ОСНОВЫ СТАТИСТИЧЕСКОГО И ИССЛЕДОВАТЕЛЬСКОГО АНАЛИЗА ДАННЫХ НА PYTHON

Дальневосточный федеральный университет

А.О. Ковальчук, В.Ю. Капитан

ОСНОВЫ СТАТИСТИЧЕСКОГО И ИССЛЕДОВАТЕЛЬСКОГО АНАЛИЗА ДАННЫХ НА PYTHON

Учебное пособие

Владивосток

2021

© Ковальчук А.О., Капитан В.Ю., 2021

© Оформление. ФГАОУ ВО ДВФУ, 2021

ISBN 978-5-7444-5060-1

УДК 519.23(075.8)

ББК 22.172я73

*Работа выполнена при финансовой поддержке гос. задания
Министерства науки и высшего образования РФ №0657-2020-0005*

Рецензенты:

О.И. Дьяченко, канд. физ.-мат. наук, доц. каф.
теоретической и ядерной физики ШЕН ДВФУ;
И.П. Яровенко, канд. физ.-мат. наук, с.н.с. Института
прикладной математики ДВО РАН

Ковальчук, А.О. Основы статистического и исследовательского анализа данных на Python : учебное пособие / А.О. Ковальчук, В.Ю. Капитан. – Владивосток : Издательство Дальневосточного федерального университета, 2021. – 1 CD-ROM ; [135 с.]. – Загл. с титул. экр. – ISBN 978-5-7444-5060-1. – DOI <https://doi.org/10.24866/7444-5060-1>. – Текст. Изображения : электронные.

Материал пособия делится на 4 главы. В первой главе описывается роль анализа данных в современном мире. Во второй главе дается материал, посвященный практическому применению математической статистики и теории вероятностей при анализе данных на Python. В третьей главе приводится описание методов сбора и предобработки данных. В четвертой главе рассматриваются основные методы визуализации данных. Практические материалы и задания для студентов выполнены с использованием Google Colaboratory. Пособие знакомит студентов с базовыми понятиями и методами анализа данных, примерами их использования в задачах обработки, прогнозной аналитики и информационного поиска.

Предназначено для студентов, обучающихся по программе высшего образования естественнонаучных и физико-технических направлений подготовки.

Текстовое электронное издание

Минимальные системные требования:

процессор с частотой 1,3 ГГц (Intel, AMD); оперативная память 256 МБ,
свободное место на винчестере 335 МБ; Windows (XP; Vista; 7 и т.п.)

Программное обеспечение:

Acrobat Reader, Foxit Reader либо любой другой их аналог

Дальневосточный федеральный университет

690922, Приморский край, г. Владивосток, о. Русский, п. Аякс, 10.

Тел./факс: (423) 226-54-43

E-mail: dvfutip@yandex.ru, prudkoglyad.sa@dvfu.ru

Изготовитель CD-ROM:

Дальневосточный федеральный университет,

690922, Приморский край, г. Владивосток, о. Русский, п. Аякс, 10.

Подписано к использованию 2.07.2021 г.

Объем 6,90 Мб. Тираж 50 экз.

© Ковальчук А.О., Капитан В.Ю., 2021

© Оформление. ФГАОУ ВО ДВФУ, 2021

Оглавление

Предисловие	4
Глава 1 Роль анализа данных в современном мире	6
1.1 Анализ больших данных	8
1.2 Инструменты анализа данных	12
1.3 Инструменты Python для анализа данных	13
Глава 2 Основы статистического анализа	15
2.1 Описательная статистика	15
2.2 Элементы теории вероятности	31
2.3 Выводы	50
Глава 3 Сбор и предобработка данных	51
3.1 Получение данных	51
3.2 Регулярные выражения	58
3.3 Web Mining	63
3.4 Предобработка данных	77
3.5 Выводы	86
Глава 4 Визуализация данных	87
4.1 Pandas	88
4.2 Matplotlib	91
4.3 Seaborn	96
4.4 Plotly	107
4.5 Выводы	113
Заключение	114
Список литературы	117
Приложение А Линейная алгебра с NumPy	120

Предисловие

В основу учебного пособия положен цикл лекций и практических работ по дисциплинам «Специальные ИТ для анализа данных и расчетов», «Суперкомпьютерные вычисления и оптимизация в моделировании физических процессов», «Управление ИТ-проектами», «Цифровая обработка информации», читаемым в Дальневосточном федеральном университете. В нем в краткой и доступной форме излагаются основы статистического и исследовательского анализа данных. Анализ данных – широкое понятие. В самом общем смысле это исследования, связанные с обработкой специальными методами и алгоритмами многомерной системы данных, имеющей множество параметров. В процессе анализа исследователь формирует определённое представление о характере явления, описываемого этими данными. Анализ данных нельзя рассматривать только как обработку информации после ее сбора, ведь это прежде всего средство проверки гипотез и решения задач исследователя, работающего практически в любой области современного знания.

Материал учебного пособия делится на четыре главы. В первой главе описывается роль анализа данных в современном мире. Во второй главе дается материал, посвящённый практическому применению математической статистики и теории вероятностей при анализе данных на Python. Для иллюстрации основных понятий и определений использованы возможности библиотек Matplotlib, Plotly, Pandas, Seaborn, NumPy, SciPy. В третьей главе приводится описание методов сбора и предобработки данных. Рассмотрены основные объекты библиотеки Pandas: DataFrame и Series. Большое внимание уделено приёмам работы с регулярными выражениями с использованием библиотеки

re, парсингу страниц (библиотеки requests для отправления HTTP-запросов и BeautifulSoup для преобразования HTML-файла в древовидную структуру) и методам обработки данных в формате JSON. Описаны способы работы с пропусками в датасетах, в том числе удаление дубликатов и объединение таблиц. В четвертой главе рассматриваются основные методы визуализации данных с использованием библиотек Pandas, Matplotlib, Seaborn и Plotly.

Практические материалы и задания для студентов выполнены с использованием Google Colaboratory — бесплатной интерактивной облачной среды для работы с кодом от Google. Для успешного освоения материала, представленного в учебном пособии, необходимо владение основами линейной алгебры и математического анализа, а также базовыми навыками программирования на языке Python.

Данное учебное пособие предназначено для специалистов в различных областях математической статистики, анализа данных, информатики, теоретической и статистической физики, компьютерного моделирования, а также для аспирантов и студентов вузов естественнонаучных и физико-технических направлений подготовки.

Глава 1 Роль анализа данных в современном мире

«Большие данные будут формировать наше общество. Если эти технологии окажутся не совместимы с человеческими ценностями, то они нанесут непоправимый ущерб человечеству ...

Сейчас исторический момент, чтобы выбрать правильный путь и получать выгоды от возможностей, которые стоят перед нами».

Gerd Gigerenzer

Max Planck Institute for Human Development

С каждым днем источников данных становится все больше: социальные сети, метеорологическая статистика, финансовые операции, мобильные приложения, информация о местонахождении абонентов сетей мобильной сотовой связи и данные с устройств аудио- и видеорегистрации, но больше всего информации создается при помощи интернета вещей (Internet of things) — более 80% всех данных. Что же такое данные? Это факты, тексты, картинки, звуки, логи, фрагменты кода, то есть информация, с которой мы взаимодействуем, храним, передаем и обрабатываем.

Данные — сведения об объектах и явлениях окружающего мира, выраженные в форме, доступной для восприятия, передачи и хранения.

Каждую секунду человек производит и потребляет данные, смартфоны и компьютеры также генерируют и скачивают данные, обмениваются ими с серверами в интернете и т.д.

Подсчитано, что объём данных, собранных за пять тысячелетий с момента изобретения письма до 2003 г., составляет около пяти эксабайт. В наше время такое же количество

информации генерируется каждые два дня. По прогнозам IDC количество данных на планете будет как минимум удваиваться каждые два года.

Рис. 1.1. Прогноз IDC по росту объема мировых данных

Однако сами по себе «данные, как горная порода, – бесполезны без извлекающих золото специалистов и технологий». Сегодня залогом успеха любой компании становится активная работа с имеющейся информацией. Каждый день собирается огромное количество информации о пользователях, их действиях и поведении. Современные компании ищут специалистов, способных правильно подготовить и обработать данные, владеющих методами моделирования, анализа и интерпретации результатов.

Инструменты аналитики используются для решения текущих проблем и поиска управленческих стратегий для предотвращения сложностей в будущем. Для некоторых предприятий данные и аналитика становятся главной движущей силой и фундаментом цифровой трансформации. Данные, их обработка и глубокий анализ позволяют выявлять тенденции, искать слабые места, выдвигать гипотезы и делать достоверные прогнозы.

1.1 Анализ больших данных

В современном мире, где информация часто обновляется и поступает из разных источников, специалистам в сфере Data Science приходится работать с огромными массивами данных. Объем, разнообразие и скорость их поступления создают новые уникальные проблемы для анализа.

Большие данные — объемные, быстро растущие в количестве и многообразные по структуре и типам информации данные. Также под этим термином понимается совокупность технологий для работы с большими массивами данных.

Термин Big Data относится к наборам данных, размер которых превосходит возможности типичных баз данных по хранению, управлению и анализу информации. В бытовом смысле большими данными можно считать то, что не помещается в оперативную память вашего компьютера. Термин появился в 2008 г. благодаря статье редактора Nature Клиффорда Линча и сразу был подхвачен маркетологами. Всплеск интереса к большим данным во многом связан с ростом вычислительных мощностей, позволяющих их обрабатывать. В настоящее время за развитием технологий Big Data следят множество компаний. В 2011 г. большие данные уже использовались такими гигантами бизнеса как Hewlett-Packard, IBM, Microsoft. В 2015 г. доля компаний, использующих большие данные, составляла 17% в мире. Сегодня доля таких компаний превышает 50%.

Понятие Big Data является собирательным и используется для обозначения технологий, которые обеспечивают переход в новую эру цифровых коммуникаций и обработку данных.

Технологии Big Data – серия инструментов и методов обработки структурированных и неструктурированных данных огромных объемов и различных форматов. Данные технологии применяются для получения воспринимаемых человеком результатов, эффективных в условиях непрерывного прироста и распределения информации по многочисленным узлам вычислительной сети. Технологии Big Data сформировались еще в конце 2000-х годов в качестве альтернативы традиционным системам управления базами данных и решениям класса Business Intelligence. В настоящее время большинство крупнейших поставщиков информационных технологий для организаций в своих деловых стратегиях используют понятие «большие данные», а основные аналитики рынка информационных технологий посвящают концепции отдельные исследования.

Сегодня технологии Big Data активно используется в самых разных отраслях – от банковского до аграрного сектора.

– IT-компании по всему миру, являющиеся одним из крупнейших пользователей больших данных, используют их для оптимизации своей работы, повышения производительности труда сотрудников и минимизации расходов. Комбинируя технологии больших данных с технологиями искусственного интеллекта, IT-сектор постоянно внедряет инновации для поиска решений даже для самых сложных проблем.

– Образование больше не ограничивается физическими рамками учебной комнаты, в том числе из-за начавшейся в 2019 г. пандемии коронавируса, обучение постепенно переходит в онлайн. Данные помогают повысить успеваемость, вовремя назначив репетитора, или выстроить индивидуальную траекторию обучения.

– Использование данных с датчиков улучшает стратегии поставок и повышает качество продукции. Также большие данные помогают создать прозрачную инфраструктуру, тем самым

выявляя и прогнозируя слабые места, которые могут негативно повлиять на бизнес.

– С помощью прогнозной аналитики медицинские организации могут предоставлять индивидуальные услуги отдельным пациентам. Кроме того, носимые устройства для фитнеса, телемедицина, дистанционный мониторинг — все это благодаря большим данным помогает изменить жизнь к лучшему.

Данные становятся таким же важным фактором производства, как трудовые ресурсы и производственные активы. Благодаря аналитике больших данных компании оптимизируют продажи и логистику, лучше узнают клиентов и разрабатывают наиболее подходящие им предложения.

1.1.1 Характеристики больших данных

Анализ больших данных (Big Data) начинается с их сбора. Информацию получают отовсюду: с наших смартфонов, кредитных карт, программных приложений, автомобилей. Из-за разных форматов и путей возникновения Big Data отличаются рядом характеристик, определенных в описательной модели больших данных под названием 3V (V V V): Volume (объёмы данных), Velocity (скорость накопления и обработки данных) и Variety (разнообразие источников и типов данных). Набор признаков 3V (V V V) был создан Douglas Laney из компании Meta Group в 2001 г. с целью указания на равную значимость управления данными по всем трём аспектам. Рассмотрим приведенные характеристики подробнее:

1. Volume (объём). Огромные объёмы данных, которые организации получают из бизнес-транзакций, интеллектуальных

(IoT) устройств, промышленного оборудования, социальных сетей и других источников, нужно где-то хранить. Ещё сравнительно недавно это было существенной проблемой, но развитие систем хранения информации облегчило ситуацию и сделало хранение доступнее.

2. Velocity (скорость). Чаще всего этот пункт относится к скорости поступления данных в реальном времени. В более широком понимании характеристика объясняет необходимость высокоскоростной обработки из-за изменения темпов прироста информации (всплесков активности).

3. Variety (вариативность). Разнообразие больших данных проявляется в их форматах: структурированные данные из клиентских баз, неструктурированные текстовые, видео- и аудиофайлы, а также частично структурированная информация из нескольких источников. Если раньше данные можно было собирать только из электронных таблиц, то сегодня данные поступают как в формате электронных писем, так и голосовых сообщений.

В дальнейшем возникли интерпретации описательной модели Big Data с четырьмя V (добавлялась veracity – достоверность), пятью V (viability – жизнеспособность и value – ценность) и даже семью V (variability – способность к изменению и visualization – визуализация). Но компания IDC, например, интерпретирует именно четвёртое V как value (ценность), подчеркивая экономическую целесообразность обработки больших объёмов данных в соответствующих условиях.

1.2 Инструменты анализа данных

В области анализа данных и интерактивных научно-исследовательских расчетов с визуализацией результатов используется довольно большое количество предметно-ориентированных языков программирования и инструментов – как с открытым исходным кодом, так и коммерческих – например, Python, R, Matlab, Stata, SAS и другие.

Рынок компьютерных программ для статистического анализа данных характеризуется высокой конкуренцией, нередко случаи консолидации и поглощений компаний-разработчиков. Перед пользователями различных категорий встает вопрос выбора оптимального программного продукта для поиска верных ответов на существующие вопросы. Очевидно, что оптимальным является вариант, сочетающий в себе необходимые функциональные возможности, высокое качество работы и умеренную цену. При выборе программы для анализа данных следует учитывать следующие параметры:

- соответствие характеру решаемых задач;
- объем обрабатываемых данных;
- требования, предъявляемые к квалификации пользователя (уровень знаний в области статистики);
- имеющееся в наличии компьютерное оборудование.

Сравнительно недавнее появление улучшенных библиотек (прежде всего, Pandas – начало разработки в 2008 г.) для Python сделало его серьезным конкурентом в решении задач манипулирования данными. В сочетании с достоинствами Python как универсального языка программирования это делает его отличным выбором для анализа данных, поэтому на сегодняшний

день Python считается одним из наиболее востребованных языков в Data Science.

Python приобретает всё большую популярность не только среди аналитиков данных. По данным одного из самых известных рейтингов TIOBE Python с 2008 г. прочно удерживается в восьмерке наиболее популярных языков программирования. Google, Яндекс, Национальное управление по воздухоплаванию и исследованию космического пространства США (NASA) и многие другие известные компании активно используют Python. Проблему лицензирования программного обеспечения снимает то, что свободно распространяемые реализации языка Python существуют для всех популярных операционных систем: Linux, Mac OS, Windows, Android, iOS и др.

В настоящее время существуют две несовместимые между собой версии Python — Python 2 и Python 3. Однако в январе 2020 г. команда разработчиков Python 3 заявила, что прекращают поддержку Python 2. Facebook, Instagram и многие другие крупные компании уже перенесли большую часть своего кода с Python 2 на Python 3. На настоящий момент последняя доступная для скачивания версия Python – Python 3.9.5.

1.3 Инструменты Python для анализа данных

Для анализа и обработки данных в материалах данного учебного пособия будет использоваться язык Python, так как он является оптимальным средством работы с данными благодаря огромному количеству полезных библиотек, написанных для него.

Библиотеки — это модули Python. То есть это самостоятельные, объединённые технически и логически, именованные части Python кода.

В Python предусмотрено большое количество библиотек для хранения и обработки данных. Рассмотрим наиболее популярные из них:

- Pandas — обработка структурированных данных;
- NumPy — поддержка больших многомерных массивов и матриц, вместе с большой библиотекой высокоуровневых математических функций для операций с этими массивами;
- SciPy — использование математических алгоритмов и функций для выполнения научных и инженерных расчётов;
- Matplotlib — визуализация данных в виде двумерных графиков (включая 3D-представления).

В данном пособии будут последовательно разобраны и изучены инструменты Python, необходимые для сбора и предобработки данных, визуализации, статистического и исследовательского анализа.

Глава 2 Основы статистического анализа

В процессе анализа данных возникает множество гипотез, которые удобно проверять статистическими методами. В этой главе будут рассмотрены основы статистики и теории вероятности.

2.1 Описательная статистика

2.1.1 Категориальные и количественные, непрерывные и дискретные переменные

При работе со статистикой важно понимать используемую терминологию, включая количественные и категориальные переменные и то, как они отличаются.

Количественные признаки можно сравнить между собой на «больше или меньше», а категориальные признаки так сравнить не получится. Категориальная переменная не имеет численного или количественного значения, а просто описывает качество или характеристику чего-то. **Категориальная (качественная)** переменная принимает некоторое значение из конечного числа значений (ограниченного набора). Цифры, используемые в категориальных или качественных данных, обозначают качество, а не измерение или количество. Если в качестве значения роста указать «низкий», «средний» и «высокий», то это будет уже категориальный признак и качественная характеристика роста. Категориальный признак принимает одно из ограниченного

набора значений. При необходимости и для удобства качественные признаки можно закодировать. Так значение «низкий» получит код 1, «средний» — 2, «высокий» — 3.

Количественные переменные измеряются и выражаются численно, имеют числовое значение и могут использоваться в расчетах (поэтому иначе их называют числовые переменные). **Количественная** переменная принимает числовое значение в диапазоне. Количественные переменные подразделяются на:

- **Непрерывные**, которые могут принимать любое численное значение.

- **Дискретные**, которые могут принимать строго определённые значения (переменные, в которых значения могут быть получены путем подсчета).

Дискретная переменная принимает независимые значения, тогда как непрерывная переменная принимает любое значение в заданном диапазоне.

2.1.2 Гистограммы

Для описания величин удобно использовать наглядное представление данных. График, который показывает, сколько раз в наборе данных встречается то или иное значение, называется гистограммой. **Гистограмма** объединяет числовые значения по диапазонам, то есть считает частоту значений в пределах каждого интервала. Её построение подобно работе метода `value_counts()`, подсчитывающего количество уникальных значений в списке. Однако `value_counts()` группирует строго одинаковые величины и хорош только для подсчёта частоты в списках с категориальными переменными.

В Pandas гистограмму строит специальный метод **hist()** (от англ. histogram — «гистограмма»). Применять его можно к списку или к столбцу датафрейма: во втором случае название столбца передаётся в параметре. Метод **hist()** находит в наборе чисел минимальное и максимальное значения, а полученный диапазон делит на области или «корзины» — **bins** (от англ. «корзины, вёдра»). Затем **hist()** считает, сколько значений попало в каждую корзину и отображает это на графике.

2.1.3 Гистограмма частот для непрерывной переменной

Одним из подходов в визуализации значений непрерывных переменных является разделение множества значений на интервалы и подсчёт количества значений, попадающих в каждый интервал.

Параметр **bins** определяет на сколько областей делить диапазон данных. По умолчанию таких «корзин» 10. В Pandas при построении гистограммы можно задавать не только количество интервалов - корзин, но и явно указывать их границы.

```
import matplotlib.pyplot as plt

data = [1.5, 2.3, 2.2, 3.8, 3.3, 3.9, 4.1, 4.3, 4.5, 4.7]
plt.hist(data, bins=[1.5, 2.3, 3.9, 4.1,
                    5]) # Края корзин. Длина nbins + 1 (левый
    ↪ край nbins и правый край последнего значения).
# Всегда один массив, даже если передаётся несколько наборов
    ↪ данных.
```

```
(array([2., 3., 1., 4.]),
 array([1.5, 2.3, 3.9, 4.1, 5. ]),
 <a list of 4 Patch objects>)
```

Однако, такой подход не может дать полное представление о значениях переменной, так как полученная гистограмма сильно зависит от того, как разбить множество значений на интервалы и какой выбрать шаг (ширину ячейки).

```
import plotly.express as px

df = px.data.tips()
fig = px.histogram(df, x="total_bill", nbins=20,
    ↪title="Разбиение на 20 корзин")
fig1 = px.histogram(df, x="total_bill", nbins=40,
    ↪title="Разбиение на 40 корзин")
fig.show()
fig1.show()
```

Разбиение на 20 корзин

Разбиение на 40 корзин

```
import pandas as pd
from pylab import *

flights = pd.read_csv('https://raw.githubusercontent.com/
↳datasciencefufu/course/main/data/flights.csv')
rcParams['figure.figsize'] = 18, 12

# Установим 4 различных ширины ячейки
for i, binwidth in enumerate([1, 5, 10, 15]):
    ax = plt.subplot(2, 2, i + 1)
```

```

ax.hist(flights['arr_delay'], bins=int(180 / binwidth),
        color='blue')

plt.xlim([-200,300])
ax.set_title('Гистограмма с шириной ячейки = %d' % binwidth,
             size=26)
ax.set_xlabel('Задержка рейса в мин', size=22)
ax.set_ylabel('Рейсы', size=22)

plt.tight_layout()
plt.show()

```

2.1.4 Гистограммы плотностей

Для того, чтобы решить недостаток разбиения на интервалы, применяется метод, отображающий частоту не высотой столбца в гистограмме, а его площадью. Площадь столбца находят как площадь прямоугольника: длину интервала умножают на высоту

столбца. Найденная площадь — частота непрерывной переменной, а высота столбца — **плотность частоты**. Гистограмма, использующая в качестве переменной плотность частоты, называется **плотностная гистограмма**.

Для того, чтобы оценить, сколько значений попало в интервал, берут два значения и ищут площадь плотностной гистограммы между ними. Полученное число и будет оценкой количества значений, попавших в интервал.

Плотность частоты для непрерывных переменных можно задавать не только прямоугольниками, как на гистограммах, но и кривыми функциями. Работает тот же принцип: площадь между двумя значениями пропорциональна частоте попадания значений в интервал между ними.

```
import seaborn as sns

rcParams['figure.figsize'] = 9, 7
sns.distplot(flights['arr_delay'], hist=True, kde=True,
             # Kernel density estimation (kde) рисуется в виде
             ↪ линии и часто повторяет форму гистограмм,
             # но дает больше информации о характере
             ↪ распределения
             bins=int(180 / 5), color='darkblue',
             hist_kws={'edgecolor': 'black'},
             kde_kws={'linewidth': 4});

plt.xlim([-100,200])
plt.title('Плотностная гистограмма задержек прибытия')
plt.xlabel('Delay (min)')
plt.ylabel('Density')
```

```
Text(0, 0.5, 'Density')
```

Кривая показывает график плотности, который по сути является гладкой версией гистограммы. Ось y соответствует плотности и гистограмма по умолчанию нормализована, так что она имеет тот же масштаб, что и график плотности.

2.1.5 Метрики локации данных

Познакомимся с характеристиками, которые помогут описать разброс данных в датасете.

Разброс данных — это понятие, которое используют, чтобы описать отклонения значений набора данных от среднего.

Такие характерные значения выборки как медиана и среднее значение также называют **метрики локации данных**. По медиане и среднему можно судить где примерно расположен набор данных на числовой оси. Для расчёта среднего значения берут все значения датасета — это наиболее полное использование информации при поиске метрики локации. Среднее значение называют **алгебраической метрикой локации**, медиану — **структурной метрикой локации**.

Такие характеристики, дающие представление о выборке, входят в **числовое описание данных**. В Python его получают методом `describe()` (от англ. «описать, охарактеризовать»). Результат отработки метода — список характерных значений:

```
data = sns.load_dataset("tips")
print(data.describe())
```

	total_bill	tip	size
count	244.000000	244.000000	244.000000
mean	19.785943	2.998279	2.569672
std	8.902412	1.383638	0.951100
min	3.070000	1.000000	1.000000
25%	13.347500	2.000000	2.000000
50%	17.795000	2.900000	2.000000
75%	24.127500	3.562500	3.000000
max	50.810000	10.000000	6.000000

Для представления о данных, недостаточно знать метрики локации, нужно ещё понимать, как данные разбросаны вокруг них. У структурной метрики локации есть структурные метрики разброса — **квартили** (см. Диаграмма размаха).

2.1.6 Средние величины данных

Существуют значения, характеризующие состояние выборки (набора данных, выбранных для проведения исследования). Чтобы примерно оценить типичные значения выборки, годятся среднее арифметическое и медиана.

Среднее арифметическое — сумма всех значений, деленная на их количество. Показывает общую тенденцию данных и описывает их одним числом.

Именно среднее арифметическое показывает какие примерно значения можно встретить в наборе с данными. Например, средний рост студентов.

Для быстрого расчета в NumPy пользуемся методом **mean()**:

```
import numpy as np

x = [1.67, 1.89, 1.90, 1.58, 1.57, 1.73, 1.75, 1.82, 1.50]
np.mean(x)
```

```
1.7122222222222223
```

Взвешенное среднее арифметическое — это среднее арифметическое для данных, где значения могут иметь разный вес, и некоторые значения должны сильнее влиять на результат, чем другие.

В данных могут быть значения, которые принимают величину существенно выше или ниже среднего и сильно отличаются от остальных — **выбросы**. Тогда вместо среднего арифметического лучше использовать медиану.

Медиана (median) — это число, которое делит упорядоченный ряд наблюдений пополам так, что одна половина располагается выше него, а другая половина — ниже.

В геометрии медиана делит фигуру на две равные по площади. В статистике она делит выборку пополам: в одной половине значения меньше медианного, в другой больше. Для определения медианы список обязательно должен быть отсортирован — либо по возрастанию, либо по убыванию.

Когда количество значений нечётное, медиана будет равна тому значению, которое оказалось ровно посередине отсортированного набора. Если же количество данных чётное, то медиана рассчитывается как среднее арифметическое двух соседних чисел в середине набора.

В NumPy для нахождения медианы есть метод **median()**:

```
np.median(x)
```

1.73

Рассмотрим пример, когда среднее значение некорректно характеризует данные, когда некоторые значения сильно выделяются среди большинства. Например, 5 сотрудников компании получают зарплату 100 000 рублей. Инвестор пригласил директора по развитию с зарплатой 300 000 рублей. На сколько процентов выросла средняя оплата труда в коллективе?

Зарплаты первоначального состава команды приведены в `old_staff`. В версии с директором по развитию — в `new_director`.

```
old_staff = pd.Series([100, 100, 100, 100, 100])
new_director = pd.Series([100, 100, 100, 100, 100, 300])
```

Вызвав метод `mean()`, рассчитаем среднее значение зарплаты в старом и новом коллективах:

```
old_staff_mean = old_staff.mean()
new_director_mean = new_director.mean()
```

Чтобы ответить, на сколько процентов выросла средняя зарплата в команде, поделим их друг на друга:

```
new_director_mean / old_staff_mean - 1
```

```
0.3333333333333335
```

Дробь в формуле показывает во сколько раз средняя зарплата в новой команде больше средней зарплаты в старой. Единицу отнимаем, чтобы определить, на какую долю выросла средняя зарплата после прихода директора по развитию. Результат выведем в процентах:

```
print('Зарплата выросла на {:.0%}'.format(new_director_mean /
↪old_staff_mean - 1))
```

Зарплата выросла на 33%

Вот так, ловким приёмом на работу нового высокооплачиваемого сотрудника средняя зарплата всех членов команды повышается на 33%.

Когда в выборке присутствуют выдающиеся значения (300 против 100), лучше использовать медиану. Применим `median()` к набору данных `new_director`

```
new_director_median = new_director.median()
print(new_director_median)
```

```
100.0
```

Значит, зарплатная медиана как была равна 100, так и останется после прихода директора по развитию. Это более объективная информация о зарплате сотрудников, чем среднее значение.

2.1.7 Дисперсия

Отклонение — это разность между конкретным значением и средним арифметическим. Отклонение показывает, насколько сильно конкретное значение отличается от среднего. Но считать отклонение для каждого значения отдельно не всегда удобно и значимо. Поэтому есть характеристики, которые опишут общее отклонение по всем данным.

Улучшенная метрика разброса — не просто среднее расстояние между значениями датасета и средним, а средний квадрат этого расстояния. Эта величина называется **дисперсия**, её находят по формуле: $\sigma = \frac{\sum(\mu - x_i)^2}{n}$, где греческая буква μ обозначает среднее арифметическое значение совокупности данных: $\mu = \frac{\sum(x_i)}{n}$.

Библиотека Numpy в Python содержит большую библиотеку высокоуровневых математических функций. Дисперсию рассчитывают методом **var()**:

```
variance = np.var(x)
print(variance)
```

0.01830617283950616

2.1.8 Стандартное отклонение

У дисперсии есть один небольшой недостаток: единица её измерения — это квадрат исходной величины. Чтобы вернуться к исходной единице измерения, из дисперсии извлекают квадратный

корень. Получившаяся величина называется **стандартным отклонением**: $\sigma = \sqrt{\frac{\sum(\mu - x_i)^2}{n}}$.

Оно характеризует разброс величин, показывает насколько значения в выборке отличаются от среднего арифметического.

Бывает, что именно стандартное отклонение позволяет понять природу распределения и определить насколько значения однородны. Чем **меньше** стандартное отклонение, тем значения конкретной характеристики **однороднее**. Чем **больше** стандартное отклонение, тем **сильнее разброс** значений признака. То есть если стандартное отклонение большое, то у нас много данных с отклонениями.

Стандартное отклонение находят методом `std()` из библиотеки Numpy:

```
standard_deviation = np.std(x)
print(standard_deviation)
```

0.1353003061323446

Если дисперсия известна заранее, можно применить метод `sqrt()` из библиотеки Numpy. Корень из дисперсии будет равен стандартному отклонению:

```
variance = 2.9166666666666665
standard_deviation = np.sqrt(variance)
print(standard_deviation)
```

1.707825127659933

Примечание

Для большинства распределений верно правило трёх стандартных отклонений или **правило трёх сигм**. Оно гласит — практически все значения (около 99%) находятся в промежутке: $(\mu - 3\sigma, \mu + 3\sigma)$. Это правило позволяет не только находить интервал, в который наверняка попадут практически все значения

интересующей нас переменной, но и искать значения вне этого интервала — часто их называют **выбросами**.

2.1.9 Корреляция данных

Взаимозависимость двух величин называют **корреляция**. Корреляция показывает насколько два ряда синхронно движутся между собой. График позволяет утверждать, что две величины явно взаимосвязаны, или **коррелируют**. В том случае, если существует прямая зависимость величин (например, чем больше одна, тем больше другая), то говорят, что корреляция **положительная**. В том случае, если существует обратная зависимость величин (чем больше одна, тем меньше другая), то говорят, что корреляция **отрицательная**. Численно взаимосвязь оценивается с помощью **коэффициента корреляции Пирсона**. Он принимает значения от - 1 до 1 и помогает определить, как сильно меняется одна величина при изменении другой.

— Если с ростом первой величины, растёт вторая, то коэффициент корреляции Пирсона — положительный. Если при изменении одной величины другая остаётся прежней, то коэффициент равен 0.

— Если рост одной величины связан с уменьшением другой, коэффициент отрицательный. Чем ближе коэффициент корреляции Пирсона к крайним значениям: 1 или - 1, тем сильнее взаимозависимость.

— Если значение близко к нулю, значит связь слабая, либо отсутствует вовсе. Бывает, что коэффициент нулевой не оттого, что связи между значениями нет, а потому что она является более сложной и нелинейной. Поэтому коэффициент корреляции такую связь не учитывает.

Коэффициент Пирсона находят методом `corr()`. Метод применяют к столбцу с первой величиной, а столбец со второй передают в параметре. Какая первая, а какая вторая — неважно:

```
print(data['column_1'].corr(data['column_2']))
print(data['column_2'].corr(data['column_1']))
```

```
print(data['total_bill'].corr(data['tip']))
print(data['tip'].corr(data['total_bill']))
```

```
0.6757341092113641
0.6757341092113641
```

Примечание

Совместное распределение для множества величин

Когда в задаче нужно найти попарные взаимосвязи величин, это можно сделать с помощью попарных диаграмм рассеяния. В Pandas такую задачу решают специальным методом:

```
pd.plotting.scatter_matrix(data)
```

Визуализировать диаграммы рассеяния можно используя метод `pairplot()` библиотеки Seaborn (см. Визуализация данных).

Помимо попарных диаграмм рассеяния, можно получить попарный коэффициент корреляции для всех величин. Это можно сделать с помощью матрицы корреляции:

```
data.corr()
```

	total_bill	tip	size
total_bill	1.000000	0.675734	0.598315
tip	0.675734	1.000000	0.489299
size	0.598315	0.489299	1.000000

Корреляция помогает определить с помощью каких параметров можно делать прогноз, а какие никакой

предсказательной силы не несут. При этом на диагонали матрицы корреляции всегда будут единицы: сравниваемые ряды полностью совпадают.

2.2 Элементы теории вероятности

2.2.1 Вероятность: эксперименты, элементарные исходы и события

Изучать теорию вероятности можно годами, но для решения большинства задач достаточно знания нескольких базовых определений.

Испытание (эксперимент) — это повторяемый опыт, который может закончиться **элементарными исходами**. Они так названы из-за своего простого устройства: исход либо случился, либо нет. Результат конкретного исхода или нескольких исходов называется **событием**. Например, при подбрасывании монетки выпала решка — это и есть событие.

В простейшем случае исходы не отличаются: вероятность каждого из них одинакова. Такие исходы называются **равновероятные**. С равной вероятностью выпадают орёл и решка при броске монеты, чётные и нечётные грани на игральном кубике, бутерброд маслом вверх и вниз.

Если провести эксперимент один раз, какой-то из исходов точно произойдёт. Вероятность такого события — 100%, или единица. Вероятности n элементарных исходов одинаковы и в сумме равны единице. Несложно посчитать, что тогда вероятность каждого исхода равна единице, делённой на n . Итак,

при проведении эксперимента с n элементарными исходами вероятность каждого исхода одинакова и равна $1/n$ (исходы равновероятны).

Рассмотрим на примере. Магазин шоколада “Конфишок” проводит розыгрыш. Для получения бесплатного набора конфет необходимо сделать репост записи с описанием конкурса в социальных сетях. Победитель будет выбран случайно. Найдем вероятность выигрыша:

```
reposts_media = 2693
probability_of_win = 1 / reposts_media
print("Вероятность выиграть бесплатный набор конфет: {:.4%}".
      ↪format(probability_of_win))
```

Вероятность выиграть бесплатный набор конфет: 0.0371%

Математически вероятность события равна отношению числа благоприятных событию исходов испытания к общему числу исходов испытания и обычно обозначается буквой P :

$$P = \frac{m}{N},$$

где m — это количество благоприятных исходов (например, количество победителей в лотерее), а N — общее количество исходов (общее количество людей, кто купил билеты, чтобы поучаствовать в лотерее).

Множество всех элементарных исходов эксперимента принято называть **вероятностным пространством**. Из него можно выделить подмножества, содержащие в себе некоторое количество элементарных исходов - события.

Невозможное событие — событие, которое не произойдёт никогда, вероятность его появления равна 0.

Достоверное событие — событие, которое точно произойдёт, вероятность его появления равна 1.

Вероятность появления других событий находится в промежутке от 0 до 1.

При сохранении условия равновероятности всех элементарных исходов **вероятность события** — количество исходов, входящих в это событие, делённое на общее количество исходов, то есть на размер вероятностного пространства.

Примечание

Закон больших чисел: чем больше раз повторяется эксперимент, тем ближе частота заданного на этом эксперименте события будет к его вероятности. Это правило можно использовать и в обратную сторону: если не известна вероятность какого-то события, но возможно много раз повторить эксперимент, по частоте выпадания исходов, входящих в это событие, можно судить о его вероятности.

2.2.2 Взаимоисключающие и независимые события

Для отображения пересечения между событиями, используется диаграмма Эйлера-Венна:

Вероятностное пространство

Рис. 2.1. Пересечение событий

События А и В пересекаются, значит существуют элементарные исходы, входящие и в А, и в В. **Взаимоисключающими** называют события, которые не могут произойти одновременно при проведении эксперимента — на диаграмме Эйлера-Венна они не пересекаются:

Рис. 2.2. Взаимоисключающие события

Вероятность взаимоисключающих событий равна нулю.

События называются **независимыми**, если наступление одного из них не влияет на вероятность другого. Если события независимы, то вероятность их пересечения равна произведению их вероятностей. Это правило работает и в обратную сторону.

Если взаимоисключающие события охватывают всё вероятностное пространство, сумма их вероятностей равна единице.

Взаимоисключаемость событий видна на диаграмме Эйлера-Венна. А вот независимость так просто не обнаружишь, нужно проверять условие равенства произведения вероятностей событий вероятности их пересечения.

2.2.3 Случайные величины, распределение вероятностей, интервалы значений

Случайная величина — это переменная, которая принимает случайные значения — те значения, которые нельзя предсказать до проведения эксперимента. У эксперимента есть исходы, которые могут описываться как количественно, так и качественно. Случайная же величина определяется на этих исходах численно. Это способ спроецировать исходы эксперимента, как бы они ни определялись, на числовую ось.

Как и все количественные переменные, случайная величина может быть дискретной или непрерывной.

Распределением вероятности случайной величины называется таблица, содержащая все возможные значения случайной величины и вероятности их появления.

Для хранения числовых таблиц, используется тип данных **array** из библиотеки Numpy:

```
table = np.array([[2, 3, 4, 5, 6, 7],
                  [3, 4, 5, 6, 7, 8], [7, 8, 9, 10, 11, 12]])
```

Для хранения значений и вероятностей их появления удобно использовать формат **словаря**. Если при работе со словарём необходимо получить список всех ключей словаря, то это можно сделать с помощью метода **keys()**. А список всех значений словаря получают с помощью метода **values()**:

```
dictionary = {1: 2, 2: 4, 3: 9}
print(dictionary.keys())
print(dictionary.values())
```

```
dict_keys([1, 2, 3])
dict_values([2, 4, 9])
```

2.2.4 Математическое ожидание

Для эксперимента можно задать случайную величину и найти численное значение, к которому она будет в среднем стремиться при многократном повторе эксперимента. Это значение называется **математическим ожиданием** случайной величины.

Если эксперимент состоит из равновероятных элементарных исходов, заданных численно, математическое ожидание будет равно среднему из возможных значений.

Математическое ожидание сл. величины — сумма всех значений случайной величины, помноженных на их вероятности: $E(X) = \sum p(x_i)x_i$

Математическое ожидание — аналог метрики локации, только не для датасета, а для случайной величины. Оно

показывает, вокруг какого значения распределена случайная величина, и — по закону больших чисел — к какому значению она будет в среднем стремиться при повторе эксперимента.

Поскольку случайная величина распределена вокруг этой «метрики локации», можно найти и меру её разброса. Для этого нужно найти математическое ожидание квадрата случайной величины — это несложно если учесть, что значения меняются, а их вероятности — нет.

Если известно математическое ожидание самой случайной величины и её квадрата, дисперсию находят по формуле: $Var(X) = E(X^2) - (E(X))^2$.

2.2.5 Распределения

Для некоторых явлений можно выяснить все возможные значения и описать частоту их появления — это называется **распределение**.

Биномиальное распределение

Эксперименты с двумя возможными исходами называются **биномиальными экспериментами**. Обычно один из результатов называют успехом, а второй, соответственно, неудачей. Если вероятность успеха равна p , то вероятность неудачи равна $(1 - p)$. Количество способов выбрать k успехов из n повторений эксперимента находят по формуле: $C_n^k = \frac{n!}{k!(n-k)!}$.

Для вычисления факториала используется библиотека `math` и её метод `factorial`:

```
from math import factorial

x = factorial(5)
print(x)
```

120

Рассмотрим задачу о биномиальном эксперименте (повторении эксперимента с двумя исходами n раз) в общем виде. Если вероятность успеха p и неуспеха $1 - p$, а эксперимент был повторен n раз, то вероятность любого количества успехов k из этих n экспериментов: $P(k \text{ успехов из } n \text{ попыток}) = C_n^k p^k (1-p)^{n-k}$.

Условия, при которых можно утверждать что случайная величина распределена биномиально:

- проводится конечное фиксированное число попыток n ;
- каждая попытка — простой биномиальный эксперимент ровно с двумя исходами;
- попытки независимы между собой;
- вероятность успеха p одинаковая для всех n попыток.

Нормальное распределение

Нормальные или тяготеющие к ним распределения распространены в жизни. Так распределяется рост людей, результаты измерения температуры. Понимание природы распределений необходимо для обнаружения важных аномалий. Так, при измерении длины изготовленных на заводе деталей ожидается распределение, тяготеющее к нормальному. Если же в результате расчётов получили распределение, скажем, с двумя

пиками, это должно насторожить аналитика. Такое может быть, если на заводе есть две производственные линии, и с одной из них приходят данные в сантиметрах, а с другой — в дециметрах.

Ключевая теорема в статистике — **центральная предельная теорема**. Если немного упростить, она гласит: «Много независимых случайных величин, сложенных вместе, дают нормальное распределение». Нормальное распределение описывает множество реальных непрерывных величин. Нормальное распределение определяют два параметра — среднее и дисперсия: $X \sim N(\mu, \sigma^2)$. Эта запись читается так: переменная X распределена нормально со средним μ и дисперсией σ^2 , то есть стандартным отклонением σ .

Для того, чтобы по известным параметрам распределения найти вероятность попадания в те или иные интервалы, вызовем два метода из пакета `scipy.stats`: **`norm.ppf`** и **`norm.cdf`**:

- `ppf` — функция процентных значений;
- `cdf` — кумулятивная функция распределения.

Обе работают с нормальным распределением, заданным средним и стандартным отклонением. Функция `norm.ppf` выдаёт значение переменной для известной вероятности интервала слева от этого значения. Функция `norm.cdf`, наоборот, выдаёт для известного значения вероятность интервала слева от этого значения.

Чтобы задать нормальное распределение, используется метод `norm()` из пакета `scipy.stats` с двумя аргументами: математическим ожиданием и стандартным отклонением. Найдём вероятность получить некоторое значение x :

```
from scipy import stats as st

# Задаем нормальное распределение
distr = st.norm(1000, 100)
x = 1000
```

```
result = distr.cdf(x)  # Считаем вероятность получить значение x
print(result)
```

0.5

С помощью функции `norm.cdf` можно посчитать вероятность получить значение в промежутке от x_1 до x_2 :

```
distr = st.norm(1000, 100)
x1 = 900
x2 = 1100
result = distr.cdf(x2) - distr.cdf(x1)  # Считаем вероятность
    ↪ получить значение между x1 и x2
print(result)
```

0.6826894921370859

Для того, чтобы по вероятности получить значение, воспользуемся методом `norm.ppf`:

```
from scipy import stats as st

# Задаем нормальное распределение
distr = st.norm(1000, 100)
p1 = 0.841344746
result = distr.ppf(p1)
print(result)
```

1099.999999971673

Примечание

Нормальная аппроксимация биномиального распределения

При большом количестве повторений биномиального эксперимента биномиальное распределение приближается к нормальному.

Для дискретного биномиального распределения, заданного числом повторов эксперимента n и вероятностью успеха p , математическое ожидание равно $n \cdot p$, а дисперсия: $n \cdot p \cdot (1 - p)$.

Если n больше 50, эти параметры биномиального распределения можно взять как среднее и дисперсию для нормального распределения, которое будет достаточно близко описывать биномиальное. Максимально близкое к биномиальному нормальное распределение задаётся его математическим ожиданием $n \cdot p$ в качестве среднего значения и дисперсией $n \cdot p \cdot (1 - p)$.

Распределение Пуассона

Ещё одно распределение, которое часто встречается — распределение Пуассона. Оно описывает число событий в единицу времени. Например, в среднем в кафе заходит 3 человека в минуту. Но это не значит, что каждую минуту заходит ровно 3 человека. Может, в эту минуту не зайдёт ни одного, а через минуту зайдёт 7. Если каждый посетитель заходит случайным образом, независимо от других — это и есть распределение Пуассона:

Рис. 2.3. Распределение Пуассона на примере числа посетителей кафе

Форма распределения Пуассона зависит от числа событий. При небольших значениях распределение смещено влево, а с увеличением числа событий становится симметричным и уходит вправо.

Невозможно получить отрицательное число заходов в кафе, но при небольшом потоке посетителей есть шанс, что в определённую минуту придут 5 посетителей или не придёт ни одного. В таком случае распределение Пуассона обрезано на нуле и имеет длинный «хвост» в положительную сторону. При увеличении числа посетителей шанс не получить ни одного захода в кафе уже почти не реализуется, поэтому распределение Пуассона становится симметричным и стремится к нормальному.

Рис. 2.4. Распределение становится симметричным

2.2.6 Случайная выборка и выборочное среднее

Логика проведения статистической проверки гипотез немного другая, по сравнению с механизмами в теории вероятностей. Прежде всего, мы будем судить о большом объёме данных, **генеральной совокупности**, по части — **выборке**.

Для анализа необязательно загружать все данные, достаточно взять небольшую, но **репрезентативную**, представляющую всю генеральную совокупность, часть данных. Самый простой способ добиться репрезентативности — взять **случайную выборку**. Из всего датасета генератором случайных чисел отбирают случайные элементы. По ним будут судить обо всей генеральной совокупности. Она может состоять из нескольких неравных по размеру частей, сильно отличающихся по исследуемому параметру. Тогда есть смысл взять пропорциональные случайные выборки из этих частей, и потом соединить между собой. Получается **стратифицированная выборка**, более репрезентативная, чем просто случайная. Она так называется, потому что генеральная совокупность была разбита на **страты** — группы, объединённые общим признаком. Случайные выборки получают уже из них.

По выборке судят о генеральной совокупности — точнее об её статистических параметрах. Обычно достаточно оценить среднее и дисперсию, чтобы сделать выводы о **равенстве или неравенстве средних значений** исследуемых совокупностей. Нас будет интересовать именно такая постановка задачи.

Среднее и дисперсию генеральной совокупности считают по ее выборке, и называют **выборочное среднее** и **выборочная дисперсия**. При условии, что выборка достаточно велика, эти данные позволяют сделать выводы о совокупностях в целом.

Одна из формулировок центральной предельной теоремы звучит так: если в выборке достаточно наблюдений, **выборочное распределение** выборочного среднего из любой генеральной совокупности распределено нормально вокруг среднего этой генеральной совокупности. «Любая генеральная совокупность» означает, что сама генеральная совокупность может быть распределена как угодно. Датасет из средних значений выборок всё равно будет нормально распределён вокруг среднего всей

генеральной совокупности. Стандартное отклонение выборочного среднего от настоящего среднего генеральной совокупности называется **стандартной ошибкой** и находится по формуле:

$$SE = \frac{S}{\sqrt{n}},$$

где S — оценка стандартного отклонения генеральной совокупности.

n — размер выборки. Так как корень из n стоит в знаменателе, стандартная ошибка уменьшается с увеличением размера выборки.

2.2.7 Диаграмма размаха

Описывая распределение, аналитики рассчитывают среднее арифметическое или медиану. Однако помимо медианы и среднего, важно знать характерный разброс — то, какие значения оказались вдали от среднего, и насколько их много.

Проще всего для оценки разброса подсчитать минимальное и максимальное значения. Такое описание не всегда точно, подвержено влиянию выбросов. Гораздо более устойчивой оценкой считается межквартильный размах.

Квартили (от латинского *quartus* — «четвёртый») разбивают упорядоченный набор данных на четыре части. Первый квартиль $Q1$ — число, отделяющее первую четверть выборки: 25% элементов меньше, а 75% — больше него. Медиана — второй квартиль $Q2$, половина элементов больше и половина меньше неё. Третий квартиль $Q3$ — это отсечка трёх четвертей: 75% элементов меньше и 25% элементов больше него. Межквартильный размах — это расстояние между $Q1$ и $Q3$.

Рис. 2.5. Разбиение набора данных на квартили

Увидеть медиану и квартили в Python можно на специальном графике. Его называют диаграмма размаха или «ящик с усами».

«Ящик» ограничен первым и третьим квартилями. Внутри ящика обозначают медиану. «Усы» простираются влево и вправо от границ ящика на расстояние, равное 1,5 межквартильным размахам (сокращённо IQR, от англ. interquartile range, «межквартильный размах»). В размах «усов» попадают нормальные значения, а за пределами находятся выбросы, изображённые точками. Если правый «ус» длиннее максимума, то он заканчивается максимумом. То же — для минимума и левого уса.

Рис. 2.6. "Ящик с усами" и выбросы на числовой прямой

Коротко перечислим основные компоненты «ящика с усами»:

- **Отрезок** (Range), отражающий диапазон значений набора данных.
- **Прямоугольник** (Box), который отображает границы основного объема данных.левой границей считается значение 25% квартиля или Q1. Правой границей является 75% квартиль или Q3.

- **Усы** (Whiskers) — линии, исходящие из прямоугольника, используются для представления разброса данных до крайних значений на отрезке: минимального слева и максимального справа.
- **Выбросы** (Outliers) — значения, находящиеся за левым и правым «крайними» значениями на оси.
- Линия внутри прямоугольника — это **медиана**. Медиана всегда равна середине упорядоченного ряда значений.

В Python диаграмму размаха строят методом **boxplot()** (от англ. box and whisker plot — «ящик с усами»).

```
sns.set_theme(style="whitegrid")
tips = sns.load_dataset("tips")
ax = sns.boxplot(x=tips["total_bill"])
```

Отообразим вертикальную прямоугольную диаграмму, сгруппированную по категориальной переменной `day` (день недели):

```
ax = sns.boxplot(x="day", y="total_bill", data=tips)
```

Ещё один способ увеличения информативности графика `boxplot` — использование параметра **hue**. Он дополнительно разделяет данные в зависимости от значения дополнительного категориального признака. Посмотрите, как меняется отображение `boxplot` при добавлении параметра `hue`:

```
ax = sns.boxplot(x="day", y="total_bill", hue="smoker",  
↪ data=tips)
```

Такой способ представления информации позволяет нам увидеть, как менялось распределение общей суммы счета в разные дни недели отдельно для курящих и для клиентов, которые не курят.

Таким образом, boxplot используется для составления гипотез и наглядной визуализации большого числа статистических данных. Внутри боксплота можно спрятать тысячи значений, но они все визуально помещаются на небольшом графике. С его помощью можно разом показать основные статистические характеристики данных.

Boxplot, или как его еще называют, «ящик с усами», по сути является «сплюсненной» гистограммой. По этому графику можно легко понять состав наших данных и основную статистическую информацию. На нем показывается медиана, а также он более компактно отражает характер распределения значений переменной.

Примечание

Данные, которые распределены согласно закону нормального распределения, располагаются **симметрично относительно своего среднего значения**, а их графическое представление имеет колоколообразную форму. Большинство значений лежат между первым квартилем $Q1$ и третьим $Q3$ — внутри колокола. Медиана и среднее арифметическое при нормальном распределении всегда совпадают и имеют одинаковое значение.

Рис. 2.7. Нормальное распределение

2.3 Выводы

В этой главе были рассмотрены основные понятия и определения математической статистики и теории вероятности, которые применяются при анализе данных. Приведены примеры на языке Python, наглядно демонстрирующие применение изученного математического аппарата на практике при анализе данных и последующей визуализации данных.

Изучив данный раздел, вы можете:

- выбирать оптимальные метрики для описания данных;
- оценивать дискретные и непрерывные величины при помощи гистограмм разных типов;
- считать статистические характеристики в Python и визуализировать их;
- делать выводы о данных по статистическим показателям;
- определять тип распределения, рассчитать нормальное, биномиальное и распределение Пуассона;
- строить диаграмму «ящик с усами» (диаграмму размаха) и использовать ее в анализе.

Глава 3 Сбор и предобработка данных

3.1 Получение данных

Данные хранят в файлах разных форматов. Из них самый распространённый и простой – **CSV** (от англ. Comma-Separated Values, «значения, разделённые запятой»). Каждая строка такого файла представляет собой одну строку таблицы, где данные разделены запятыми. В первой строке собраны заголовки столбцов (если они есть).

Файлы CSV удобнее всего открывать вызовом метода `read_csv()` из библиотеки Pandas.

```
import pandas as pd

df = pd.read_csv('https://github.com/datasciencefefe/course/raw/
↪main/data/iris.csv')
```

Теперь все данные из файла можно напечатать на экране командой `print(df)`, но это не всегда нужно делать — не исключено, что таблица огромна и неудобна при изучении. Для знакомства с данными запрашивают несколько строк из начала или конца таблицы, вызывая специальные методы **head()** и **tail()**. По умолчанию `head()` возвращает первые 5 строк набора данных, а метод `tail()` – последние 5 строк. Когда нужно не 5, количество строк передаётся этим методам как аргумент. Например, `head(8)` вернёт первые 8 строк.

```
print(df.head(8))
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
5	5.4	3.9	1.7	0.4	setosa
6	4.6	3.4	1.4	0.3	setosa
7	5.0	3.4	1.5	0.2	setosa

Полученная таблица хранится в структуре данных `DataFrame`. Давайте подробно разберём, из чего состоит этот объект и какие операции с ним можно выполнять.

3.1.1 DataFrame

DataFrame — это двумерная структура данных `Pandas`, где у каждого элемента есть две координаты: строка и столбец. Каждая строка — это одно наблюдение, запись об объекте исследования. А столбцы — признаки объектов.

Для лучшего понимания данных полезно получить доступ к их описанию. Часто описание выполняют в виде документации со сведениями о содержании каждого столбца.

Для датасета `iris.csv` есть следующее описание. Имеется набор из данных о 150 экземплярах ирисов, по 50 экземпляров трех следующих видов:

- ирис щетинистый (`iris setosa`);
- ирис версиколор (`iris versicolor`);
- ирис виргинский (`iris virginica`).

Для каждого экземпляра приведены 4 следующие характеристики:

- длина чашелистика (sepal length);
- ширина чашелистика (sepal width);
- длина лепестка (petal length);
- ширина лепестка (petal width).

У DataFrame есть неотъемлемые свойства, значения которых можно запросить. Они называются **атрибуты**. Например, атрибут `columns` содержит информацию о названиях столбцов в наборе данных.

```
print(df.columns)
```

```
Index(['sepal_length', 'sepal_width', 'petal_length',  
      ↪ 'petal_width',  
        'species'],  
      dtype='object')
```

В данном случае атрибут `columns` вернул список названий столбцов и сообщил, что большинство из них имеет тип данных `float64`. Вообще типы данных могут быть разные. Для просмотра типа данных каждого столбца лучше всего использовать атрибут **`dtypes`**.

```
print(df.dtypes)
```

```
sepal_length    float64  
sepal_width     float64  
petal_length    float64  
petal_width     float64  
species         object  
dtype: object
```

Типы данных, о которых сообщают атрибуты — это типы данных библиотеки Pandalas. Каждому из них соответствует определённый тип данных языка Python. Так, типу `int` в

Pandas будет соответствовать тип `int64`. Тип данных `object` используется, когда данные не подходят ни под одну категорию или соответствуют в Python типу «строка». Приведем таблицу соответствия типов данных Pandas и Python (см. таблицу 1):

Таблица 1

Соответствие типов данных Pandas и Python

Pandas dtype	Python type	Описание
object	str	Строка
int64	int	Целые числа
float64	float	Вещественные числа
bool	bool	Логический тип данных

О размерах таблицы с данными сообщает её атрибут **shape**. В результате получается **кортеж** (неизменяемый список) из двух чисел: первое – количество строк, второе – количество столбцов.

```
print(df.shape)
```

```
(150, 5)
```

Всю информацию, которую предоставляют разные атрибуты `DataFrame`, можно получить вызовом одного-единственного метода **info()**. Изучив результаты, которые этот метод возвращает, аналитик выбирает тактику дальнейшей работы с таблицей.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 150 entries, 0 to 149
```

```
Data columns (total 5 columns):
```

#	Column	Non-Null Count	Dtype
---	-----	-----	-----
0	sepal_length	150 non-null	float64
1	sepal_width	150 non-null	float64

```

2   petal_length  150 non-null    float64
3   petal_width   150 non-null    float64
4   species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB

```

Индексация в DataFrame

К каждой ячейке с данными в DataFrame можно обратиться по её индексу и названию столбца. В зависимости от запроса к DataFrame можно получить различные срезы данных. Этот процесс называется **индексация**. Для DataFrame она проводится разными способами.

Атрибут `.loc[строка, столбец]` даёт доступ к элементу по строке и столбцу (см. таблицу 2).

Таблица 2

Индексация DataFrame

Вид	Реализация
Одна ячейка	<code>.loc[7, 'species']</code>
Один столбец	<code>.loc[:, 'species']</code>
Несколько столбцов	<code>.loc[:, ['species', 'sepal_length']]</code>
Несколько столбцов подряд (срез)	<code>.loc[:, 'species': 'sepal_length']</code>
Одна строка	<code>.loc[1]</code>
Все строки, начиная с заданной	<code>.loc[1:]</code>
Все строки до заданной	<code>.loc[:3]</code>
Несколько строк подряд (срез)	<code>.loc[2:5]</code>

Запрос к атрибуту `.loc[ ]` использует квадратные скобки, это напоминает списки в Python. Индексация здесь очень похожа на индексацию списков.

Важное замечание: атрибут `.loc[ ]` включает и начало, и конец среза.

В то же время нужно уметь подсчитать количество определённых значений в столбце или во всем датасете. В Pandas для этого есть метод `count()`. Его вызывают для подсчёта количества ячеек, соответствующих определённому условию. Например, необходимо подсчитать количество ячеек столбца `species`, где значение равно `versicolor`. Для столбца `species` таблицы `df` такие ячейки отвечают логическому условию `df.loc[:, 'species'] == 'versicolor'`. Поскольку в указании, какие именно значения считать, нужен логический оператор, такой доступ к значению ячейки называют **логическая индексация**.

```
print(df.loc[df.loc[:, 'species'] == 'versicolor']['  
        'species'].count()) # используем метод .count() для  
↪ подсчёта записей, удовлетворяющих условию в столбце species
```

3.1.2 Series

Сама таблица представляет собой `DataFrame`, но каждый столбец сам по себе — вовсе не структура данных `DataFrame`. Если взять отдельный столбец таблицы, то он представляет собой совсем иную структуру данных — `Series` — одномерный объект, содержащий набор данных (любого типа, поддерживаемого NumPy) и ассоциированный с ним массив меток, который называется индексом.


```
type(df['species'])
```

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

Series — одномерная таблица, и обратиться к её элементам можно по индексу. Каждый индекс — это номер отдельного наблюдения, и поэтому несколько различных Series вместе составляют DataFrame. В Series хранятся данные одного типа.

У каждой Series есть имя (Name), информация о количестве данных в столбце (Length) и тип данных, которые хранятся в ней (dtype).

```
print(df['species'])
```

```
0      setosa
1      setosa
2      setosa
3      setosa
4      setosa
...
148    virginica
149    virginica
Name: species, Length: 150, dtype: object
```

Индексация в Series (см. таблицу 3) аналогична индексации элементов столбца в DataFrame.

Таблица 3

Индексация Series

Вид	Реализация	Сокращённая запись
Один элемент	<code>a.loc[7]</code>	<code>a[7]</code>
Несколько элементов	<code>a.loc[[5, 7, 10]]</code>	<code>a[[5, 7, 10]]</code>

Индексация Series

Вид	Реализация	Сокращённая запись
Несколько элементов подряд (срез)	<code>a.loc[5:10]</code> (включая 10)	<code>a[5:10]</code> (не включая 10)
Все элементы, начиная с заданного	<code>a.loc[1:]</code>	<code>a[1:]</code>
Все элементы до заданного	<code>a.loc[:3]</code> (включая 3)	<code>a[:3]</code> (не включая 3)

Для Series также как и для DataFrame возможна и логическая индексация.

3.2 Регулярные выражения

Из датасетов аналитик извлекает информацию, фильтрует неправдоподобные данные и ищет общие элементы. Здесь в ход идёт мощный инструмент поиска строк — регулярные выражения.

Регулярное выражение — правило для поиска подстрок (фрагментов текста внутри строк).

Например, если необходимо найти сочетание букв «со» в предложении «Считается, что кроме трёх всем известных – мясо, просо, колесо – четвёртого такого слова в русском языке не существует», для Python это поиск подстроки “со” в строке ‘Считается, что кроме трёх всем известных – мясо, просо, колесо – четвёртого такого слова в русском языке не существует’.

Регулярные выражения позволяют создавать сложные правила, так что одно выражение вернёт несколько подстрок.

Для работы с регулярными выражениями в Python импортируют библиотеку **re** (от англ. regular expressions — «регулярные выражения»). Дальше поиск ведётся в два этапа.

Сначала создают шаблон регулярного выражения. Это алгоритм, по которому нужно искать строку в тексте. Затем готовый шаблон передают специальным методам библиотеки **re**, которые ищут, заменяют и удаляют нужные символы. Таким образом, шаблон определяет, что и как искать, а метод — что с этим потом делать.

В таблице 4 приведены простейшие шаблоны регулярных выражений. Сложные регулярные выражения состоят из их комбинаций.

Таблица 4

Простейшие шаблоны регулярных выражений

Регулярное выражение	Описание
[]	Один из символов в скобках
[^...]	Отрицание
—	Интервал
.	Один любой символ, кроме символа перевода строки
\w	Любая буква, цифра или _
[A-z]	Любая латинская буква
[А-я]	Любая буква кириллицы

Простейшие шаблоны регулярных выражений

Регулярное выражение	Описание
?	Ноль или одно вхождение
\d	Любая цифра
+	Одно и более вхождений
*	Ноль и более вхождений
^	Начало строки
\$	Конец строки

Самые распространённые задачи с использованием регулярных выражений:

- найти подстроку в строке;
- разбить строки на подстроки на основании шаблона;
- заменить части строки на другую строку.

Рассмотрим подробнее методы библиотеки `re`, которые понадобятся для решения этих задач.

`search(pattern, string)` ищет шаблон `pattern` в строке `string`. Хотя `search()` ищет шаблон во всей строке, возвращает он только первую найденную подстроку:

```
import re

string = "«Taxi Driver» 8 февраля 1976 года Нью-Йорк Роберт_↪ДеНиро"
print(re.search('\w+', string))
```

```
<re.Match object; span=(1, 5), match='Taxi'>
```

Метод `search()` возвращает объект типа **`match`** (от англ. «соответствовать»). Параметр `span` (от англ. «диапазон») указывает диапазон индексов, подходящих под шаблон. В нашем

случае открывающая кавычка “«” не отвечает правилу, которое игнорирует знаки препинания. Вот потому индексы идут с 1 по 8: от буквы «Т» до буквы «i». В параметре `match` указано само значение подстроки.

Шаблону `'\w+'` соответствует любая подстрока, содержащая одну и более букв, цифр или символ нижнего подчёркивания “_”. Метод `search()` нашёл, что этому шаблону соответствует первое слово в строке. Так как под правило `'\w+'` не подходит пробел, метод вернул всё, что идёт до первого пробела. Если нам не нужны дополнительные сведения о диапазоне, выведем только найденную подстроку методом `group()`:

```
string = "«Taxi Driver» 8 февраля 1976 года Нью-Йорк Роберт_
↪ДеНиро"
print(re.search('\w+', string).group())
```

Taxi

Рассмотрим способ добыть информацию между определенными словами. Извлечем данные, содержащие буквы или пробел между символами “«” и “»”:

```
string = "«Taxi Driver» 8 февраля 1976 года Нью-Йорк Роберт_
↪ДеНиро"
print(re.search('<[A-zA-я ]+>', string).group())
```

«Taxi Driver»

Обратите внимание, что учтены все символы, содержащиеся в нужной подстроке, в том числе — пробел.

`split(pattern, string)` (от англ. «расщеплять, разбивать») разделяет строку `string` по границе шаблона `pattern`.

```
string = "«Taxi Driver» 8 февраля 1976 года Нью-Йорк Роберт_
↪ДеНиро"
print(re.split('\d+', string))
```

```
['«Taxi Driver» ', ' февраля ', ' года Нью-Йорк Роберт ДеНиро']
```

Строка разделена на три части. Границы деления строки проходят там, где метод встретил указанный в аргументе шаблон. В нашем случае шаблону регулярного выражения `'\d+'` соответствует одна и более цифр. Поэтому строка поделилась на три части: «Taxi Driver», 8 февраля 1976 года Нью-Йорк Роберт ДеНиро. Количество делений строки можно управлять. За это отвечает параметр **maxsplit** метода `split()` (по умолчанию равен 0).

```
string = "«Taxi Driver» 8 февраля 1976 года Нью-Йорк Роберт ДеНиро"
print(re.split('\d+', string, maxsplit = 1))
```

```
['«Taxi Driver» ', ' февраля 1976 года Нью-Йорк Роберт ДеНиро']
```

Строка разделилась один раз по первому найденному шаблону.

Метод **sub(pattern, repl, string)** (от англ. substring, «подстрока») ищет подстроку `pattern` в строке `string` и заменяет его на подстроку `repl` (от англ. replace — «заменить»).

```
string = "«Taxi Driver» 8 февраля 1976 года Нью-Йорк Роберт ДеНиро"
print(re.sub('\d+', '', string)) # Ищем числа и заменяем на пустоту
```

```
«Taxi Driver»  февраля  года Нью-Йорк Роберт ДеНиро
```

Все подстроки с числами заменены на пустоту.

Метод **findall(pattern, string)** возвращает список всех подстрок в `string`, удовлетворяющих шаблону `pattern`. А не только первую подходящую подстроку, как `search()`. Найдём все слова, оканчивающиеся на “co”:

```
so = "Считается, что кроме трёх всем известных - мясо, просо,
↳ колесо - четвёртого такого слова в русском языке не
↳ существует"
print(re.findall('[А-я]+co', so))
```

```
['мясо', 'просо', 'колесо']
```

Метод **findall()** удобен тем, что можно сразу посчитать количество повторяющихся подстрок в строке функцией **len()**:

```
string = "«Taxi Driver» 8 февраля 1976 года Нью-Йорк Роберт
↳ ДеНиро"
print(len(re.findall('\w+', string)))
```

10

Функция **len()** вывела результат на единицу больше, чем ожидалось. Так произошло, потому что слово «Нью-Йорк» разбилось на два слова, ведь «-» не подходит под шаблон `\w+`. Добавим `'-'` в шаблон регулярного выражения и учтем Нью-Йорк как одно слово.

```
string = "«Taxi Driver» 8 февраля 1976 года Нью-Йорк Роберт
↳ ДеНиро"
print(len(re.findall('[\w-]+', string)))
```

9

3.3 Web Mining

Если данных мало, то может понадобиться обогащение данных. Это процесс добавления данных в уже готовую выборку. Сперва находят ценные для исследования веб-ресурсы, а затем

извлекают из них нужные данные. Этот процесс называют **Web Mining** (от англ. web — «сеть»; mining — «добыча, разработка полезных ископаемых»). В результате удастся не только учесть больше факторов, но и выявить новые закономерности, прийти к неожиданным выводам.

Что добывают в процессе Web Mining:

- Контент сайтов. Текст, картинки, файлы, таблицы.
- Структуры сайтов. Ссылки и взаимосвязи веб-страниц — всё, что даёт представление об устройстве сайта.
- Сведения о пользователях. Характеристики взаимодействия пользователя с сайтом.

По-другому Web Mining называют парсинг (от англ. parse — «проводить разбор, анализ»). Аналитики на профессиональном жаргоне говорят, что «парсят» сайты.

3.3.1 Get-запрос

Для получения данных с сервера понадобится метод **get()**. Для отправления HTTP-запросов подключают библиотеку **requests** (от англ. «запросы»). Импортируем библиотеку:

```
import requests
```

Прежде всего, нужна ссылка на сайт. Сохраним ссылку в переменной **URL** (от англ. Uniform Resource Locator — «единый указатель ресурса»).

```
URL='https://datasciencefufu.github.io/course/scorsese.html'
```

Метод **get()** библиотеки **Requests** выступает в роли браузера. Передадим ему ссылку на сайт как аргумент. Метод отправит get-запрос на сервер, обработает полученный оттуда результат и

вернёт объект **response** (от англ. «ответ») — специальный объект, содержащий ответ сервера на HTTP-запрос:

```
req = requests.get(URL)
```

Объект `response` содержит ответ сервера: код состояния, содержание запроса и код самой HTML-страницы. Атрибуты объекта `Response` позволяют возвращать не все данные с сервера, а только нужные для анализа. Например, объект `Response` с атрибутом `text` вернет лишь текстовое содержание запроса:

```
print(req.text)
```

```
<!DOCTYPE HTML>
```

```
<html>
```

```
    <head>
```

```
    <title>Содержимое курса</title>
```

```
        <meta charset="utf-8" />
```

```
        <meta name="viewport" ␣
```

```
↪content="width=device-width, initial-  
scale=1, user-scalable=no" />
```

```
        <link rel="stylesheet" href="assets/css/main.css"  
        /
```

```
        .....
```

```
        .....
```

```
        .....>
```

Подробнее текстовое содержание запроса рассмотрено в соответствующей лабораторной работе.

Атрибут `status_code` (от англ. «код состояния») позволяет понять отправил ли сервер ответ или возникла какая-то ошибка.

```
print(req.status_code)
```

Не все запросы возвращаются с данными. Иногда результатом запроса бывает ошибка: в зависимости от типа её обозначают специальным кодом (см. таблицу 5).

Таблица 5

Коды ошибок их значения	
Код ошибки	Значение
200	OK
302	Found
400	Bad Request
404	Not found
500	Internal Server Error
502	Bad Gateway
503	Server Unavailable

3.3.2 Парсинг HTML

После того как мы отправили get-запрос и добыли код страницы, необходимо добыть чистые данные. Вручную это делать достаточно долго и сложно, поэтому следует обратиться к возможностям библиотеки **BeautifulSoup** (от англ. beautiful soup, «красивый суп»). Имя библиотеки выбрано в противовес tag soup (от англ. «суп из тегов»), как уничижительно называют неструктурированный, небрежно написанный код веб-страницы.

Методы библиотеки BeautifulSoup превращают HTML-файл в древовидную структуру. После этого нужный контент легко отыскать по тегам и атрибутам.

Импортируем библиотеку и создадим объект BeautifulSoup:

```
from bs4 import BeautifulSoup

soup = BeautifulSoup(req.text, 'lxml')
```

Первый аргумент — это данные, из которых будет собираться древовидная структура. Второй аргумент — синтаксический анализатор, или парсер. Он отвечает за то, как именно из кода веб-страницы получается «дерево». Парсеров много, они создают разные структуры из одного и того же HTML-документа. За высокую скорость работы часто выбирают анализатор lxml. Есть и другие, например, html.parser или xml.

Первый метод поиска называется **find()** (от англ. «найти»). В HTML-документе он находит первый элемент, имя которого ему передали в качестве аргумента, и возвращает его весь, с тегами и контентом. Найдём, к примеру, первый заголовок третьего уровня:

```
heading_3=soup.find('h3')
print(heading_3)
```

```
<h3 class="major">Награды Каннского и Берлинского кинофестивалей</h3>
```

Чтобы посмотреть контент без тега, вызывают метод **text**. Результат возвращается в виде строки:

```
print(heading_3.text)
```

Награды Каннского и Берлинского кинофестивалей

Существует и другой метод поиска — **find_all()** (от англ. «найти всё»). В отличие от предыдущего метода, find_all() находит все вхождения определённого элемента в HTML-документе и возвращает список:

```
paragraph=soup.find_all('p') # напомним, p - это параграф, текст
    ↳ между тегами <p> и </p>
print(paragraph)
```

```
[<p>Мартин Скорсезе - американский кинорежиссёр, продюсер,
    ↳ сценарист и актёр.
```

Фильмам Скорсезе присущи выразительная жестокость и насилие, в

кинематографических кругах он известен как мастер гангстерских
 ↳лент. Скорсезе
 неоднократно признавался одним из величайших и наиболее
 ↳влиятельных
 кинорежиссёров современности. Посмотрим на некоторые награды
 ↳знаменитого
 режиссёра.</p>]

Методом `text` вычленим только контент из параграфов:

```
for paragraph in soup.find_all('p'):
    print(paragraph.text)
```

Мартин Скорсезе - американский кинорежиссёр, продюсер, сценарист
 ↳и актёр.
 Фильмам Скорсезе присущи выразительная жестокость и насилие, в
 кинематографических кругах он известен как мастер гангстерских
 ↳лент. Скорсезе
 неоднократно признавался одним из величайших и наиболее
 ↳влиятельных
 кинорежиссёров современности. Посмотрим на некоторые награды
 ↳знаменитого
 режиссёра.

У методов `find()` и `find_all()` есть дополнительный фильтр поиска элементов страницы — параметр **attrs** (от англ. attributes, «атрибуты»). Он находит идентификаторы и их классы. Параметру `attrs` передают словарь с именами и значениями атрибутов. Например, найдем элемент с идентификатором 'oscar':

```
print(soup.find('table', attrs={'id': 'oscar'}))
```

```
<table id="oscar">
<thead>
<tr>
<th>Название фильма</th>
<th>Дата американской премьеры</th>
<th>Основное место съёмок</th>
<th>Главный актер</th>
```

```

</tr>
</thead>
<tbody>
<tr>
<td>«Raging Bull»</td>
<td>14 ноября 1980 года</td>
<td>Нью-Йорк</td>
<td>Роберт ДеНиро</td>
</tr>
<tr>
<td>«The Last Temptation of Christ»</td>
<td>12 августа 1988 года</td>
<td>Уарзазат</td>
<td>Уиллем Дефо</td>
</tr>
<tr>
<td>«Gangs of New York»</td>
<td>27 марта 2003 года</td>
<td>Студия Чинечитта</td>
<td>Леонардо ДиКаприо</td>
</tr>
<tr>
<td>«The Aviator»</td>
<td>25 декабря 2004 года</td>
<td>Монреаль</td>
<td>Леонардо ДиКаприо</td>
</tr>
<tr>
<td>«Hugo»</td>
<td>5 января 2012 года</td>
<td>Париж</td>
<td>Эйса Баттерфилд</td>
</tr>
<tr>
<td>«The Wolf of Wall Street»</td>
<td>6 февраля 2014 года</td>
<td>Нью-Йорк</td>
<td>Леонардо ДиКаприо</td>
</tr>
<tr>
<td>«The Irishman»</td>

```

```

<td>1 ноября 2019 года</td>
<td>Нью-Йорк</td>
<td>Роберт ДеНиро</td>
</tr>
</tbody>
</table>

```

Усложним задачу. Достанем таблицу с номинациями Мартина Скорсезе на Оскар и превратим её в датафрейм:

```

table = soup.find('table', attrs={'id': 'oscar'})
# Применим метод find к тегу table
# Укажем атрибут первой таблицы: oscar
print(table)

```

```

<table id="oscar">
<thead>
<tr>
<th>Название фильма</th>
<th>Дата американской премьеры</th>
<th>Основное место съёмок</th>
<th>Главный актер</th>
</tr>
</thead>
<tbody>
<tr>
<td>«Raging Bull»</td>
<td>14 ноября 1980 года</td>
<td>Нью-Йорк</td>
<td>Роберт ДеНиро</td>
</tr>
<tr>
<td>«The Last Temptation of Christ»</td>
<td>12 августа 1988 года</td>
<td>Уарзазат</td>
<td>Уиллем Дефо</td>
</tr>
<tr>
<td>«Gangs of New York»</td>
<td>27 марта 2003 года</td>
<td>Студия Чинечитта</td>

```

```

<td>Леонардо ДиКаприо</td>
</tr>
<tr>
<td>«The Aviator»</td>
<td>25 декабря 2004 года</td>
<td>Монреаль</td>
<td>Леонардо ДиКаприо</td>
</tr>
<tr>
<td>«Hugo»</td>
<td>5 января 2012 года</td>
<td>Париж</td>
<td>Эйса Баттерфилд</td>
</tr>
<tr>
<td>«The Wolf of Wall Street»</td>
<td>6 февраля 2014 года</td>
<td>Нью-Йорк</td>
<td>Леонардо ДиКаприо</td>
</tr>
<tr>
<td>«The Irishman»</td>
<td>1 ноября 2019 года</td>
<td>Нью-Йорк</td>
<td>Роберт ДеНиро</td>
</tr>
</tbody>
</table>

```

Напомним, что открывающий тег `<table>` указывает начало таблицы, а закрывающий `</table>` — её конец. Внутри теги строк `<tr>`; ячеек `<td>` и заголовков столбцов `<th>`.

Создадим пустой список `heading_table`, где сохраним названия столбцов. В цикле методом `find_all()` найдём все элементы `th`. Методом `text` добудем их контент и добавим его в список `heading_table`:

```

heading_table = [] # Список, в котором будут храниться названия
    ↳ столбцов
for row in table.find_all('th'): # Названия столбцов прячутся в
    ↳ элементах th,
    # поэтому будем искать все элементы th внутри table и
    ↳ пробежать по ним в цикле
    heading_table.append(row.text) # Добавляем контент из тега
    ↳ th в список heading_table
print(heading_table)

```

```

['Название фильма', 'Дата американской премьеры', 'Основное место
    ↳ съёмки',
'Главный актер']

```

Создадим пустой список `content`, сохраним там данные таблицы. В цикле обратимся к каждой строке по имени элемента `tr`.

Отдельно отметим, что самая первая строка таблицы с заголовками в тегах `<th>` `</th>`, нас не интересует. Потому перед тем, как в цикле добавлять значения в пустой список, укажем, что это не касается строки с заголовками: `if not row.find_all('th')`.

Применим метод `find_all()` к элементам `<td>`. Методом `text` очистим полученные ячейки от тегов и сложим в список `content`.

```

content = [] # Список, в котором будут храниться данные из
    ↳ таблицы
for row in table.find_all('tr'):
    # Каждая строка обрамляется тегом tr, необходимо пробежаться
    ↳ в цикле по всем строкам
    if not row.find_all('th'):
        # Эта проверка необходима, чтобы пропустить первую
        ↳ строку таблицы с заголовками
        content.append([element.text.replace('\xa0', ' ') for
            ↳ element in row.find_all('td')])
        # В каждой строке контент ячейки обрамляется тегами <td>
        ↳ </td>

```



```

# Необходимо пройти в цикле по всем элементам td,
→вычленим контент из ячеек
# и добавим его в список
# Затем добавим каждый из списков в список content
print(content)

```

```

[['«Raging Bull»', '14 ноября 1980 года', 'Нью-Йорк', 'Роберт ДеНиро'],
 ['«The Last Temptation of Christ»', '12 августа 1988 года', 'Уарзазат',
  'Уиллем Дефо'],
 ['«Gangs of New York»', '27 марта 2003 года', 'Студия Чинечитта',
  'Леонардо ДиКаприо'],
 ['«The Aviator»', '25 декабря 2004 года', 'Монреаль',
  'Леонардо ДиКаприо'],
 ['«Hugo»', '5 января 2012 года', 'Париж', 'Эйса Баттерфилд'],
 ['«The Wolf of Wall Street»', '6 февраля 2014 года', 'Нью-Йорк',
  'Леонардо ДиКаприо'],
 ['«The Irishman»', '1 ноября 2019 года', 'Нью-Йорк', 'Роберт ДеНиро']]

```

В результате получены 2 списка. В списке `heading_table` сохранили названия столбцов. В `content` — наполнение таблицы в виде двумерного массива.

```

oscar_nomination = pd.DataFrame(content, columns=heading_table)
# В качестве данных передаем двумерный список content, а в
→качестве заголовков - heading_table
print(oscar_nomination.head())

```

	Название фильма	...	Главный актер
0	«Raging Bull»	...	Роберт ДеНиро
1	«The Last Temptation of Christ»	...	Уиллем Дефо
2	«Gangs of New York»	...	Леонардо ДиКаприо
3	«The Aviator»	...	Леонардо ДиКаприо
4	«Hugo»	...	Эйса Баттерфилд

```
[5 rows x 4 columns]
```

Теперь таблица с сайта преобразована в удобный для анализа датафрейм.

3.3.3 JSON

Отвечая на запрос, сервер возвращает структурированные данные в одном из специальных форматов. Самый распространённый из них — **JSON**.

JSON расшифровывается как JavaScript Object Notation (от англ. «объектная запись JavaScript»). Вот так выглядят данные в этом формате:

```
[
  {
    "Название фильма": "«Taxi Driver»",
    "Дата американской премьеры": "8 февраля 1976 года",
    "Основное место съёмок": "Нью-Йорк",
    "Главный актер": "Роберт ДеНиро"
  },
  {
    "Название фильма": "«After Hours»",
    "Дата американской премьеры": "11 сентября 1985 года",
    "Основное место съёмок": "Нью-Йорк",
    "Главный актер": "Гриффин Данн"
  },
  {
    "Название фильма": "«Goodfellas»",
    "Дата американской премьеры": "9 сентября 1990 года",
    "Основное место съёмок": "Нью-Йорк",
    "Главный актер": "Роберт ДеНиро"
  }
]
```

```
[{'Главный актер': 'Роберт ДеНиро',
  'Дата американской премьеры': '8 февраля 1976 года',
  'Название фильма': '«Taxi Driver»',
  'Основное место съёмок': 'Нью-Йорк'},
{'Главный актер': 'Гриффин Данн',
  'Дата американской премьеры': '11 сентября 1985 года',
  'Название фильма': '«After Hours»',
  'Основное место съёмок': 'Нью-Йорк'},
```

```
{'Главный актер': 'Роберт ДеНиро',
  'Дата американской премьеры': '9 сентября 1990 года',
  'Название фильма': '«Goodfellas»',
  'Основное место съёмок': 'Нью-Йорк'}]
```

Если JSON содержит несколько элементов, то их перечисляют внутри квадратных скобок: [...] — как в списках. Каждый элемент JSON похож на словарь — он записан в фигурных скобках, внутри которых пары **ключ : значение**.

Поскольку данные записаны в строго определённой форме, их можно превратить в строку и передать в HTTP-запросе. В этом и заключается смысл JSON: он позволяет собирать данные в объект (список пар **ключ : значение**), затем преобразовывать этот объект в строку для передачи в запросе. Получатель превращает эту строку обратно в объект.

Ключи обрاملены двойными кавычками. Значениями ключей могут быть строки, числа, булевы значения, null, массивы, объекты.

JSON начинается либо с фигурных скобок, если содержит один объект, либо с квадратных — если список объектов. JSON не может содержать функций, переменных или комментариев.

В Python есть встроенный модуль для работы с данными в формате JSON:

```
import json
```

Метод **json.loads()** (от англ. loads — «загрузки») конвертирует строки в формате JSON:

```
x = '{"Название фильма": "«Taxi Driver»", "Дата американской_
  ↳премьеры": "8 февраля 1976 года"}'
y = json.loads(x)

print('Название фильма : {0}, Дата американской премьеры : {1}'.
  ↳format(y['Название фильма'], y['Дата американской премьеры']))
```

Название фильма : «Taxi Driver», Дата американской премьеры : 8
 ↪ февраля 1976
 года

Напечатали название фильма и дату премьеры. Метод `json.loads()` преобразовал строку так, что к значениям 'Название фильма' и 'Дата американской премьеры' теперь возможно обратиться из функции `print()`.

Однако в этом случае в JSON была информация только по одному объекту. Чаше передают целый список:

```
x = '[{"Название фильма": "«Taxi Driver»", "Дата американской  

  ↪премьеры": "8 февраля 1976 года"}, {"Название фильма":  

  ↪"«After Hours»", "Дата американской премьеры": "11 сентября  

  ↪1985 года"}, {"Название фильма": "«Goodfellas»", "Дата  

  ↪американской премьеры": "9 сентября 1990 года"}]'
```

```
y = json.loads(x)
for i in y:
    print('Название фильма : {0}, Дата американской премьеры :  

  ↪{1}'.format(i['Название фильма'], i['Дата американской  

  ↪премьеры']))
```

Название фильма : «Taxi Driver», Дата американской премьеры : 8
 ↪ февраля 1976
 года

Название фильма : «After Hours», Дата американской премьеры : 11
 ↪ сентября 1985
 года

Название фильма : «Goodfellas», Дата американской премьеры : 9
 ↪ сентября 1990
 года

Метод `json.dumps()` (от англ. `dumps` — «разгрузки»), наоборот, конвертирует данные из Python в формат JSON. При работе с буквами кириллицы указывают аргумент `ensure_ascii=False` (строки запишутся как есть). Если `ensure_ascii=True`, все не ASCII символы в выводе будут экранированы последовательностями `\uXXXX`, и результатом будет строка, содержащая только ASCII символы.

```
out = json.dumps(y, ensure_ascii=False)
print(out)
```

```
[{"Название фильма": "«Taxi Driver»", "Дата американской ↵
↵премьеры": "8 февраля
1976 года"}, {"Название фильма": "«After Hours»", "Дата ↵
↵американской премьеры":
"11 сентября 1985 года"}, {"Название фильма": "«Goodfellas»", ↵
↵"Дата американской
премьеры": "9 сентября 1990 года"}]
```

3.4 Предобработка данных

Перед тем как приступить к работе над аналитической задачей, следует предварительно подготовить сами данные. Процесс подготовки данных для дальнейшего анализа называется **предобработка**. Заключается она в поиске проблем, которые могут быть в данных, и в решении этих проблем.

В программировании работает принцип GIGO (от англ. garbage in — garbage out, буквально «мусор на входе — мусор на выходе»). Это значит, что при ошибках во входных данных даже правильный алгоритм работы выдаёт неверные результаты.

Получаемые аналитиком данные не всегда соответствуют ожидаемому уровню качества. Человеческий фактор, ошибки системы или процесса выгрузки могут «испортить» их, то есть сделать непригодными для анализа.

Аналитики всегда готовят данные к дальнейшей работе с ними, так как в исходном виде они практически всегда содержат:

- ошибки ввода;
- технические ошибки;
- пропущенные значения;

- дубликаты;
- информационный шум и ненужную информацию.

Такие проблемы не позволят извлечь полезную информацию из данных, поэтому предобработка — важная часть работы аналитиков. Порой подготовка данных занимает больше времени, чем сам анализ.

Изучать методы предобработки данных лучше на практике. Основой для примеров предобработки данных будут две таблицы: `ratings.csv` и `movies.csv`.

Зачастую необходимая информация находится не в одной, а в сразу нескольких таблицах и вот почему:

1. Данные формируются в нескольких независимых процессах и хранятся в отдельных таблицах. Например, у нас есть сеть кинотеатров, которая работает в нескольких городах. Скорее всего, данные по кинотеатрам по каждому отдельному городу будут храниться в разных таблицах, которые будут формироваться независимо друг от друга. И объединять их в единую таблицу придется аналитику.

2. Хранить все данные в одной таблице накладно для ёмкости диска. Например, названия фильмов из наших данных хранятся в отдельной небольшой таблице, а в логах стоит идентификатор фильма. Это делается для того, чтобы уменьшить объем занимаемой памяти на диске. Названия могут растягиваться на миллионы строк, а числовой идентификатор фильма занимает гораздо меньше места. Именно поэтому логи с идентификаторами выигрывают у единой таблицы с названиями.

Подробнее опишем имеющиеся датасеты.

Таблица `movies.csv` — расшифровка идентификаторов фильмов.

Описание столбцов:

- `movieId` — идентификатор фильма;
- `title` — название фильма и год производства;

— genres — список жанров, к которым относится фильм.

Прочитаем датасеты в формате csv, где все значения разделены точкой с запятой в movies и запятой в ratings. Это наши исходные данные. Чтобы применить к ним все возможности языка Python и библиотеки Pandas, надо импортировать эту библиотеку и сохранить её в переменной. По сокращённому названию панельных данных (panel data), с которых начиналась Pandas, эту переменную принято называть pd:

```
movies = pd.read_csv('https://raw.githubusercontent.com/
↳datasciencefufu/course/main/data/movies.csv', sep=";")
movies.head()
```

	movieId	...	genres
0	1	...	Adventure Animation Children Comedy Fantasy
1	2	...	Adventure Children Fantasy
2	3	...	Comedy Romance
3	4	...	Comedy Drama Romance
4	5	...	Comedy

[5 rows x 3 columns]

Таблица ratings.csv — данные о выставленных фильмам оценках.

Описание столбцов таблицы:

- userId — идентификатор пользователя, который поставил фильму оценку;
- movieId — идентификатор фильма;
- rating — выставленная оценка;
- timestamp — время (в формате unix time), когда была выставлена оценка.

```
ratings = pd.read_csv('https://raw.githubusercontent.com/
↳datasciencefufu/course/main/data/ratings.csv', sep=",")
ratings.head()
```

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931

Unix time — формат описания моментов во времени, определяется как количество секунд, прошедших с полуночи (00:00:00) 1 января 1970 года. Можно воспользоваться методом `to_datetime` для приведения времени к стандартному виду. Первый аргумент — это столбик для преобразования, второй аргумент — это единицы, в которых указано время (в нашем случае это секунды — “s”):

```
time = pd.to_datetime(ratings['timestamp'],unit='s')
ratings['timestamp'] = time
ratings.tail()
```

	userId	movieId	rating	timestamp
100831	610	166534	4.0	2017-05-03 21:53:22
100832	610	168248	5.0	2017-05-03 22:21:31
100833	610	168250	5.0	2017-05-08 19:50:47
100834	610	168252	5.0	2017-05-03 21:19:12
100835	610	170875	3.0	2017-05-03 21:20:15

Прежде чем приступить к предобработке данных, изучим общую информацию о датасетах:

```
movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9757 entries, 0 to 9756
Data columns (total 3 columns):
#   Column      Non-Null Count  Dtype
---  -
0   movieId     9757 non-null   int64
1   title       9751 non-null   object
2   genres      9757 non-null   object
```



```
dtypes: int64(1), object(2)
memory usage: 228.8+ KB
```

Заметим, что количество строк в столбцах различно — значит, в данных есть пропуски. И прежде чем анализировать такие данные, их нужно обработать (см. Работа с пропусками).

```
ratings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100836 entries, 0 to 100835
Data columns (total 4 columns):
#   Column      Non-Null Count  Dtype
---  -
0   userId      100836 non-null  int64
1   movieId     100836 non-null  int64
2   rating       100836 non-null  float64
3   timestamp   100836 non-null  datetime64[ns]
dtypes: datetime64[ns](1), float64(1), int64(2)
memory usage: 3.1 MB
```

А в таблице ratings.csv количество строк во всех столбцах одинаковое — скорее всего, пропущенных значений нет.

3.4.1 Работа с пропусками

Аналитикам часто приходится работать с данными, в которых есть пропуски. Пропуски в данных появляются по разным причинам. Некоторые пропуски возникают случайно, например, автоматизированная система сбора информации дала сбой или пользователи забыли внести данные. Другие пропуски остаются в таблицах умышленно — для заполнения спецсимволами.

Способ работы с пропущенными значениями зависит от их типа. В датафрейме pandas столбцы могут содержать

категориальные или **количественные** данные. Они соотносятся с типами переменных в статистике. Категориальная принимает одно значение из ограниченного набора, а количественная — любое числовое значение в диапазоне (см. Категориальные и количественные, непрерывные и дискретные переменные).

Посмотрим сколько пропусков в нашем датасете:

```
ratings.isnull().sum()
```

```
userId      0
movieId      0
rating       0
timestamp    0
dtype: int64
```

В данных о выставленных фильмам оценках пропущенных значений нет.

```
movies.isnull().sum()
```

```
movieId      0
title         6
genres        0
dtype: int64
```

Восстановить значения в столбце 'title' не представляется возможным, но так как пропущенных значений мало, то следует **их** удалить с помощью метода `.dropna()`:

```
movies.dropna(subset=['title'], inplace=True)
```

Чтобы программа правильно интерпретировала пропуски, при чтении файла с помощью метода `read_csv` можно передать в параметр `na_values` значение или список значений, которые при чтении будут помечены как пропуски.

3.4.2 Удаление дубликатов

Посмотрим сколько дубликатов фильмов в таблице `movies`.

```
print('Дубликатов в таблице:', movies.duplicated(['movieId']).
      ↪sum())
```

Дубликатов в таблице: 9

Для удаления дубликатов используем метод `drop_duplicates`. Рассмотрим подробнее параметры данного метода:

- в параметре `subset` указываем один или несколько столбцов, по комбинации которых хотим удалить дубликаты;
- с помощью параметра `keep` указываем, какой из дубликатов оставить (например, первый или последний);
- параметр `inplace` указывает, что изменения нужно сохранить в датафрейме, к которому применяется метод (в нашем случае — в датафрейме `movies`).

```
movies.drop_duplicates(subset = 'movieId', keep = 'first',
      ↪inplace = True)
movies.head()
```

	movieId	...	genres
0	1	...	Adventure Animation Children Comedy Fantasy
1	2	...	Adventure Children Fantasy
2	3	...	Comedy Romance
3	4	...	Comedy Drama Romance
4	5	...	Comedy

[5 rows x 3 columns]

Убедимся, что теперь в нашем датасете нет дубликатов:

```
print('Дубликатов в таблице:', movies.duplicated(['movieId']).
      ↪sum())
```

Дубликатов в таблице: 0

3.4.3 Объединение таблиц

Объединение датафреймов с помощью метода `merge` имеет особенности: есть ситуации, которые в результате приводят к дублированию строк в объединенном датасете. Поэтому ранее мы избавились от дубликатов.

Метод **`merge()`** применяют к таблице, к которой присоединяют другую.

```
joined = left_df.merge(right_df, on='', how='')
```

У метода следующие аргументы:

- `left_df / right_df` — имена объединяемых `DataFrame` или `Series`. К “правому” датафрейму присоединяем “левый” (в нашем примере “левый” датафрейм — `ratings`, “правый” — `movies`);
- `on` — название общего столбца в двух соединяемых таблицах: по нему происходит слияние. Для объединения по нескольким столбцам используют `on = ['col1', 'col2']`;
- `how` — параметр объединения записей. Он может иметь четыре значения: `left`, `right`, `inner` и `outer`. Если аргумент `how` принимает значение `left`: тогда в итоговую таблицу будут включены `id` из левой таблицы. Если же `how` принимает значение `right`, тогда итоговая таблица включает `id` из правой таблицы.

Рассмотрим датафреймы внимательнее: у `ratings` и `movies` есть общий столбец `movieId`. Значит, можно объединить эти датафреймы в одну таблицу, используя метод `merge`.

Объединим таблицы `ratings` и `movies` со следующими условиями:

- `ratings` — таблица, к которой будем присоединять другую таблицу;
- `movies` — таблица, которую присоединяем к `ratings`;

- ‘movieId’ — общий столбец в двух таблицах, по нему будем объединять;
- how=‘left’ — movieId таблицы ratings включены в итоговую таблицу joined.

```
joined = ratings.merge(movies, on='movieId', how='left')
joined.head()
```

	userId	...	genres
0	1	...	Adventure Animation Children Comedy Fantasy
1	1	...	Comedy Romance
2	1	...	Action Crime Thriller
3	1	...	Crime Mystery Thriller
4	1	...	Children Comedy

[5 rows x 6 columns]

3.5 Выводы

В этой главе были описаны основные способы сбора и предобработки данных на Python. Подробны рассмотрены DataFrame и Series – основные объекты библиотеки **Pandas**. Определены способы фильтрации данных при помощи регулярных выражений (библиотека **re**), методы парсинга страниц и приемы обработки данных в формате JSON. В качестве методов предобработки данных описаны методы обработки пропусков, способы нахождения и удаления дубликатов, варианты объединения таблиц.

После изучения данного раздела вы можете:

- извлекать данные из файлов различных форматов;
- фильтровать данные;
- обогащать имеющиеся данные путем парсинга страниц;
- удалять пропуски;
- заменять пропуски;
- удалять дубликаты;
- объединять таблицы.

Глава 4 Визуализация данных

Одним из самых творческих этапов в работе с данными является этап **визуализации**. Использование графиков и диаграмм позволяет более наглядно представить данные и продемонстрировать полученные результаты их анализа.

Визуализация данных — это не просто создание причудливой графики для отчетов. Визуализация используется в повседневной работе на всех этапах работы над проектом.

Начнем с того, что визуальное исследование данных — это первое, что обычно делают при работе с новой задачей. Аналитики проводят предварительные проверки и анализ с использованием графиков и таблиц, чтобы обобщить имеющиеся данные. Людям гораздо удобнее понять основные моменты таким образом, чем читать множество строк необработанных данных. Удивительно, как много понимания можно получить из, казалось бы, простых диаграмм, созданных с помощью доступных инструментов визуализации.

При анализе производительности модели или результатов отчёта аналитики также часто используют диаграммы и изображения. Иногда для интерпретации сложной модели необходимо проецировать многомерные пространства на более визуально понятные 2D или 3D-фигуры.

В общем, визуализация — это относительно быстрый способ узнать что-то новое об имеющихся данных. Таким образом, жизненно важно изучить его наиболее полезные методы и сделать их частью своего повседневного набора инструментов при анализе данных.

В этом разделе будут рассмотрены некоторые примеры визуального исследования данных с использованием популярных библиотек, таких как Pandas, Matplotlib, Seaborn и Plotly.

4.1 Pandas

Для визуализации результатов анализа удобно использовать библиотеку **Pandas**. Возможностей Pandas достаточно, чтобы построить основные типы графиков и диаграмм.

Будем рассматривать визуализацию на данных встроенного датасета `tips` — он содержит информацию о посещениях ресторана и оставленных чаевых. В датасете семь признаков:

- `total_bill` — общая сумма счета;
- `tip` — размер чаевых;
- `sex` — пол клиента;
- `smoker` — является ли клиент курящим;
- `day` — день недели;
- `time` — время (обед или ужин);
- `size` — количество посетителей, обедавших за столом.

```
import seaborn as sns

data = sns.load_dataset("tips")
data
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
..	...	...	...	...	...	...	...

239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

[244 rows x 7 columns]

Основным методом для быстрого создания графиков и диаграмм в Pandas является метод **plot()**:

```
data.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4c302b29d0>
```

Если обратиться к методу `plot()` без указания параметров, то получится график для всех количественных признаков в датафрейме (см. Категориальные и количественные, непрерывные и дискретные переменные).

По умолчанию на графике, созданном методом `plot()`, по оси *x* отложены порядковые номера данных в датафрейме (индексы), а по оси *y* — значения признаков.

4.1.1 Отображение нескольких графиков

В методе `plot()` есть ещё один полезный параметр **`subplots`**, который поможет разместить несколько графиков в одной рабочей области. Параметр принимает логические значения `True` или `False`. Если установим значение `True`, то разместим сразу два графика для признаков из нашего датафрейма:

```
data[['total_bill', 'tip']].plot(kind='hist',
                                grid=True,
                                subplots=True,
                                title=['Общая сумма счёта',
    ↪ 'Сумма чаевых'],
                                legend=False)
```

4.2 Matplotlib

Для продвинутой работы с графиками (в том числе гистограммами) импортируют библиотеку `matplotlib` (от англ. `math` — «математический», `plot` — «график, диаграмма», `library` — «библиотека»). `Matplotlib` — это популярная профессиональная библиотека для создания графиков и изображений. Нам понадобится модуль `pyplot`, он содержит функции и методы для визуализации данных:

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

При визуализации данных с помощью `matplotlib` рекомендуется использовать объектно-ориентированный подход. Его суть заключается в создании объектов, характеристики которых можно легко изменять, этот подход позволяет создавать объекты-контейнеры. Такой контейнер может содержать другие объекты, например, координатную плоскость или графики. Удобно, что такие объекты можно легко изменять и настраивать независимо друг от друга.

Посмотрим на примерах, как работает ООП в визуализации. Создадим объект с именем `fig` (фигура) — это контейнер для будущего графика:

```
fig = plt.figure()
```

<Figure size 432x288 with 0 Axes>

Добавим в объект `fig` новый объект - одну координатную плоскость. Она начинается в левом нижнем углу без отступов (координаты 0, 0) и занимает отведённое место в объекте `fig` (ширина и высота равны 1). В созданной координатной плоскости

можно разместить любую графическую информацию. Поместим туда гистограмму с помощью метода `hist()`:

```
axes = fig.add_axes([0, 0, 1, 1])
axes.hist(data['total_bill'])
```

```
(array([ 7., 42., 68., 51., 31., 19., 12.,  7.,  3.,  4.]),
 array([ 3.07 ,  7.844, 12.618, 17.392, 22.166, 26.94 , 31.714, 36.488,
        41.262, 46.036, 50.81 ]),
 <a list of 10 Patch objects>)
```

У метода `hist()` есть параметры `color` и `bins`: с их помощью можем поменять цвет диаграммы и увеличить количество столбцов для демонстрации распределения. Такие возможности отличают визуализацию в Matplotlib от построения графиков в Pandas. В Matplotlib создается объект-холст и все графики строятся внутри него. В Pandas можно строить графики сразу — без создания дополнительных объектов.

```
fig = plt.figure()
axes = fig.add_axes([0, 0, 1, 1])
axes.hist(data['total_bill'], bins = 20, color = 'green')
axes.set_title('Общая сумма счёта')
```

Можем попробовать нарисовать графики и других видов, применив к объекту `axes` другие методы. Например, создадим диаграмму рассеяния (**scatter plot**).

Диаграмма рассеяния показывает распределение элементов множества в плоскости между двумя переменными. Один признак откладывается по оси x , а второй признак — по оси y . Для этого воспользуемся методом `scatter()`:

```
fig = plt.figure()
axes = fig.add_axes([0, 0, 1, 1])
axes.scatter(x=data['total_bill'], y=data['tip'], marker='o')
axes.set_title('Корреляция общей суммы счета и суммы чаевых')
```

```
Text(0.5, 1.0, 'Корреляция общей суммы счета и суммы чаевых')
```

Диаграмма рассеяния показывает как между собой могут быть связаны сумма счёта и размер чаевых. Каждая точка на графике — это один столик в ресторане. По графику видно, что чем выше сумма чека, тем выше чаевые, однако зависимость может быть обманчивой. Если просто смотреть на таблицу с данными, то заметить такую связь очень сложно. Особенно, если у аналитика пока нет большого опыта работы с данными.

4.2.1 Добавление легенды графика

При отображении на графике нескольких столбцов датасета необходимо добавить пояснения, какой информации соответствует тот или иной цвет. Такие комментарии называют легендой. Чтобы создать её, используем метод **legend()** для объекта `axes`. Легенду можно поместить в разных местах на координатной плоскости, поэтому специальным параметром `loc` зададим место её размещения на графике:

```
axes.legend(loc = 1)
```

Если легенда перекрывает часть данных, то можно увеличить максимальное значение оси ординат (ось `y`) с помощью метода `set_ylim()` к объекту `axes`:

```
axes.set_ylim(0, 60)
```

```
(0.0, 60.0)
```

4.2.2 Создание нескольких графиков

В одном пространстве (объекте `fig`) можно разместить сразу несколько графиков методом `subplots()`. Посмотрим, как использовать метод при визуализации в ООП. В параметрах фигуры необходимо указать количество строк и столбцов, в которых будут размещаться графики. Можно даже задать размер фигуры в дюймах (ширина*высота, 1 дюйм = 2,54 см):

```
fig, axes = plt.subplots(1, 3, figsize = (12,4)).
```

После выполнения кода переменная `fig` содержит общий объект, а переменная `axes` содержит список, состоящий из трёх внутренних объектов. Обращаться к элементам получившегося списка можно по индексам (от 0 до 2):

```
fig, axes = plt.subplots(1, 3, figsize=(12, 4))
# В параметрах фигуры необходимо указать количество строк и
# столбцов, в которых будут размещаться графики
axes[0].hist(data.total_bill)
axes[0].set_title('Общая сумма счёта')
axes[1].hist(data.tip)
axes[1].set_title('Сумма чаевых')
axes[2].scatter(x=data.total_bill, y=data.tip, marker='o')
axes[2].set_title('Корреляция общей суммы счёта и суммы чаевых')
```

```
Text(0.5, 1.0, 'Корреляция общей суммы счёта и суммы чаевых')
```

4.3 Seaborn

Для создания визуализаций со сложными настройками параметров отображения данных используется библиотека **Seaborn**.

Seaborn — графическая многофункциональная библиотека, которая написана на основе Matplotlib. Импорт библиотеки Seaborn был сделан еще в самом начале при загрузке датасета tips.

Перед созданием гистограммы установим стандартные параметры отображения графиков с помощью команды **sns.set()**:

```
sns.set()
```

В библиотеке Seaborn названия методов для создания уже знакомых диаграмм будут отличаться. Самый простой график, показывающий линию вдоль двухмерной плоскости и используется для просмотра корреляции между двумя величинами — **lmlplot**. Seaborn позволяет установить высоту, цветовую палитру для создаваемого графика.

```
sns.lmlplot(x="total_bill", y="tip", data=data, height=4,   
→palette="dark")
```

```
<seaborn.axisgrid.FacetGrid at 0x7f4c2fa96550>
```

График оценки плотности ядра — Kernel Density Estimate (KDE) — является сглаженной непрерывной версией гистограммы и используется для визуализации распределения вероятности одномерных данных.

```
sns.kdeplot(data['tip'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4c277f6b50>
```

Чтобы построить несколько распределений одним вызовом функции используется метод **distplot()** — комбинация **kdeplot** и гистограмм. В результате мы получим график плотности поверх соответствующей гистограммы.

```
sns.distplot(data.total_bill)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4c24494310>
```

Seaborn позволит нам разместить на одном графике гистограммы распределения двух признаков и показать связь между ними, то есть можно построить две гистограммы для двух признаков (общий счёт заказа, сумма чаевых) и связывающую их диаграмму рассеяния. Используем метод **jointplot()**, чтобы нарисовать нужную визуализацию:

```
with sns.axes_style('white'):  
    sns.jointplot(x='total_bill', y='tip', data=data, kind='hex')
```

Посмотрим, какие параметры можно отрегулировать на графике. Для начала изменим характер представления информации в центральной части графика — сделаем это с помощью параметра `kind`. Добавим на график линию тренда:

```
sns.jointplot(x='total_bill', y='tip', data=data, kind='reg')
```

```
<seaborn.axisgrid.JointGrid at 0x7f4c242ac790>
```

Ещё один способ визуализировать распределение признаков в датасете построить график, который будет показывать отношения сразу между всеми парами переменных. Где-то это будет гистограмма, а где-то точечная диаграмма рассеяния. График строится методом **pairplot()**:

```
sns.pairplot(data)
```

```
<seaborn.axisgrid.PairGrid at 0x7f4c2fb0a590>
```

Для отображения корреляции различных числовых атрибутов в наборе данных удобно использовать **тепловые карты** (heatmap). Для создания тепловой карты создадим матрицу корреляции и передадим её в параметр тепловой карты. Для отображения подписей установим значение параметра аннотации True.

```
sns.heatmap(data.corr(), annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4c2420f710>
```

При визуализации информации о категориальных (качественных) признаках методами библиотек Pandas и Matplotlib приходится предварительно обрабатывать датафрейм и группировать нужные данные. В Seaborn такой проблемы нет: при построении графиков можно обращаться к столбцам с категориальными признаками напрямую. Для построения гистограммы распределения категориального признака можно воспользоваться методом **countplot()**:

```
sns.countplot(x='day', data=data)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4c23f84050>
```

Категориальные значения при построении графиков можно использовать в качестве одного из параметров метода **barplot()**. Он сам сгруппирует данные в зависимости от значения категориальной переменной. По умолчанию для каждой группы вычисляется среднее арифметическое значение заданной количественной переменной (вместо среднего значения можно вычислить и любой другой показатель — настроить это можно в параметре **estimator**). При необходимости гистограммы можно строить как по горизонтали, так и по вертикали.

```
sns.barplot(x='sex', y='total_bill', data=data)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4c23e5e3d0>
```

```
ax = sns.barplot(x='total_bill', y='sex', data=data)
```

График показывает сравнение среднего чека у мужчин и женщин: в среднем гости-мужчины оставляли в ресторане больше денег, чем женщины.

Теперь посмотрим как зависит сумма, которую персонал ресторана получает в виде чаевых, от различных переменных. Проще всего это сделать используя гистограммы подмножеств **FacetGrid**:

```
import numpy as np

data['tip_pct'] = 100 * data['tip'] / data['total_bill']
grid = sns.FacetGrid(data, row="sex", col="time",
    ↪margin_titles=True)
grid.map(plt.hist, "tip_pct", bins=np.linspace(0, 40, 15));
```

4.4 Plotly

Ещё один интересный продукт, с помощью которого можно создавать интерактивные графики и диаграммы — библиотека Plotly. Это коммерческий продукт, но многие функции доступны для бесплатного использования.

Библиотека Plotly будет полезна для того чтобы:

- получать отдельное значение признака в интерактивном режиме;
- регулировать отображение признаков (отключать и включать признаки на графике);
- увеличивать фрагмент графика;
- сохранить график как изображение в формате PNG.

Большой популярностью среди аналитиков пользуется модуль Plotly Express. Plotly Express — это высокоуровневый API для визуализации данных. С помощью него можно упростить и ускорить создание графических объектов вызовом соответствующих функций. Plotly Express поддерживает широкий спектр диаграмм, включая многомерные отображения, матрицы рассеяния (SPLOM), графики параллельных координат и параллельных категорий.

С помощью Plotly Express можно добавить интерактивность базовым графикам **bar**.

```
import plotly.express as px

fig = px.bar(data, x="sex", y="total_bill", color="smoker",
             ↪barmode="group", facet_row="time",
             facet_col="day", category_orders={"day": ["Thur",
             ↪"Fri", "Sat", "Sun"],
                                               "time": ["Lunch",
             ↪"Dinner"]})
fig.show()
```

Используя данный модуль можно отобразить на одном графике несколько гистограмм для различных значений одного столбца. А с помощью ключевого слова `marginal` рядом с гистограммой рисуется подзаголовок, визуализирующий распределение.

```
fig = px.histogram(data, x="total_bill", color="sex",  
    ↪marginal="box",  
    hover_data=data.columns)  
fig.show()
```

Для визуализации взаимосвязи между несколькими наборами категорий в наборе данных удобно использовать такой инструмент как **параллельные категории**.

```
fig = px.parallel_categories(data, color="size",  
    ↪color_continuous_scale=px.colors.sequential.Inferno)  
fig.show()
```

В модуле Plotly Graph Objects есть специальные объекты под названием graph objects (графические объекты). Один из примеров 3D-диаграмм с выбором опций есть на сайте plotly:

```
import plotly.graph_objects as go
import pandas as pd

# load dataset
df = pd.read_csv("https://raw.githubusercontent.com/plotly/
↳ datasets/master/volcano.csv")

# create figure
fig = go.Figure()

# Add surface trace
fig.add_trace(go.Surface(z=df.values.tolist(),
↳ colorscale="Viridis"))

# Update plot sizing
```

```

fig.update_layout(
    width=800,
    height=900,
    autosize=False,
    margin=dict(t=0, b=0, l=0, r=0),
    template="plotly_white",
)

# Update 3D scene options
fig.update_scenes(
    aspectratio=dict(x=1, y=1, z=0.7),
    aspectmode="manual"
)

# Add dropdown
fig.update_layout(
    updatemenus=[
        dict(
            type="buttons",
            direction="left",
            buttons=list([
                dict(
                    args=["type", "surface"],
                    label="3D Surface",
                    method="restyle"
                ),
                dict(
                    args=["type", "heatmap"],
                    label="Heatmap",
                    method="restyle"
                )
            ])
        ),
        pad={"r": 10, "t": 10},
        showactive=True,
        x=0.11,
        xanchor="left",
        y=1.1,
        yanchor="top"
    ),
]

```

```
)  
  
# Add annotation  
fig.update_layout(  
    annotations=[  
        dict(text="Trace type:", showarrow=False,  
            x=0, y=1.08, yref="paper", align="left")  
    ]  
)  
  
fig.show()
```

4.5 Выводы

В данной главе описаны некоторые методы визуального исследования данных и приведены примеры их реализации в библиотеках Pandas, Matplotlib, Seaborn и Plotly. Рассмотрены различные способы работы с графиками, в том числе отображение нескольких графиков в одной рабочей области, добавление легенды и линии тренда.

Используя полученные знания, вы можете:

- визуализировать распределение признаков в датасете;
- выбирать подходящий инструмент визуализации;
- строить гистограммы, в том числе гистограммы распределения категориального признака;
- строить диаграммы рассеяния (scatter);
- строить графики плотности распределения;
- строить тепловые карты;
- визуализировать взаимосвязи между несколькими наборами категорий, используя инструмент "параллельные категории";
- создавать сложные графические объекты, в том числе 3D-диаграммы.

Заключение

Анализом данных человечество занимается с незапамятных времен. Люди всегда сопоставляли и анализировали имеющиеся данные, пытаясь извлечь из них новую полезную информацию - знание. Ученые кропотливо и аккуратно производили наблюдения, фиксировали результаты и обнаруживали аналитические зависимости одних величин от других. Так, Исаак Ньютон математически выводил свои три закона механики с целью поиска обоснований законам небесной механики Иоганна Кеплера. Кеплер выводил законы для создания теории, которая бы корректно описывала опытные данные датского астронома Тихо Браге, который измерил положение всех на тот момент известных планет и записал их координаты, но не сумел окончательно сформулировать закон движения планет относительно Солнца. Таким образом, и Ньютон, и Кеплер фактически проделали работу аналитиков больших данных, сумев выявить и зафиксировать на бумаге закономерности, скрытые в гигантском объеме данных, полученных великими предшественниками. Именно выявление закономерностей и зависимостей помогли этим гениальным ученым сформулировать базовые законы небесной механики.

Если раньше сбор материала, его систематизация, анализ и выдвижение гипотез производились вручную и требовали большого количества времени и усилий, то сегодня на вооружении ученых и специалистов по обработке данных новые мощные вычислительные инструменты. Для обозначения отрасли знаний и специалистов, которые должны разбираться в огромных массивах больших данных, был введен термин «наука о данных» (термин Data Science прозвучал только в 1974-м).

Современную науку о данных можно разделить на два больших раздела: анализ данных (Data Science) и машинное обучение (Machine Learning). С инструментами аналитиков данных (дата-сайентистов) вы познакомились в этом пособии: узнали что такое распределение, познакомились с базовой проверкой данных, научились строить гистограммы и находить взаимосвязи между разными значениями путем построения графиков разброса (scatter) и матриц корреляции. На практических примерах вы изучили методы сбора и предобработки данных, поняли как статистика и теория вероятностей помогают решать задачи бизнеса. Теперь вы знаете, что аналитики не только производят расчёты, но и осуществляют поиск необычных явлений, артефактов и ошибок в данных. В этом дата-сайентистам помогают построение и анализ графиков: визуальное представление позволяет буквально заглянуть в данные, обнаружить проблемы и сделать зачастую неожиданные выводы, которые могут принести едва ли не больше пользы, чем решение первоначальной задачи. Специалисты по машинному обучению занимаются другим классом задач — обучают машины делать автоматические предсказания. Например, предугадывать изменение спроса на тот или иной товар в зависимости от различных факторов. Если вам интересно изучить методы машинного обучения, то лучше всего начинать с библиотеки NumPy. NumPy — это библиотека машинного обучения Python предоставляет возможности для обработки данных и чисел. Возможности библиотеки позволяют создать объект N-мерного массива, в который пользователи могут помещать свои данные, и предлагают функции преобразования этих данных. Объект массива также можно легко адаптировать к другим базам данных, что делает его хорошим инструментом для машинного обучения. Краткий обзор функциональных возможностей библиотеки NumPy приведен в Приложении.

Профессию дата-сайентиста из года в год называют одной из самых востребованных. Умение собирать, обрабатывать и интерпретировать информацию пригодится в любой сфере — неважно, работаете ли вы в ретейле или медицине. Все больше компаний нанимают аналитиков и готовы хорошо им платить. По запросу «аналитик данных» на hh.ru есть порядка 10 тысяч вакансий: junior может рассчитывать на зарплату от 70 тысяч, опытный специалист (senior) — до 200 тысяч рублей. Аналитики помогают бизнесу развиваться эффективнее. Согласно опросам, 65% компаний не в состоянии проанализировать имеющиеся данные, при этом 89% из них уверены, что если бы они могли это сделать, то получили бы конкурентное преимущество. Может показаться, что аналитик данных не самая творческая профессия: ему каждый день приходится заниматься рутинным сбором и анализом. Однако именно его работа может дать удивительные результаты — обнаружить проблемы, предсказать будущее, сделать явным то, что никто иначе бы не узнал.

Список литературы

1. Beautiful Soup Documentation — Beautiful Soup 4.4.0 documentation [Электронный ресурс]. — Режим доступа: <https://beautiful-soup-4.readthedocs.io/en/latest/>. — Дата доступа: 01.06.2021.
2. Colaboratory — Google [Электронный ресурс]. — Режим доступа: <https://research.google.com/colaboratory/faq.html>. — Дата доступа: 01.06.2021.
3. D.Reinsel, J.Gants, J.Rydning The Digitalization of the World // IDC White Paper [Электронный ресурс]. — Режим доступа: <https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf>. — Дата доступа: 12.03.2021.
4. Explanation of 3V's Model of Big data Given by Doug Laney [Электронный ресурс]. — Режим доступа: <https://www.cosoit.com/explanation-of-3v-model-of-big-data>. — Дата доступа: 15.05.2021.
5. HTTP Documentation [Электронный ресурс]. — Режим доступа: <https://httpwg.org/specs/>. — Дата доступа: 15.04.2021.
6. Json — JSON encoder and decoder — Python 3.9.5 documentation [Электронный ресурс]. — Режим доступа: <https://docs.python.org/3/library/json.html>. — Дата доступа: 17.05.2021.
7. Lynch C. Big data: How do your data grow?//Nature. — 2008.— Т. 455. — №. 7209. — p.28-29.
8. Matplotlib: Python plotting — Matplotlib 3.4.2 documentation [Электронный ресурс]. — Режим доступа: <https://matplotlib.org/stable/index.html>. — Дата доступа: 28.04.2021.
9. NumPy [Электронный ресурс]. — Режим доступа: <https://numpy.org>. — Дата доступа: 19.04.2021.

10. Pandas - Python Data Analysis Library [Электронный ресурс]. – Режим доступа: <https://pandas.pydata.org>. – Дата доступа: 20.04.2021.
11. Plotly Python Graphing Library | Python | Plotly [Электронный ресурс]. – Режим доступа: <https://plotly.com/python/>. – Дата доступа: 10.04.2021.
12. Project Jupyter | Documentation [Электронный ресурс]. – Режим доступа: <https://jupyter.org/documentation>. – Дата доступа: 18.03.2021.
13. Re — Regular expression operations — Python 3.9.5 documentation [Электронный ресурс]. – Режим доступа: <https://docs.python.org/3/library/re.html>. – Дата доступа: 12.03.2021.
14. Requests: HTTP for Humans — Requests 2.25.1 documentation [Электронный ресурс]. – Режим доступа: <https://docs.python-requests.org/en/master/index.html>. – Дата доступа: 17.01.2021.
15. SciPy.org — SciPy.org [Электронный ресурс]. – Режим доступа: <https://www.scipy.org>. – Дата доступа: 17.04.2021.
16. Seaborn: statistical data visualization — seaborn 0.11.1 documentation [Электронный ресурс]. – Режим доступа: <https://seaborn.pydata.org>. – Дата доступа: 10.05.2021.
17. TIOBE Index [Электронный ресурс]. – Режим доступа: <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>. – Дата доступа: 17.03.2021.
18. Welcome to Python.org [Электронный ресурс]. – Режим доступа: <https://www.python.org>. – Дата доступа: 19.02.2021.
19. Исследование - Академия больших данных MADE [Электронный ресурс]. – Режим доступа: <https://data.mail.ru/pages/research/>. – Дата доступа: 19.05.2021.

20. Работа аналитиком данных [Электронный ресурс]. – Режим доступа: <https://hh.ru/vacancies/analitik-dannyh>. – Дата доступа: 02.06.2021.

21. Теория вероятностей [Электронный ресурс]: лекции для физиков / Е.Л. Кулешов; Дальневосточный федеральный университет. – Электрон. дан. – Владивосток: Издательский дом Дальневост. федерал. ун-та, 2018. – Режим доступа: Загл. с экрана.

22. Уэс Маккинли Python и анализ данных / Пер. с англ. Слинкин А. А. – М.: ДМК Пресс, 2015. – 482 с.: ил.

Приложение А

Линейная алгебра с NumPy

Анализ данных сложно представить без погружения в математику. Большая часть идей в машинном обучении построена на математических принципах. В данном разделе мы рассмотрим зачем аналитику линейная алгебра, что такое массивы и как Python работает с векторами и матрицами, используя библиотеку NumPy.

А.1 Массивы

Основной структурой и объектом не только в библиотеке NumPy, но и в машинном обучении являются массивы.

Массив — это структура данных, содержащая упорядоченный набор значений (элементов), идентифицируемых по индексу или набору индексов.

В библиотеке NumPy содержатся функции и методы для математических операций над объектами **ndarray** — N-мерных однородных массивов с заранее заданным количеством элементов. Массив называют однородным, поскольку он объединяет элементы одного типа данных. **Размерностью** массива называют количество индексов, необходимых для однозначного определения его элементов. Первый элемент в массиве Python имеет индекс 0.

Массивы могут быть:

- одномерными (вектор);
- двумерными (матрица);
- многомерными.

Массив в Python задаётся с помощью функции `array()` перечислением элементов внутри квадратных скобок `[ ]` через запятую:

```
import numpy as np

arr = np.array([1,2,3,4,5,6])
```

Функция `array()` позволяет создать массив с любым количеством измерений (осей). Создание двумерного массива происходит следующим образом:

```
arr = np.array([[1,2,3],[4,5,6],[7,8,9]])
```

Узнать размерность массива можно с помощью функции `shape()`. `Shape` — это обычный кортеж языка Python.

```
np.shape(arr)
```

 $(3, 3)$

Можно создавать массивы из нулей (`np.zeros`) или из единиц (`np.ones`):

```
A = np.ones((7, 9))
A
```

[illegible]

Также часто используется создание векторов из натуральных чисел:

```
vec = np.arange(15)
vec
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

И случайно сгенерированные массивы:

```
r = np.random.rand(3, 4)
r
```

```
array([[0.18400186, 0.04549867, 0.03347525, 0.058673  ],
       [0.56254952, 0.28285337, 0.18458058, 0.90245314],
       [0.36086068, 0.20261408, 0.72518862, 0.48448959]])
```

Важное отличие массивов NumPy от списков Python: при создании среза NumPy не копирует данные в новый массив, а создает представление исходного массива. В памяти по-прежнему находится один массив, а в срезе хранятся только указатели, которые ссылаются на данные из исходного массива.

Работа с одномерными массивами похожа на работу со списком. Перечислим ключевые моменты индексирования массивов в NumPy:

- индексация массива начинается с 0;
- для обращения к элементу массива по индексу указываем имя переменной, в которой хранится массив, и индекс в квадратных скобках;
- допускается использование отрицательных индексов;
- для обращения к нескольким идущим подряд элементам массива создается срез, где указывается индекс первого элемента среза и индекс элемента, следующего за последним, разделенные двоеточием;

— при создании среза можно задать шаг, в этом случае в срез будут включены не все элементы, а только отстоящие друг от друга на величину шага.

Рассмотрим некоторые примеры индексации элементов массива на примере массива `arr`.

```
print(arr)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
arr[2, 1] # выбрать 1 элемент
```

```
8
```

```
arr[1] # выделить строку
```

```
array([4, 5, 6])
```

```
arr[:, 1] # выделить столбец
```

```
array([2, 5, 8])
```

```
arr[:, -1] # выделить последний столбец
```

```
array([3, 6, 9])
```

```
arr[:, ::2] # выделить все столбы с чётными номерами
```

```
array([[1, 3],
       [4, 6],
       [7, 9]])
```

```
arr[(arr % 2) == 1] # логическая индексация: выделить нечётные
↪элементы
```

```
array([1, 3, 5, 7, 9])
```

A.2 Одномерные массивы: вектор

Вектор — это набор чисел, записанных в определенном порядке, например, в столбик или в строчку.

В Python удобно представить вектор как одномерный массив NumPy. Сложение и вычитание векторов в Python происходит при помощи знаков $+$ и $-$:

```
import numpy as np
a = np.array([1, 4, 7, 8])
b = np.array([8, 5, 2, 1])
print(a+b)
```

```
[9 9 9 9]
```

Для умножения вектора на число в Python диспользуют оператор $*$:

```
import numpy as np
a = np.array([10, 8, 5, 1])
print(a*2)
```

```
[20 16 10  2]
```

Вектор можно преобразовать в матрицу:

```
print(a.reshape(2,2))
```

```
[[10  8]
 [ 5  1]]
```

А.3 Двумерные массивы: матрица

Матрицы похожи на двумерные векторы. Их удобно использовать для работы с системами уравнений и применения в разных методах машинного обучения.

Матрица — набор чисел, расположенных по строкам и столбцам, как в таблице. Машинное обучение состоит из целого комплекса разных линейных операций, поэтому на практике любая задача решается через последовательность простых линейных задач.

Двумерная матрица состоит из строк и столбцов, чтобы обратиться к элементу матрицы необходимо указать имя матрицы и в квадратных скобках два индекса: первый индекс отвечает за номер строки, второй — за номер столбца. В Python задать матрицу можно с помощью функции `array`:

```
arr = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(arr.shape)
print('a_23 = ', arr[1,2],'; a_32 = ', arr[2,1])
```

(3, 3)

a_23 = 6 ; a_32 = 8

Или используя метод `np.matrix()`:

```
arr2 = np.matrix("1,7;2,5;1,3")
print(arr2.shape)
print('a_21 = ', arr2[1,0],'; a_12 = ', arr2[0,1])
```

(3, 2)

a_21 = 2 ; a_12 = 7

Матрицы складываются и вычитаются поэлементно. Складывать и вычитать можно только матрицы одинакового размера. Две матрицы могут быть сложены или вычтены одна

из другой, только если у матриц одинаковое количество рядов и столбцов.

```
A = np.matrix("1,8;3,7;2,4")
B = np.matrix("6,-7;2,5;4,0")
print(A+B)
```

```
[[ 7  1]
 [ 5 12]
 [ 6  4]]
```

Умножение матрицы на отдельное число выполняют так же, как и умножение вектора на число.

```
print(B*5)
```

```
[[ 30 -35]
 [ 10  25]
 [ 20   0]]
```

Аналогично выполняется деление:

```
print(A/2)
```

```
[[0.5  4. ]
 [1.5  3.5]
 [1.   2. ]]
```

Транспонирование матрицы — это процесс, при котором строки матрицы записываются в столбцы, а столбцы — в строки, и матрица переворачивается. То есть получается её зеркальное отображение по главной диагональной линии, которая начинается в верхнем левом и идёт в правый нижний угол.

```
print(arr)
print(arr.T)
```

```
[[1 2 3]
 [4 5 6]]
```

```
[7 8 9]]
[[1 4 7]
 [2 5 8]
 [3 6 9]]
```

Частный случай — это транспонирование векторов. Если транспонировать вектор-столбец, получится вектор-строка и наоборот.

```
A = np.matrix("1, 4, 3, 5")
print(A.T)
```

```
[[1]
 [4]
 [3]
 [5]]
```

Умножить две матрицы можно только в том случае, если число столбцов в одной матрице равно числу строк во второй либо в одном из массивов размер 1. Например, матрица размера (7, 3) и вектор размера (7, 1). В этом случае при выполнении операции столбец будет как бы “дублироваться” для каждого столбца в первой матрице. Если же матрица A содержит m строк, а матрица B содержит n столбцов, то результатом их умножения будет матрица размера mn . При этом результаты умножения не только отличаются численно, но и имеют разный размер. Рассмотрим на конкретном примере:

```
A_1 = np.matrix("1,-2;-4,-5;1,3")
A_2 = np.matrix("7,9,3;8,4,-3")
print(np.dot(A_1,A_2))
print(np.dot(A_2,A_1))
```

```
[[ -9   1   9]
 [-68 -56   3]
 [ 31  21  -6]]
[[-26 -50]
 [-11 -45]]
```

Можно также писать:

```
print(np.dot(A_1,A_2))
print(np.dot(A_2,A_1))
```

```
[[ -9   1   9]
 [-68 -56   3]
 [ 31  21  -6]]
[[-26 -50]
 [-11 -45]]
```

или так:

```
print(A_1@A_2)
print(A_2@A_1)
```

```
[[ -9   1   9]
 [-68 -56   3]
 [ 31  21  -6]]
[[-26 -50]
 [-11 -45]]
```

Можно не только складывать, умножать, вычитать и транспонировать сами массивы NumPy, но и вычислять различные значения внутри массивов, такие как сумма элементов, произведение элементов и другие. В библиотеке NumPy для решения таких задач есть отличное решение — **универсальные функции**. Они выполняют поэлементные операции с данными объектов `ndarray`. Большинство универсальных функций относятся к типу унарных операций. **Унарные операции** — это операции, которые выполняются над каждым элементом массива по очереди. Приведем перечень часто используемых универсальных функций в таблице 6:

Универсальные функции NumPy

Функция	Описание
<code>np.abs()</code>	Абсолютное значение (модуль) целых или вещественных элементов массива
<code>np.isnan()</code>	Массив логических (булевых) значений, показывающий какие из элементов исходного массива являются NaN (не числами)
<code>np.modf()</code>	Дробные и целые части массива в виде отдельных массивов
<code>np.prod(имя, ось)</code>	Произведение всех элементов массива по заданной оси
<code>np.sqrt()</code>	Квадратный корень каждого элемента массива
<code>np.sum()</code>	Сумма всех элементов массива. Вторым аргументом можно задать ось, по которой будет определена сумма

Агрегирующие операции агрегируют информацию в строках, столбцах, во всем массиве и т. д. Самые популярные операции: суммирование `np.sum`, усреднение `np.mean`, медиана `np.median`, максимум `np.max` и минимум `np.min`.

```
print(a)
np.sum(a)
```

```
[10  8  5  1]
```

24

Матрицы можно суммировать по строкам и по столбцам:

```
print(arr)
np.sum(arr, axis = 1) # по строкам
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
array([ 6, 15, 24])
```

```
np.sum(arr, axis = 0) # по столбцам
```

```
array([12, 15, 18])
```

В результате суммирования получается вектор (размерность на 1 меньше, чем у исходной матрицы).

```
np.sum(arr, axis = 1).shape
```

```
(3,)
```

Можно указать `keepdims=True`, чтобы сохранить размерности:

```
np.sum(arr, axis = 1, keepdims=True).shape
```

```
(3, 1)
```

А.3.1 Оси

Для удобного перехода между размерностями часто добавляют новые оси. Так, вектор можно сделать матрицей с размером 1 по одной из размерностей.

```
ones = np.ones(4)
ones[:, np.newaxis]
```

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

```
ones[:, np.newaxis].shape
# вместо вектора с формой (4,) стала матрица с формой (4, 1)
```

```
(4, 1)
```

```
ones[np.newaxis, :].shape
# вместо вектора с формой (4,) стала матрица с формой (1, 4)
```

```
(1, 4)
```

С помощью функции **hstack()** можно присоединять данные по горизонтальной оси.

```
a = np.array([1,2,3])
b = np.array([4,5,6])
print('Добавление данных по горизонтали:', np.hstack((a, b)))
```

Добавление данных по горизонтали: [1 2 3 4 5 6]

Добавлять данные по вертикальной оси удобно используя **vstack()**.

```
print('Добавление данных по вертикали:', np.vstack((a, b)))
```

Добавление данных по вертикали:
[[1 2 3]
[4 5 6]]

Удаляют строки при помощи **delete()**:

```
np.delete(arr, 1, axis = 0)
```

```
array([[1, 2, 3],
       [7, 8, 9]])
```

Аналогично можно удалять столбцы:

```
np.delete(arr, 0, axis = 1)
```

```
array([[2, 3],
       [5, 6],
       [8, 9]])
```

А.4 Проверочные задания

1. **Ряд квадратов** Вывести квадраты первых десяти натуральных чисел
2. **По порядку**
 - а) Вывести целочисленный массив размером 10*10 со значениями от 1 до 10 в каждой строке;
 - б) Вывести таблицу умножения от 1 до 10.
3. **Сортировки** Дан массив случайных чисел.

```
rand = np.random.RandomState(28)
matrix = rand.randint(10, 100, [5,7])
print(matrix)
```

```
[[11 15 32 42 13 97 22]
 [74 61 33 22 79 18 34]
 [53 98 17 11 88 55 70]
 [44 61 47 80 78 38 25]
 [36 14 17 22 54 86 42]]
```

Необходимо с помощью `sort()`:

- а) Отсортировать массив по строкам;
- б) Отсортировать массив по столбцам.

4. **Смена знака** Дан массив случайных чисел от 1 до 100 размером 10×10 . Необходимо поменять знак у тех элементов, которые меньше 50.

```
m = np.random.randint(low=1, high=100, size=[10,10])
print(m)
```

```
[[73 57 59 75 94 83 75 25  2 84]
 [48 60 22 30  5 44 46 52 84 39]
 [97 35 60 46 23 47 69 98 98 32]
 [56 41 22 57 53 60 63 67  7 89]
 [83 12 62 72  9 78 76 84 67 53]
 [96 66 35 19 64 40 16 95 31 54]
 [10 76 57 46 64 77 82 92 77 99]
 [57  3 24 65 62 39 10 66 80 66]
 [56 69 12 42 95 59 14 54 87 54]
 [90 13 17 59 38 95 23 84 50 20]]
```

5. **Из int в float** Дан массив целочисленных значений (`matrix` из задания 3). Изменить тип массива с `int` на `float`, используя `astype()`

6. **Выше. Ниже. Диагональ** Создать матрицу из нулей 8×8 :

- а) С числами от 1 до 7 **под диагональю**;
- б) С числами от 1 до 7 **над диагональю**.

Полученная матрица должна быть типа `int`

8. **Шахматная доска** Создать матрицу 8×8 и заполнить ее 1 и 0 в шахматном порядке

9. **Сумма и произведение** Дан массив чисел

```
arr = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])
print(arr)
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

а) Необходимо с помощью встроенных функций вывести массив такой, чтобы во втором столбце было **произведение** первых двух чисел в строке, а в третьем столбце — произведение всех трех чисел в строке. В первом столбце сохраняются значения исходного массива;

б) Необходимо с помощью встроенных функций вывести массив такой, чтобы во втором столбце была **сумма** первых двух чисел в строке, а в третьем столбце — сумма всех трех чисел в строке. В первом столбце сохраняются значения исходного массива.