивали вес самцов и самок пингвинов и вычисляли доверительный интервал для их разности. Теперь проделаем то же самое с длиной крыльев. Наблюдаемая разница составляет около 4.6 мм:

```
grouped = adelie.groupby("Sex")

lengths_male = grouped.get_group("MALE")["Flipper Length (mm)"]
lengths_female = grouped.get_group("FEMALE")["Flipper Length (mm)"]
observed_diff = lengths_male.mean() - lengths_female.mean()
observed_diff
```

```
4.616438356164366
```

С помощью функции `sampling_dist_mean` создайте объекты `Normal`, представляющие выборочные распределения средней длины крыльев в двух группах, обращая при этом внимание на несовпадение размеров групп. Затем вычислите выборочное распределение разности и 90%-ный доверительный интервал.

Упражнение 14.2

Пользуясь данными NSFG, мы вычисляли корреляцию между весом новорожденного и возрастом матери, а также с помощью t-распределения определяли p-значение. Теперь проделаем то же самое с весом новорожденного и возрастом отца, который указан в столбце `hpagelb`:

```
valid = live.dropna(subset=["hpagelb", "totalwgt_lb"])
n = len(valid)
n
```

```
8933
```

Наблюдаемая корреляция составляет около 0.065:

```
data = valid["hpagelb"].values, valid["totalwgt_lb"].values
r_actual = correlation(data)
r_actual
```

```
0.06468629895432174
```

Вычислите преобразованное значение коэффициента корреляции `t_actual`. Используйте ИФР t-распределения для вычисления p-значения — является ли эта корреляция статистически значимой? Проверьте полученные результаты с помощью функции `pearsonr` библиотеки SciPy.

Упражнение 14.3

В одном из упражнений главы 11 мы рассматривали гипотезу Триверса — Уилларда, которая предполагает, что у многих млекопитающих соотношение полов зависит от «состояния матери», то есть таких факторов, как возраст, физические параметры, состояние здоровья и социальный статус. Ряд исследований показал наличие этого эффекта у людей, но результаты неоднозначны.

В качестве иллюстрации и возможности попрактиковаться с критерием хи-квадрат посмотрим, существует ли связь между полом ребенка и семейным положением матери. В Jupyter-блокноте этой главы содержатся инструкции, которые помогут вам приступить к делу.

Упражнение 14.4

Метод, который мы использовали в этой главе для анализа разностей показателей в группах, может быть расширен для анализа «разности разностей» — распространенной экспериментальной схемы. В качестве примера используем данные из статьи 2014 года, в которой исследуются последствия вмешательства, направленного на смягчение гендерных стереотипов при распределении задач в студенческих инженерных командах.

До и после вмешательства студенты отвечали на вопросы анкеты, в которой их просили оценить свой вклад в каждую часть учебных проектов по семибалльной шкале.

До вмешательства студенты-мужчины выше оценили свое участие в задачах по программированию в рамках проектов, чем студентки-женщины: средний балл для мужчин составил 3.57 при стандартной ошибке (СО) 0.28, а средний балл для женщин — 1.91 при стандартной ошибке 0.32.

После вмешательства гендерный разрыв сократился: средний балл для мужчин составил 3.44 (СО 0.16), средний балл для женщин — 3.18 (СО 0.16).

1. Чтобы представить выборочные распределения оценок средних значений до и после вмешательства для студентов мужского и женского пола, создайте четыре объекта `Normal`. Поскольку у нас есть стандартные ошибки для оценок средних значений, то для получения параметров выборочных распределений нам не нужно знать размер выборки.
2. Рассчитайте выборочное распределение гендерного разрыва — разности средних баллов — до и после вмешательства.
3. Затем вычислите выборочное распределение разности разностей, то есть изменение размера разрыва. Вычислите 95%-ный доверительный интервал и *p*-значение.

Есть ли доказательства того, что после проведенного вмешательства гендерный разрыв сократился?

Об авторе

Аллен Дауни (Allen Downey) — почетный профессор Олин-колледжа и консультант, специализирующийся в области data science и байесовской статистики. Он является автором нескольких книг, в том числе «Think Python»[1] (O'Reilly, 2024), «Think Bayes»[2] (O'Reilly, 2021) и «Probably Overthinking It» (University of Chicago Press, 2023), а также блога о программировании и data science. Аллен — обладатель степени Ph.D. в области computer science Калифорнийского университета в Беркли, а также степеней бакалавра и магистра Массачусетского технологического института (MIT).

[1] Дауни Аллен Б. «Основы Python. Научитесь думать как программист».

[2] Дауни Аллен Б. «Байесовские модели. Байесовская статистика на языке Python».

Иллюстрация на обложке

Животное на обложке третьего издания «Думай как аналитик» — рыба-брызгун семейства *Toxotidae*. Рыбы этого семейства охотятся на наземных насекомых и мелких животных. Они сбивают жертву струей воды из своего особым образом устроенного рта. Семейство насчитывает семь видов, распространенных от Индии до Филиппин, Австралии и Полинезии.

У брызгуна приплюснутое с боков туловище, спина между спинным плавником и ртом — прямая. Рот способен выдвигаться вперед, нижняя губа более выступающая. Такая форма рта идеально приспособлена для охоты: узкая бороздка на нёбе позволяет рыбе плевать в жертву струей воды. Прижимая язык к бороздке и сокращая жабры, рыба извергает мощную струю воды длиной до пяти метров. Брызгуны начинают учиться стрелять водой, когда достигают в длину 2,5 см. Поначалу они не отличаются меткостью, поэтому охотятся небольшими стайками, но со временем набираются опыта.

Глаза брызгуна также играют большую роль в добывании пищи. У этой рыбы прекрасное зрение, и, целясь в добычу, она способна компенсировать преломление света при прохождении границы между воздухом и водой. Заметив жертву, брызгун поворачивает свой глаз так, чтобы ее изображение попадало на определенную часть сетчатки. Часто, если насекомое находится в пределах досягаемости, брызгун выпрыгивает из воды, чтобы схватить его ртом.

Многие из животных, изображенных на обложках книг издательства O'Reilly, находятся под угрозой исчезновения; их выживание важно для человечества.

Иллюстрация на обложке выполнена Карен Монтгомери (Karen Montgomery) по мотивам черно-белой гравюры из Дувра. Дизайн серии разработан Эди Фридманом (Edie Freedman), Элли Волкхаузен (Ellie Volckhausen) и Карен Монтгомери.