

Методические указания к курсовой работе по дисциплине «Алгоритмизация и программирование»

1.1. Цель и задачи дисциплины

Целью освоения дисциплины является формирование в области информатики и вычислительной техники теоретических знаний и практических навыков для решения прикладных задач, связанных с использованием принципов и моделей эффективного использования современных средств и методов разработки программного обеспечения.

Задачами освоения дисциплины являются:

- овладение основными навыками алгоритмизации и методами построения программ на языках высокого уровня;
- способность разрабатывать модели задач для составления программ, разрабатывать и отлаживать программы, проводить тестирование.

1.2. Руководство курсовой работой

Руководитель курсовой работы:

- оказывает помощь бакалавру в выборе темы курсовой работы;
- составляет задание по курсовой работе;
- помогает бакалавру в подборе списка литературных источников и интернет-ресурсов, необходимых для выполнения курсовой работы;
- проводит консультации бакалавру;
- проверяет выполнение работы.

1.3. Методические рекомендации по выполнению курсовой работы

Курсовая работа является одной из основных форм самостоятельной работы обучающегося, призвана подготовить будущего специалиста к практической работе, повысить уровень его профессиональной подготовки.

Результатом выполнения курсовой работы является:

- 1) готовый программный продукт;
- 2) техническая документация к разработанному программному продукту;
- 3) пояснительная записка, оформленная в соответствии с требованиями.

Оформление пояснительной записки смотрите в документе «Методические рекомендации по оформлению ВКР и курсовых работ (проектов) (<https://bspu.ru/files/106695> - пункт 2).

2. Требования к курсовой работе

По итогам курсовой работы проверяется степень освоения бакалавром следующих компетенций:

- способен разрабатывать алгоритмы и программы, пригодные для практического применения

3. Структура курсовой работы и требования к ее содержанию

Пояснительная записка к курсовой работе должна состоять из следующих разделов:

1. Титульный лист (см. Приложение 1).
2. Содержание.

3. Введение (содержит:
 - актуальность работы, цель курсовой работы, задачи для достижения цели).
4. Обзор методов и инструментов для решения поставленной задачи.
5. Постановка задачи и методы решения:
 - математическая модель задачи (при необходимости, в зависимости от выбранной темы) или содержательная постановка задачи;
 - обзор возможных методов/алгоритмов решения;
 - алгоритм решения и/или информационная схема программного продукта.
6. Программная реализация и тестирование:
 - описание применяемых средств программирования;
 - описание собственной разработки (либо описание используемого программного пакета);
 - проверка работоспособности программы / результаты вычислительного эксперимента или тестирование программы.
7. Заключение:
 - общие выводы по результатам работы;
 - предложения по использованию результатов работы, возможности внедрения разработанного программного продукта.
8. Список использованной литературы.
9. Приложения: листинг программы; скриншоты результатов выполнения, дополнительные справочные материалы (при необходимости), имеющие вспомогательное значение.

Содержание пояснительной записки определяется целями и задачами курсовой работы и делится на главы и параграфы. Между главами должна быть органическая внутренняя связь, материал внутри глав должен излагаться в четкой логической последовательности. Каждая глава заканчивается краткими выводами. Названия глав должны быть предельно краткими, четкими и точно отражать их основное содержание.

Список использованных источников и интернет-ресурсов должен включать не менее 8-10 наименований. На каждый указанный источник должна быть ссылка в тексте курсовой. Нумерация использованных источников должна быть сплошной. Литература на иностранном языке приводится после основного списка библиографии. В конце библиографии размещаются адреса компьютерных источников из Интернета.

Все материалы, помещаемые в приложениях, должны быть связаны с основным текстом, в котором обязательно делаются ссылки на соответствующие приложения.

Общий объем пояснительной записки к курсовой работе (рекомендуемый) – 15 – 30 машинописных страниц.

Выбор темы курсовой работы входит в задание.

Возможные темы курсовых работ:

1. Реализация игровых программ.
2. Разработка программ с использованием пользовательских типов-классов:
 - массив
 - многочлен
 - матрица
 - строка

- дата
 - время
 - список
 - бинарное дерево
 - одно- и двусвязный списки
 - графический объект
3. Программирование численных методов.
 4. Программные решения задач с использованием алгоритмов на графах, таких, как:
 - поиск кратчайших расстояний в графе;
 - нахождение остова минимального веса;
 - решение задачи коммивояжера
 - нахождение критических путей в сетевом графике.
 5. Разработка мобильного приложения.

Возможна как групповая работа (команда 2-3 человека), так и индивидуальная.

Тема курсовой работы определяется бакалавром совместно с руководителем, утверждение темы курсовой работы происходит только после обсуждения и согласования с преподавателем.

После выбора темы работы обучающийся в установленный на кафедре срок должен оформить задание на выполнение курсовой работы и ознакомиться с планом-графиком написания работы.

4. Порядок выполнения и представления курсовой работы

4.1. Составление рабочего плана

Рабочий план курсовой работы разрабатывается обучающимся при участии руководителя.

Первоначальный вариант плана должен отражать основную идею работы. При составлении первоначального варианта плана работы следует определить содержание работы, структуру и модули программного обеспечения.

Рабочий план курсовой работы должен быть гибким. Изменения в плане работы могут быть связаны с некоторой корректировкой направления работы, необходимость в котором может возникнуть после детального ознакомления с изучаемой проблемой.

Все изменения в плане работы должны быть согласованы с руководителем, окончательный вариант плана курсовой работы утверждается руководителем.

4.2. Подбор и ознакомление с литературой и источниками по избранной теме

Подбор литературы и источников следует начинать сразу же после выбора темы курсовой работы.

При подборе литературы следует обращаться к предметно-тематическим каталогам и библиографическим справочникам как библиотеки БГПУ (<http://lib13.bspu.ru/marcweb/index.asp>), в частности, подписные электронные ресурсы (<https://e.lanbook.com/>), так и любой публичной библиотеки г. Уфы, а также использовать систему Internet.

Изучение источников по выбранной теме нужно начинать с общих работ, чтобы получить представление об основных вопросах, к которым примыкает избранная тема, а затем уже вести поиск нового материала. При изучении источников желательно соблюдать следующие рекомендации:

- 1) начинать следует с источников, раскрывающих теоретические аспекты изучаемого вопроса – монографий и журнальных статей, после этого

использовать инструктивные материалы (инструктивные материалы используются только последних изданий);

- 2) при изучении источников необходимо отбирать информацию, которая имеет непосредственное отношение к теме работы; критерием оценки прочитанного является возможность его практического использования в работе;
- 3) изучая источники, тщательно следить за оформлением ссылок, чтобы в дальнейшем было легко ими пользоваться;
- 4) необходимо ориентироваться на последние данные, по соответствующей проблеме опираться на самые авторитетные источники, точно указывать, откуда взяты материалы; при отборе фактов из литературных источников нужно подходить к ним критически.

4. 3. Разработка программы

Разработка программы – ответственный этап выполнения курсовой работы.

Программа (приложение) разрабатывается в соответствии с рабочим планом выполнения курсовой работы и на соответствующих этапах синхронизируется с написанием разделов пояснительной записки.

4.4 Порядок защиты курсовой работы

Защита работы проводится на открытом заседании утвержденной на кафедре комиссии с участием не менее двух преподавателей. Студент (или студенты в случае командной работы) защищает курсовую работу на заседании комиссии по защите курсовых работ (далее – комиссия).

Обязательными элементами процедуры защиты курсовой работы являются:

- 1) выступление автора (авторов) работы;
- 2) презентация;
- 3) демонстрация работы разработанного программного обеспечения.

На заседании комиссии приглашаются все желающие.

После окончания доклада студент (студенты) отвечает на вопросы. При ответе разрешается пользоваться курсовой работой, графическим материалом, презентацией, разработанным программным обеспечением. Студент должен продемонстрировать умение быстро ориентироваться в различных вопросах, касаемых представленной темы, показать научную и общую эрудицию.

Комиссия выставляет оценку за защиту курсовой работы на закрытом заседании. При выставлении оценки комиссия руководствуется критериями оценки курсовой работы.

МИНИСТЕРСТВО ПРОСВЕЩЕНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БАШКИРСКИЙ ГОСУДАРСТВЕННЫЙ ПЕДАГОГИЧЕСКИЙ
УНИВЕРСИТЕТ ИМ. М. АКМУЛЛЫ»

ИНСТИТУТ ФИЗИКИ, МАТЕМАТИКИ, ЦИФРОВЫХ И
НАНОТЕХНОЛОГИЙ

Кафедра информационных
технологий
Направление: **03.03.01**
Прикладные математика и
физика,
направленность (профиль)
Компьютерные технологии и
интеллектуальный анализ данных
Курс I

ФИО
ТЕМА

КУРСОВАЯ РАБОТА ПО ДИСЦИПЛИНЕ
«АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ»

Руководитель:
к.т.н., доцент Л.И. Васильева

(оценка)

(дата, подпись)

Уфа 2024

МИНИСТЕРСТВО ПРОСВЕЩЕНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БАШКИРСКИЙ ГОСУДАРСТВЕННЫЙ ПЕДАГОГИЧЕСКИЙ
УНИВЕРСИТЕТ ИМ. М. АКМУЛЛЫ»

ИНСТИТУТ ФИЗИКИ, МАТЕМАТИКИ, ЦИФРОВЫХ И
НАНОТЕХНОЛОГИЙ

Кафедра информационных
технологий

Направление: **03.03.01**

**Прикладные математика и
физика,**

направленность (профиль)

Математика и физика

Курс I

**ФИО
ТЕМА**

КУРСОВАЯ РАБОТА ПО ДИСЦИПЛИНЕ
«АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ»

Руководитель:
д.ф.-м.н., профессор Картак В.М.

(оценка)

(дата, подпись)

Уфа 2024

И.Х. Хуснуллин

МЕТОДЫ ОБРАБОТКИ ДАННЫХ

Учебное пособие

Уфа 2024

Хуснуллин И.Х. Методы обработки данных: учеб. пособие [Текст]. – Уфа: Изд-во БГПУ, 2024. – 92с.

В учебном пособии рассмотрены современные методы обработки данных. Для работы с данными используется язык программирования Python. Python — один из самых популярных и доступных языков программирования. Его используют для обработки данных, машинного обучения и работы с нейросетями. У Python есть объёмная библиотека с готовыми инструментами для анализа данных. Каждый метод обработки данных реализован в виде блокнота Google Colaboratory или Colab. Colab облачный сервис от Google Research. Это IDE, которая позволяет любому пользователю писать исходный код в редакторе и запускать его из браузера. Ориентирован на задачи машинного обучения, анализ данных, образовательные проекты и т. д. В пособии рассмотрены примеры обработки как простых (учебных) данных так и более сложных (реальных) данных. После каждой темы приведены задания для самостоятельного решения.

Рецензенты:

Е.Г. Кудашева, к.ф.-м.н., доцент (БГПУ им.М.Акумлы);

В.Ф. Вильданова, к.ф.-м.н., доцент (ИМВЦ УФИЦ РАН).

ISBN

© Издательство БГПУ, 2024

© Хуснуллин И.Х., 2024

СОДЕРЖАНИЕ

Введение	4
1. Первичная статистическая обработка данных	10
2. Визуализация данных с помощью библиотеки Matplotlib	20
3. Визуализация данных с помощью библиотеки Seaborn	37
4. Нормализация и стандартизация данных	45
5. Обработка категориальных данных	57
6. Обработка отсутствующих данных	67
7. Поиск аномалий и выбросов	81
Заключение	93
Литература	94

ВВЕДЕНИЕ

Информация и данные являются важными понятиями в информатике и информационных технологиях. **Информация** - это знания, которые мы получаем из различных источников, таких как книги, журналы, телевидение и интернет. **Данные** - это факты и цифры, которые могут быть обработаны компьютером. Например, если вы ищете информацию о погоде, данные могут включать температуру, влажность, давление и скорость ветра.

Объем данных в мире растет с невероятной скоростью. По прогнозам специалистов, к 2025 году объем данных будет увеличиваться на 60% каждый год. Это связано с ростом числа устройств, генерирующих данные, таких как смартфоны, датчики, камеры и другие устройства. Кроме того, развитие технологий, таких как искусственный интеллект и машинное обучение, также способствует увеличению объема данных. На сегодняшний день накоплено огромное количество данных. По оценкам экспертов, объем данных в мире составляет более 40 зеттабайт. Это означает, что каждый день генерируется более 2,5 эксабайт новых данных. 40 зеттабайт - это примерно 40 миллионов терабайт.

Математические методы обработки информации - это методы, которые используются для анализа и обработки данных. Они включают в себя различные виды статистики, математический анализ, теорию вероятностей и другие дисциплины.

Одним из наиболее распространенных методов является регрессионный анализ. Он используется для определения зависимости между двумя переменными и позволяет предсказывать значения одной переменной на основе значений другой. Регрессионный анализ включает в себя несколько методов, которые используются для определения зависимости между переменными. Некоторые из них включают линейную регрессию, множественную регрессию, логистическую регрессию и другие. Каждый из этих методов имеет свои особенности и применяется в зависимости от задачи. Например, линейная регрессия используется для прогнозирования значений на основе линейной зависимости, а множественная регрессия - для определения зависимости между несколькими переменными.

Еще один метод - это кластерный анализ. Он позволяет разделить данные на группы, которые имеют схожие характеристики. Это может быть полезно для классификации данных или для выявления тенденций в данных. Кластерный анализ используется для разделения данных на группы, или кластеры. Существует несколько методов кластерного анализа, включая иерархическую кластеризацию, k-средних и DBSCAN. Иерархическая кластеризация создает дерево кластеров, где каждый

кластер является подклассом предыдущего кластера. К-средних разделяет данные на k кластеров, используя центроиды кластеров в качестве центров.

Кроме того, существуют методы, которые позволяют обрабатывать большие объемы данных, такие как машинное обучение и нейронные сети. Эти методы позволяют автоматизировать процесс обработки данных и делать более точные прогнозы.

В целом, математические методы обработки информации являются важным инструментом для анализа данных и принятия решений на основе этих данных.

Машинное обучение и нейронные сети - это два разных подхода к обработке данных. Машинное обучение использует алгоритмы для обучения на данных, а нейронные сети имитируют работу мозга, используя слои нейронов. Оба подхода используются для обработки больших объемов данных и могут быть использованы для решения различных задач, таких как распознавание образов, классификация, прогнозирование и т.д. Современные методы обработки данных включают машинное обучение, глубокое обучение, обработку естественного языка, компьютерное зрение и анализ данных. Машинное обучение используется для предсказания будущих значений на основе исторических данных, а глубокое обучение позволяет создавать модели, которые могут обучаться на больших объемах данных. Обработка естественного языка используется для анализа текста и ответа на вопросы, а компьютерное зрение - для обработки изображений и видео.

Прогнозирование - это процесс предсказания будущих значений переменной на основе ее прошлых значений. Задачи прогнозирования могут быть решены с помощью различных методов, таких как регрессионный анализ, временные ряды и нейронные сети. Регрессионный анализ используется для предсказания зависимой переменной на основе одной или нескольких независимых переменных. Временные ряды используются для прогнозирования будущих значений на основе прошлых значений с учетом сезонности и тренда.

Задачи классификации - это задачи, в которых необходимо разделить объекты на группы на основе их признаков. Для решения задач классификации используются различные методы, такие как деревья решений, машины опорных векторов и нейронные сети.

Большие данные - это термин, который описывает массивные объемы данных, которые слишком велики или сложны для обработки традиционными методами. Это может включать в себя различные типы данных, такие как текст, аудио, видео, и другие. Большие данные могут быть использованы для различных целей, таких как анализ тенденций,

обнаружение паттернов, прогнозирование поведения и многое другое. Одним из примеров использования больших данных является анализ социальных медиа. Компании могут использовать данные из социальных сетей, чтобы узнать больше о своих клиентах и понять их поведение. Это помогает компаниям улучшить свои услуги и предложения, а также улучшить свои маркетинговые стратегии. Еще одним примером использования больших данных является прогнозирование погоды. Метеорологические службы используют данные с различных источников, таких как спутники, радары и датчики, чтобы предсказать погоду на следующий день или неделю.

Однако, большие данные также могут представлять угрозу для конфиденциальности. Данные могут быть использованы без согласия людей, что может привести к нарушению их прав на приватность. Чтобы предотвратить это, правительства и организации должны работать над улучшением защиты данных. Большие данные обладают рядом свойств, которые делают их уникальными и полезными для анализа. Вот некоторые из них:

- Объем: Большие данные представляют собой огромные объемы информации, которые невозможно обработать традиционными методами.
- Разнообразие: Большие данные могут включать различные типы данных, такие как текстовые, аудио, видео и другие.
- Скорость роста: Объем больших данных растет очень быстро, что требует разработки новых методов обработки и анализа данных.

Методы обработки естественного языка (**NLP**) используются для анализа и понимания текста. Они включают в себя распознавание речи, извлечение информации, классификацию текста и ответы на вопросы. NLP используется в различных приложениях, таких как распознавание речи для голосовых помощников, анализ текста для поисковых систем и ответы на вопросы для чат-ботов. Примеры современных языковых моделей:

— **YaGPT** - это языковая модель, разработанная компанией Яндекс. Она использует алгоритм генерации текста GPT-3, но обучена на русскоязычных текстах. YaGPT может генерировать тексты на русском языке, отвечать на вопросы и выполнять другие задачи, связанные с обработкой естественного языка.

— **GPT-3** (Generative Pretrained Transformer 3) - это мощная языковая модель, разработанная OpenAI. Она использует архитектуру трансформера и обучается на огромном объеме текстовых данных. GPT-3 может генерировать текст, отвечать на вопросы, писать стихи и многое другое. Однако, следует помнить, что GPT-3 не всегда может давать точные

ответы, так как она все еще является моделью машинного обучения и может допускать ошибки.

— **YaGPT 2** - это улучшенная версия YaGPT, разработанная компанией Яндекс. Он использует новую технологию генерации текста, которая позволяет создавать более реалистичные и разнообразные тексты. YaGPT 2 также обладает улучшенными функциями обработки естественного языка и может отвечать на более сложные вопросы.

Анамалия в данных - это отклонение от нормы, которое может быть вызвано различными причинами, такими как ошибки измерения, ошибки обработки данных или аномальные события. Выбросы - это точки данных, которые значительно отличаются от остальных данных. Они могут возникать из-за ошибок измерения, выбросов шума или аномальных событий. Для обнаружения аномалий и выбросов используются различные методы, такие как анализ главных компонент, кластерный анализ, регрессионный анализ и другие. Однако, перед использованием этих методов необходимо провести предварительный анализ данных, чтобы понять, какие аномалии могут быть обнаружены и как они могут быть интерпретированы.

Методы обнаружения аномалий:

- Визуальный анализ: это самый простой способ обнаружения выбросов. Он заключается в том, чтобы просмотреть график данных и найти точки, которые сильно отличаются от остальных.
- Метод Ирвина: этот метод основан на расчете расстояния между точкой и ближайшими соседями. Если расстояние превышает определенное пороговое значение, то точка считается выбросом.
- Метод Граббса: этот метод также основан на расстоянии до ближайшего соседа, но использует стандартизированные значения.
- Анализ главных компонент: этот метод позволяет уменьшить размерность данных и выявить выбросы на основе их влияния на главные компоненты.
- Кластерный анализ: выбросы могут быть обнаружены при помощи кластерного анализа, так как они часто не входят ни в один из кластеров.
- Регрессионный анализ: выбросы можно обнаружить, анализируя остатки регрессионной модели.

Если точка имеет большое отклонение от предсказания, то она может быть выбросом. Обнаружение выбросов необходимо для того, чтобы убрать из анализа данных точки, которые могут исказить результаты. Выбросы

могут возникнуть из-за ошибок в измерении, шума или аномальных событий, и их игнорирование может привести к неправильным выводам. Кроме того, выбросы могут указывать на проблемы в процессе сбора данных или на наличие аномальных значений, которые требуют дополнительного исследования.

Пропущенные данные могут исказить результаты анализа, поэтому их необходимо обрабатывать. Обработка пропущенных данных позволяет получить более точные результаты и улучшить качество моделей. Обработка пропущенных данных может включать несколько методов:

1. Заполнение средним значением: это простой метод, который заключается в замене пропущенного значения средним значением по выборке.
2. Метод k-ближайших соседей: этот метод находит k ближайших соседей для каждого наблюдения и заменяет пропущенное значение средним значением этих соседей.

Категориальные данные - это данные, которые можно разделить на группы или категории. Например, пол человека (мужской, женский), цвет волос (блондин, брюнет, шатен), национальность (русский, украинец, белорус) и т.д. Категориальные данные могут быть представлены в виде номинальных шкал или дискретных переменных.

Примеры категориальных данных:

- Пол: мужской, женский
- Возраст: 0-17 лет, 18-24 лет, 25-34 лет, ...
- Профессия: врач, учитель, инженер, ...

Категориальные данные часто используются в анализе данных, поскольку они позволяют более точно описывать объекты и явления. Однако, обработка категориальных данных может быть сложной задачей, так как требует специальных методов и подходов. Для обработки категориальных данных можно использовать различные методы. Один из самых распространенных методов - это кодирование категорий. В этом методе каждой категории присваивается числовое значение, которое затем используется в анализе данных.

Искусственный интеллект (ИИ) является одной из самых перспективных областей развития технологий. С каждым годом ИИ становится все более распространенным и используется в различных сферах жизни. Одной из главных задач развития ИИ является создание

систем, которые могут обрабатывать и анализировать большие объемы данных. Также перспективным направлением развития ИИ является использование нейросетей. Нейросети позволяют создавать модели, способные обучаться на основе данных и принимать решения на основе этих данных. Нейросети могут использоваться для распознавания образов, классификации данных, прогнозирования и других задач. Кроме того, развитие ИИ может привести к созданию систем, которые смогут заменить человека в некоторых задачах. Например, уже сейчас существуют системы, которые могут выполнять работу юристов, врачей и других специалистов. В будущем такие системы могут стать еще более совершенными и заменить человека во многих сферах деятельности.

Данное пособие подходит для изучения дисциплин «Методы математической обработки данных», «Методы количественного и качественного анализа данных», «Методы математической статистики» для направлений 44.03.05, 44.04.01.

ПЕРВИЧНАЯ СТАТИСТИЧЕСКАЯ ОБРАБОТКА ДАННЫХ

Все методы количественной обработки принято подразделять на первичные и вторичные.

Первичная статистическая обработка нацелена на упорядочивание информации об объекте и предмете изучения. На этой стадии “сырые” сведения группируются по тем или иным критериям, заносятся в сводные таблицы. Первично обработанные данные, представленные в удобной форме, дают исследователю в первом приближении понятие о характере всей совокупности данных в целом: об их однородности - неоднородности, компактности - разбросанности, четкости - размытости и т. д. Эта информация хорошо считывается с наглядных форм представления данных и дает сведения об их распределении.

В ходе применения первичных методов статистической обработки получают показатели, непосредственно связанные с производимыми в исследовании измерениями.

К основным методам первичной статистической обработки относятся: вычисление мер центральной тенденции и мер разброса (изменчивости) данных.

Первичный статистический анализ всей совокупности полученных в исследовании данных дает возможность охарактеризовать ее в предельно сжатом виде и ответить на два главных вопроса: 1) какое значение наиболее характерно для выборки; 2) велик ли разброс данных относительно этого характерного значения, т. е. какова “размытость” данных. Для решения первого вопроса вычисляются меры центральной тенденции, для решения второго - меры изменчивости (или разброса). Эти статистические показатели используются в отношении количественных данных, представленных в порядковой, интервальной или пропорциональной шкале.

Меры центральной тенденции - это величины, вокруг которых группируются остальные данные. Данные величины являются как бы обобщающими всю выборку показателями, что, во-первых, позволяет судить по ним обо всей выборке, а во-вторых, дает возможность сравнивать разные выборки, разные серии между собой. К мерам центральной тенденции в обработке результатов исследований относятся: выборочное среднее, медиана, мода.

Медиана (Me) - это значение, выше и ниже которого количество отличающихся значений одинаково, т. е. это центральное значение в последовательном ряду данных. Медиана не обязательно должна совпадать с конкретным значением. Совпадение происходит в случае нечетного числа значений (ответов), несовпадение - при четном их числе. В последнем случае медиана вычисляется как среднее арифметическое двух центральных значений в упорядоченном ряду.

Мода (Mo) - это значение, наиболее часто встречающееся в выборке, т. е. значение с наибольшей частотой. Если все значения в группе встречаются одинаково часто, то считается, что моды нет. Если два соседних значения имеют одинаковую частоту и больше частоты любого другого значения, мода есть среднее этих двух значений. Если то же самое относится к двум несмежным значениям, то существует две моды, а группа оценок является бимодальной.

Меры разброса (изменчивости) - это статистические показатели, характеризующие различия между отдельными значениями выборки. Они позволяют судить о степени однородности полученного множества, его компактности, а косвенно и о надежности полученных данных и вытекающих из них результатов. Наиболее используемые в психологических исследованиях показатели: среднее отклонение, дисперсия, стандартное отклонение.

Размах (R) - это интервал между максимальным и минимальным значениями признака. Определяется легко и быстро, но чувствителен к случайностям, особенно при малом числе данных.

Дисперсия (D) характеризует отклонения от средней величины в данной выборке.

Выполнение первичной статистической обработки данных в среде Google Colab

Импорт библиотек

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set()
```

```
x = np.array([4, 4, 3, 3, 3.5, 4, 4.5, 3, 3.5, 5, 5, 4, 5, 3.5, 5, 4, 3.5, 4, 3.5, 5, 4, 4, 4, 4, 4])
```

```
print(x)
```

[4. 4. 3. 3. 3.5 4. 4.5 3. 3.5 5. 5. 4. 5. 3.5 5. 4. 3.5 4.
3.5 5. 4. 4. 4. 4. 4.]

Сортировка данных. Построение вариационного ряда

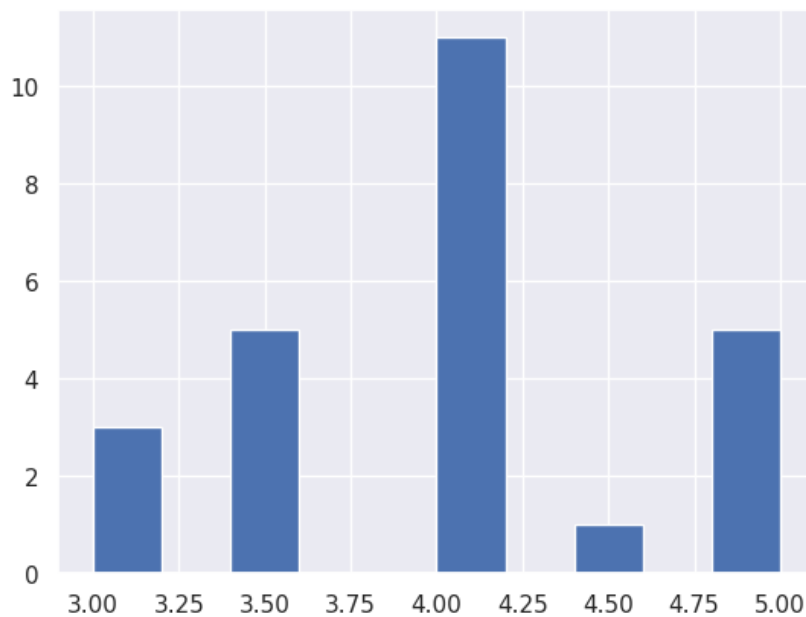
```
x = np.sort(x)
```

```
print(x)
```

[3. 3. 3. 3.5 3.5 3.5 3.5 3.5 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4.5 5. 5. 5. 5. 5.]

Построение гистограммы частот

```
plt.hist(x);
```



Вычисление показателей

```
print('Среднее значение = ', np.mean(x))
```

Среднее значение = 4.0

```
print('max = ', np.max(x))
```

max = 5.0

```

print('min = ', np.min(x))
min = 3.0

print('размах = ', np.max(x) - np.min(x))
размах = 2.0

print('дисперсия = ', np.var(x))
дисперсия = 0.38

print('ср.кв.откл = ', np.std(x))
ср.кв.откл = 0.6164414002968976

print('медиана = ', np.median(x))
медиана = 4.0

```

Квантиль в математической статистике — значение, которое заданная случайная величина не превышает с фиксированной вероятностью. Если вероятность задана в процентах, то квантиль называется процентилем или перцентилем.

- **0,25-квантиль** называется первым (или нижним) квартилем (от лат. quarta — четверть);
- **0,5-квантиль** называется медианой (от лат. mediāna — середина) или вторым квартилем;
- **0,75-квантиль** называется третьим (или верхним) квартилем.

```

print('квантили = ', np.quantile(x, [0.25, 0.5, 0.75]))
квантили = [3.5 4. 4. ]

```

Работа с реальными данными

Чему равен средний рост президентов США

Файл можно скачать по ссылке:

<https://drive.google.com/file/d/18prssmMKvYMclcFXimBdKXFXfWG0reji/view?usp=sharing>

Или по QR коду:



Имеющиеся в библиотеке NumPy функции агрегирования могут очень пригодиться для обобщения набора значений. В качестве простого примера рассмотрим рост всех президентов США. Эти данные доступны в файле `president_heights.csv`, представляющем собой простой разделенный запятыми список меток и значений:

Загрузка файла

```
data = pd.read_csv('/content/sample_data/president_heights (2).csv')
```

Просмотр содержимого файла

```
print(data)
```

	order	name	height(cm)
0	1	George Washington	189
1	2	John Adams	170
2	3	Thomas Jefferson	189
3	4	James Madison	163
4	5	James Monroe	183
5	6	John Quincy Adams	171
6	7	Andrew Jackson	185
7	8	Martin Van Buren	168
8	9	William Henry Harrison	173
9	10	John Tyler	183
10	11	James K. Polk	173
11	12	Zachary Taylor	173
12	13	Millard Fillmore	175
13	14	Franklin Pierce	178
14	15	James Buchanan	183
15	16	Abraham Lincoln	193
16	17	Andrew Johnson	178
17	18	Ulysses S. Grant	173
18	19	Rutherford B. Hayes	174
19	20	James A. Garfield	183
20	21	Chester A. Arthur	183
21	23	Benjamin Harrison	168
22	25	William McKinley	170

23	26	Theodore Roosevelt	178
24	27	William Howard Taft	182
25	28	Woodrow Wilson	180
26	29	Warren G. Harding	183
27	30	Calvin Coolidge	178
28	31	Herbert Hoover	182
29	32	Franklin D. Roosevelt	188
30	33	Harry S. Truman	175
31	34	Dwight D. Eisenhower	179
32	35	John F. Kennedy	183
33	36	Lyndon B. Johnson	193
34	37	Richard Nixon	182
35	38	Gerald Ford	183
36	39	Jimmy Carter	177
37	40	Ronald Reagan	185
38	41	George H. W. Bush	188
39	42	Bill Clinton	188
40	43	George W. Bush	182
41	44	Barack Obama	185

Просмотр информации о файле

data.info

<bound method DataFrame.info of				order	name height(cm)	
0	1	George Washington	189			
1	2	John Adams	170			
2	3	Thomas Jefferson	189			
3	4	James Madison	163			
4	5	James Monroe	183			
5	6	John Quincy Adams	171			
6	7	Andrew Jackson	185			
7	8	Martin Van Buren	168			
8	9	William Henry Harrison	173			
9	10	John Tyler	183			
10	11	James K. Polk	173			
11	12	Zachary Taylor	173			
12	13	Millard Fillmore	175			
13	14	Franklin Pierce	178			
14	15	James Buchanan	183			
15	16	Abraham Lincoln	193			
16	17	Andrew Johnson	178			
17	18	Ulysses S. Grant	173			
18	19	Rutherford B. Hayes	174			
19	20	James A. Garfield	183			

20	21	Chester A. Arthur	183
21	23	Benjamin Harrison	168
22	25	William McKinley	170
23	26	Theodore Roosevelt	178
24	27	William Howard Taft	182
25	28	Woodrow Wilson	180
26	29	Warren G. Harding	183
27	30	Calvin Coolidge	178
28	31	Herbert Hoover	182
29	32	Franklin D. Roosevelt	188
30	33	Harry S. Truman	175
31	34	Dwight D. Eisenhower	179
32	35	John F. Kennedy	183
33	36	Lyndon B. Johnson	193
34	37	Richard Nixon	182
35	38	Gerald Ford	183
36	39	Jimmy Carter	177
37	40	Ronald Reagan	185
38	41	George H. W. Bush	188
39	42	Bill Clinton	188
40	43	George W. Bush	182
41	44	Barack Obama	185

data.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42 entries, 0 to 41
Data columns (total 3 columns):
#   Column      Non-Null Count  Dtype
---  -
0   order       42 non-null    int64
1   name        42 non-null    object
2   height(cm)  42 non-null    int64
dtypes: int64(2), object(1)
memory usage: 1.1+ KB
```

Копирование информации о росте в отдельный файл

```
heights = np.array(data['height(cm)'])
print(heights)
```

```
[189 170 189 163 183 171 185 168 173 183 173 173 175 178 183 193 178 173
 174 183 183 168 170 178 182 180 183 178 182 188 175 179 183 193 182 183
 177 185 188 188 182 185]
```

Копирование имен в отдельный файл

```
names = data['name']  
#Вывод первых 5 имен  
print(names[:5])
```

```
0  George Washington  
1    John Adams  
2  Thomas Jefferson  
3    James Madison  
4    James Monroe  
Name: name, dtype: object
```

Теперь, получив такой массив данных, мы можем вычислить множество сводных статистических показателей:

```
print("Средний рост:      ", heights.mean())  
print("Стандартное отклонение:", heights.std())  
print("Наименьший рост:   ", heights.min())  
print("Наибольший рост:   ", heights.max())
```

```
Средний рост:      179.73809523809524  
Стандартное отклонение: 6.931843442745892  
Наименьший рост:   163  
Наибольший рост:   193
```

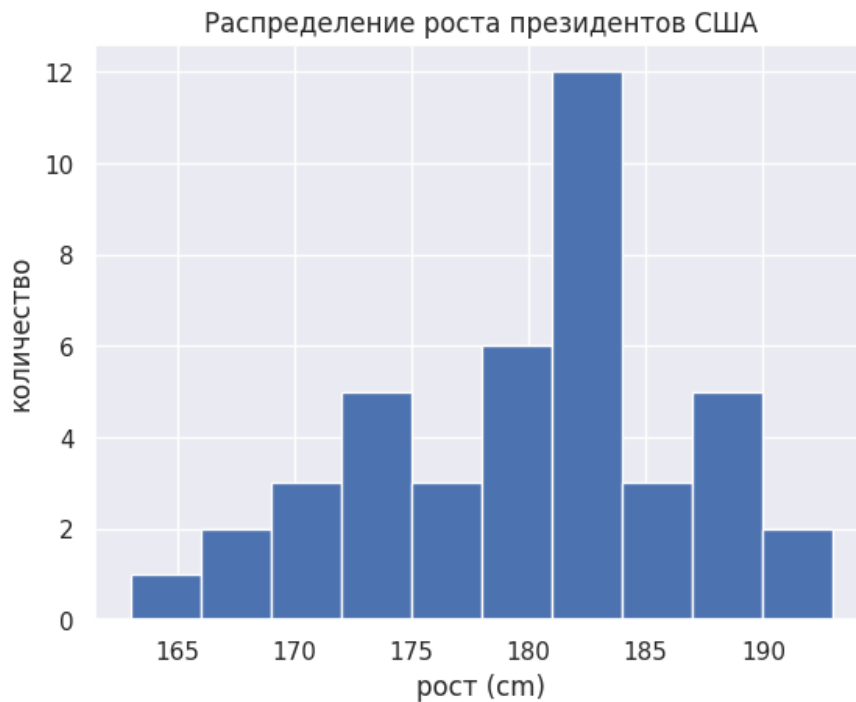
Обратите внимание, что в каждом случае операция агрегирования редуцирует весь массив к одному итоговому значению, дающему нам информацию о распределении значений. Возможно, нам захочется также вычислить квантили:

```
print("25й процентиль:   ", np.percentile(heights, 25))  
print("Медиана или 50й процентиль: ", np.median(heights))  
print("75й процентиль:   ", np.percentile(heights, 75))
```

```
25й процентиль:     174.25  
Медиана или 50й процентиль: 182.0  
75й процентиль:     183.0
```

Иногда полезнее видеть графическое представление подобных данных, которое можно получить с помощью утилит из Matplotlib

```
plt.hist(heights)  
plt.title('Распределение роста президентов США')  
plt.xlabel('рост (см)')  
plt.ylabel('количество');
```



heights

```
array([189, 170, 189, 163, 183, 171, 185, 168, 173, 183, 173, 173, 175,
       178, 183, 193, 178, 173, 174, 183, 183, 168, 170, 178, 182, 180,
       183, 178, 182, 188, 175, 179, 183, 193, 182, 183, 177, 185, 188,
       188, 182, 185])
```

Определим имя самого высокого президента:

```
names[np.argmax(heights)]
```

'Abraham Lincoln'

Определим имя самого низкого президента:

```
names[np.argmin(heights)]
```

'James Madison'

Задания.

- 1) Выполнить все примеры выше в среде Google Colab.
- 2) Выполнить первичную статистическую обработку данных для своего варианта. Номер варианта соответствует порядковому номеру в списке.

Задания можно скачать по ссылке:

https://drive.google.com/file/d/1FQy2FVNlph8dy889Gvf8yVKMNC3XLUoH/view?usp=drive_link

или по QR коду:



Полноценную версию блокнота Google Colab в pdf формате можно посмотреть по QR коду:



ВИЗУАЛИЗАЦИЯ ДАННЫХ С ПОМОЩЬЮ БИБЛИОТЕКИ MATPLOTLIB

Matplotlib — мультиплатформенная библиотека для визуализации данных, основанная на массивах библиотеки NumPy и спроектированная в расчете на работу с обширным стеком SciPy. Она была задумана Джоном Хантером в 2002 году и изначально представляла собой патч к оболочке IPython, предназначенный для реализации возможности интерактивного построения с помощью утилиты gnuplot графиков в стиле MATLAB из командной строки IPython. Создатель оболочки IPython Фернандо Перес в этот момент был занят завершением написания диссертации, он сообщил Джону, что в ближайшие несколько месяцев у него не будет времени на анализ патча. Хантер принял это как благословение на самостоятельную разработку — так родился пакет Matplotlib, версия 0.1 которого была выпущена в 2003 году. Институт исследований космоса с помощью космического телескопа (Space Telescope Science Institute, занимающийся управлением телескопом «Хаббл») финансово поддержал разработку пакета Matplotlib и обеспечил расширение его возможностей, избрав в качестве пакета для формирования графических изображений.

Простые линейные графики

Вероятно, простейшим из всех графиков является визуализация отдельной функции $y = f(x)$. В этом разделе мы рассмотрим создание простого графика такого типа. Как и во всех следующих разделах, начнем с настройки блокнота для построения графиков и импорта функций, которые будем использовать:

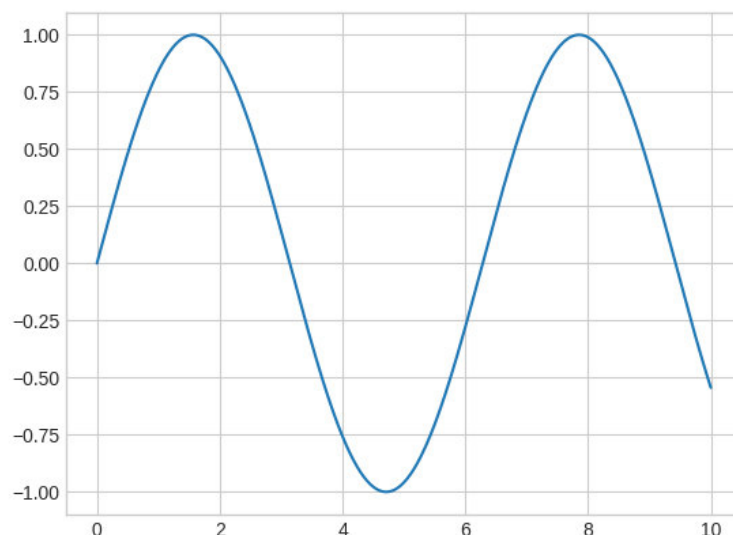
```
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import numpy as np
```

```
plt.style.use('seaborn-whitegrid')
```

Начнем с простой синусоиды

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

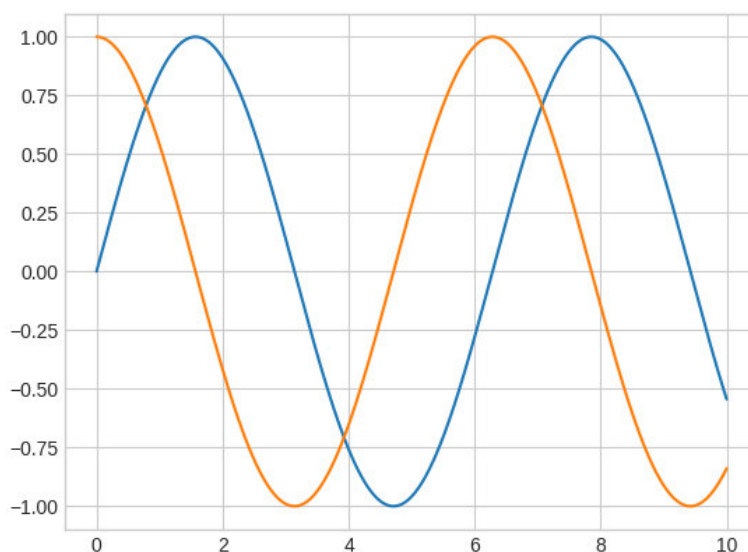
```
x = np.linspace(0, 10, 1000)
ax.plot(x, np.sin(x));
```



Если нужно создать простой рисунок с несколькими линиями, можно вызвать функцию `plot` несколько раз

```
plt.plot(x, np.sin(x))  
plt.plot(x, np.cos(x))
```

[<matplotlib.lines.Line2D at 0x7855ac5e8ee0>]

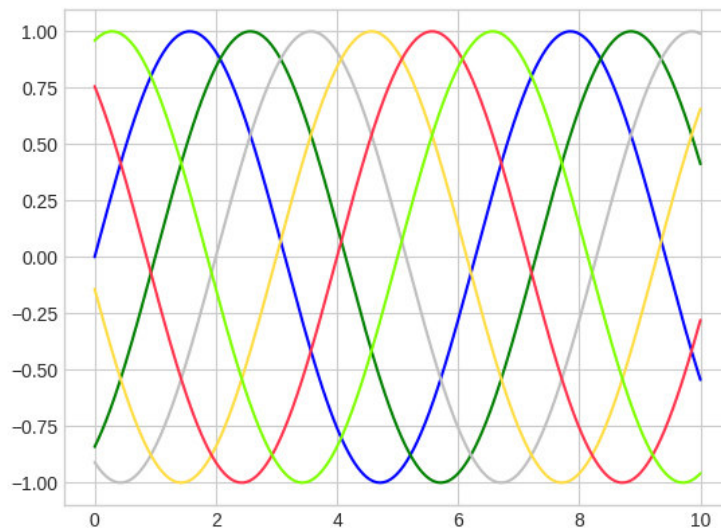


Настройка графика: цвета и стили линий

Первое, что вы можете захотеть сделать с графиком, — научиться управлять цветами и стилями линий. Функция `plt.plot` принимает дополнительные аргументы, которыми можно воспользоваться для этой цели. Для настройки цвета используйте ключевое слово `color`, которому ставится в соответствие строковый аргумент, задающий практически любой цвет. Задать цвет можно разными способами

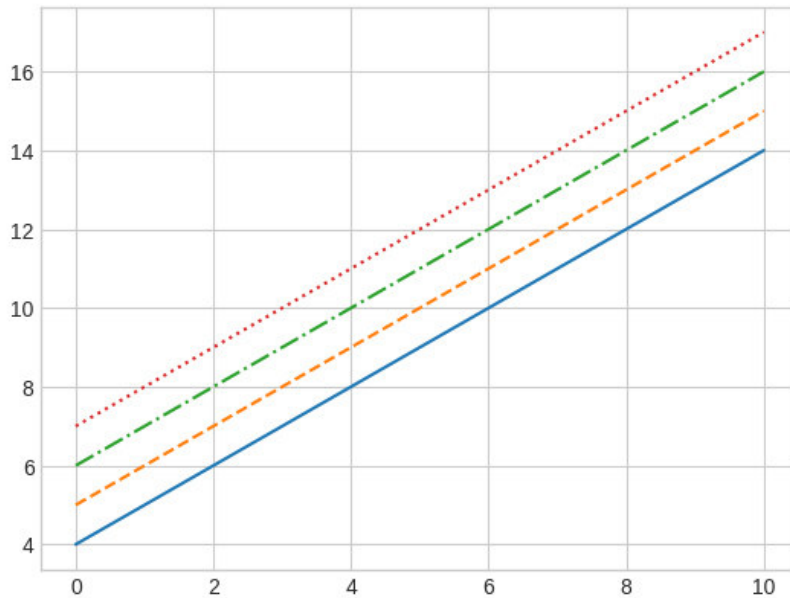
```
plt.plot(x, np.sin(x - 0), color='blue')
plt.plot(x, np.sin(x - 1), color='g')
plt.plot(x, np.sin(x - 2), color='0.75')
plt.plot(x, np.sin(x - 3), color='#FFDD44')
plt.plot(x, np.sin(x - 4), color=(1.0,0.2,0.3))
plt.plot(x, np.sin(x - 5), color='chartreuse')
```

[<matplotlib.lines.Line2D at 0x7855ac1796c0>]



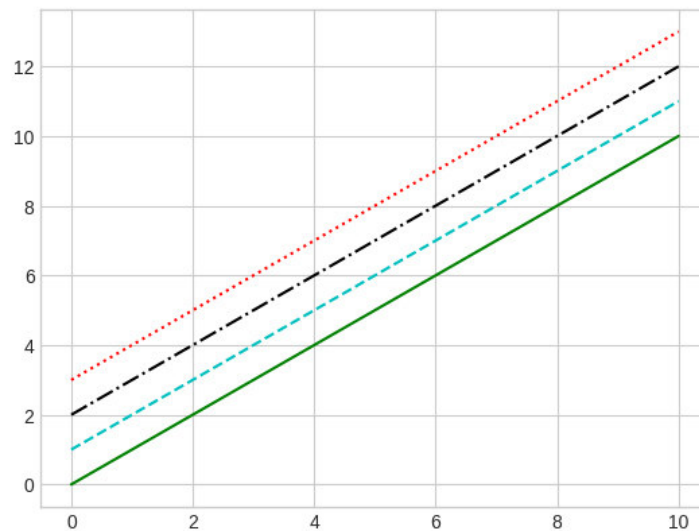
Если цвет не задан, библиотека Matplotlib будет автоматически перебирать по циклу набор цветов по умолчанию при наличии на графике нескольких линий. Стиль линий можно настраивать и с помощью ключевого слова `linestyle`

```
plt.plot(x, x + 4, linestyle='-') # solid
plt.plot(x, x + 5, linestyle='--') # dashed
plt.plot(x, x + 6, linestyle='-.') # dashdot
plt.plot(x, x + 7, linestyle=':'); # dotted
```



Если вы предпочитаете максимально сжатый синтаксис, можно объединить задание кодов `linestyle` и `color` в одном неключевом аргументе функции `plt.plot`

```
plt.plot(x, x + 0, '-g') # solid green
plt.plot(x, x + 1, '--c') # dashed cyan
plt.plot(x, x + 2, '-.k') # dashdot black
plt.plot(x, x + 3, ':r') # dotted red
```

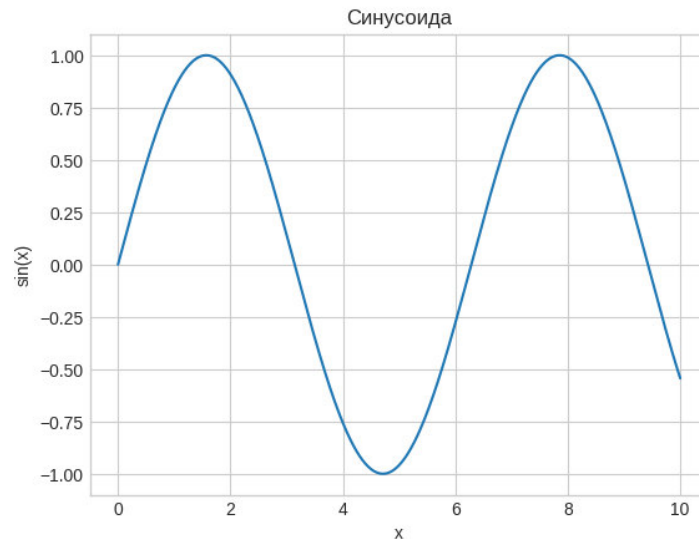


Метки на графиках

Рассмотрим маркирование графиков: названия, метки осей координат и простые легенды. Названия и метки осей — простейшие из подобных меток. Существуют методы, позволяющие быстро задать их значения

```
plt.plot(x, np.sin(x))
plt.title("Синусоида")
plt.xlabel("x")
plt.ylabel("sin(x)")
```

```
Text(0, 0.5, 'sin(x)')
```

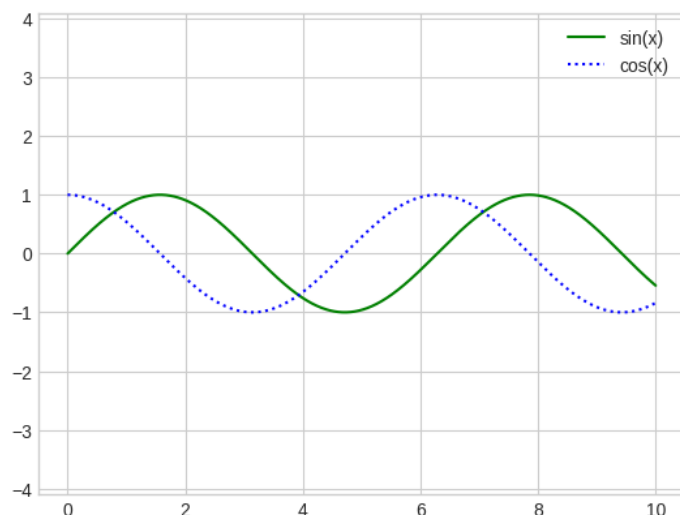


В случае отображения нескольких линий в одной координатной сетке удобно создать легенду для графика, на которой бы отмечался каждый тип линии. В библиотеке Matplotlib для быстрого создания такой легенды имеется встроенный метод `plt.legend()`. Хотя существует несколько возможных способов, проще всего, задать метку каждой линии с помощью ключевого слова `label` функции `plot`

```
plt.plot(x, np.sin(x), '-g', label='sin(x)')
plt.plot(x, np.cos(x), ':b', label='cos(x)')
plt.axis('equal')
```

```
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7855ac1c6170>
```



Простые диаграммы рассеяния

Еще один часто используемый тип графиков — диаграммы рассеяния, родственные линейным графикам. В них точки не соединяются отрезками линий, а представлены по отдельности точками, кругами или другими фигурами на графике.

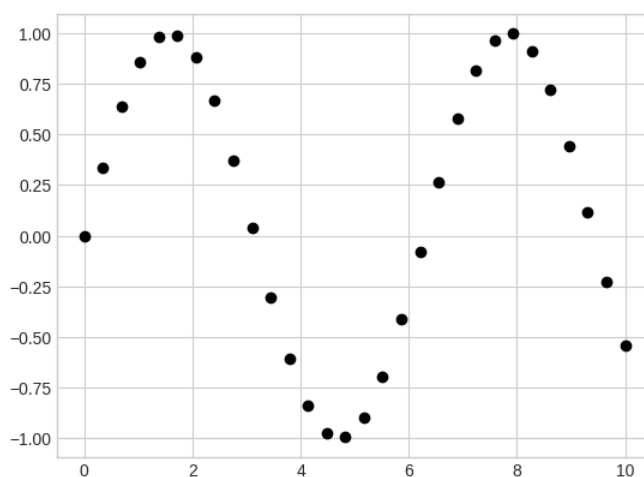
Построение диаграмм рассеяния с помощью функции `plt.plot`

В предыдущем разделе мы рассмотрели построение линейных графиков посредством функции `plt.plot/ax.plot`. С ее помощью можно строить и диаграммы рассеяния

```
x = np.linspace(0, 10, 30)
y = np.sin(x)
```

```
plt.plot(x, y, 'o', color='black')
```

```
[<matplotlib.lines.Line2D at 0x7855907c9990>]
```

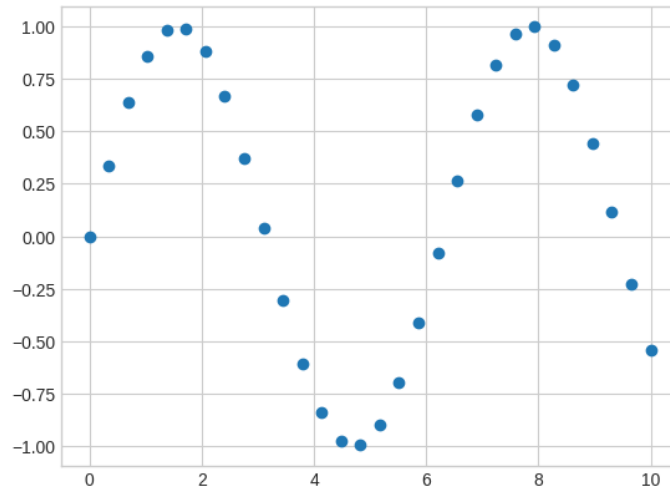


Построение диаграмм рассеяния с помощью функции `plt.scatter`

Еще большими возможностями обладает метод построения диаграмм рассеяния с помощью функции `plt.scatter`, во многом напоминающей функцию `plt.plot`

```
plt.scatter(x, y, marker='o')
```

```
<matplotlib.collections.PathCollection at 0x78559065cd00>
```



Продemonстрируем это, создав случайную диаграмму рассеяния с точками различных цветов и размеров. Чтобы лучше видеть перекрывающиеся результаты, воспользуемся ключевым словом `alpha` для настройки уровня прозрачности

```
rng = np.random.RandomState(0)
```

```
x = rng.randn(100)
```

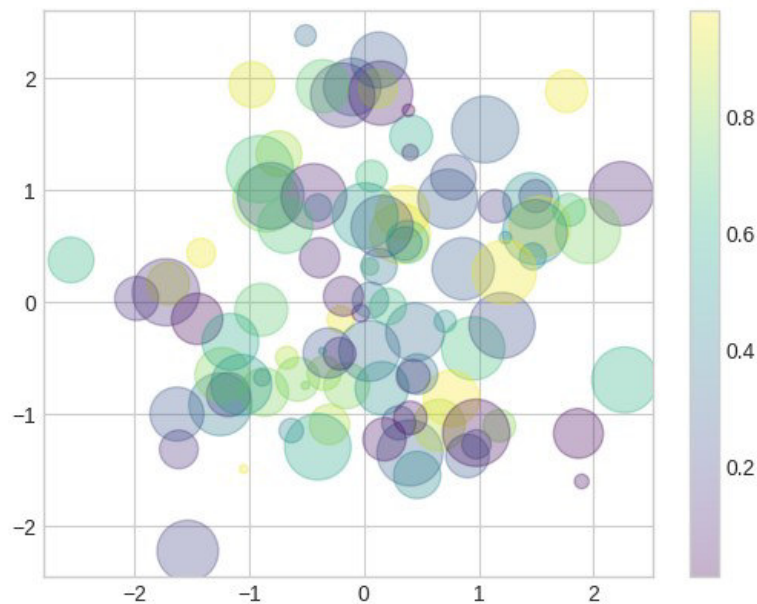
```
y = rng.randn(100)
```

```
colors = rng.rand(100)
```

```
sizes = 1000 * rng.rand(100)
```

```
plt.scatter(x, y, c=colors, s=sizes, alpha=0.3, cmap='viridis')
```

```
plt.colorbar();
```



Визуализация погрешностей.

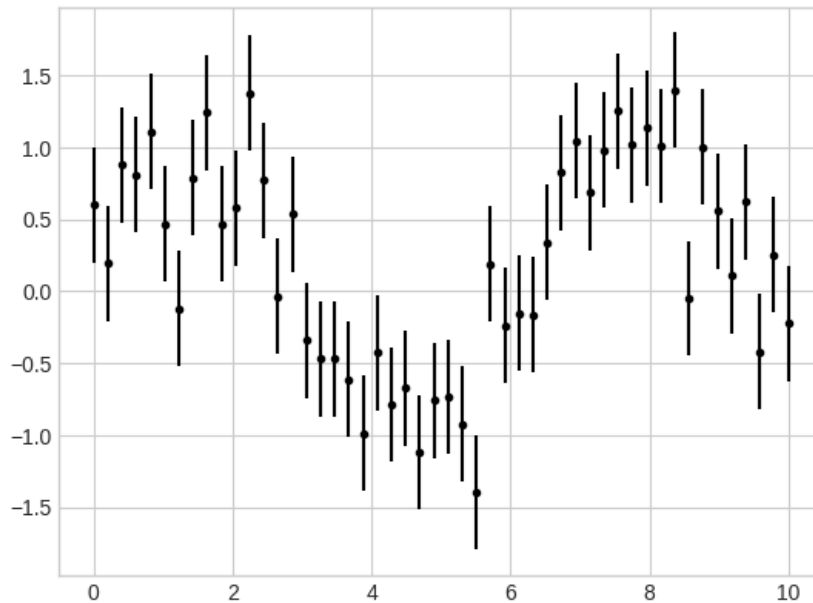
Простые планки погрешностей.

Точный учет погрешностей, как и точная информация о самих значениях важен для любых научных измерений. Например, представьте, что я использую некоторые астрофизические наблюдения для оценки постоянной Хаббла, то есть измеряю скорость расширения Вселенной в данной точке. Мне известно, что в современных источниках по этому вопросу указывается значение около 71 (км/с)/Мпк, а я с помощью моего метода получил значение 74 (км/с)/Мпк. Не противоречат ли значения друг другу? По вышеприведенной информации ответить на этот вопрос невозможно.

```
x = np.linspace(0, 10, 50)
dy = 0.4
y = np.sin(x) + dy * np.random.randn(50)

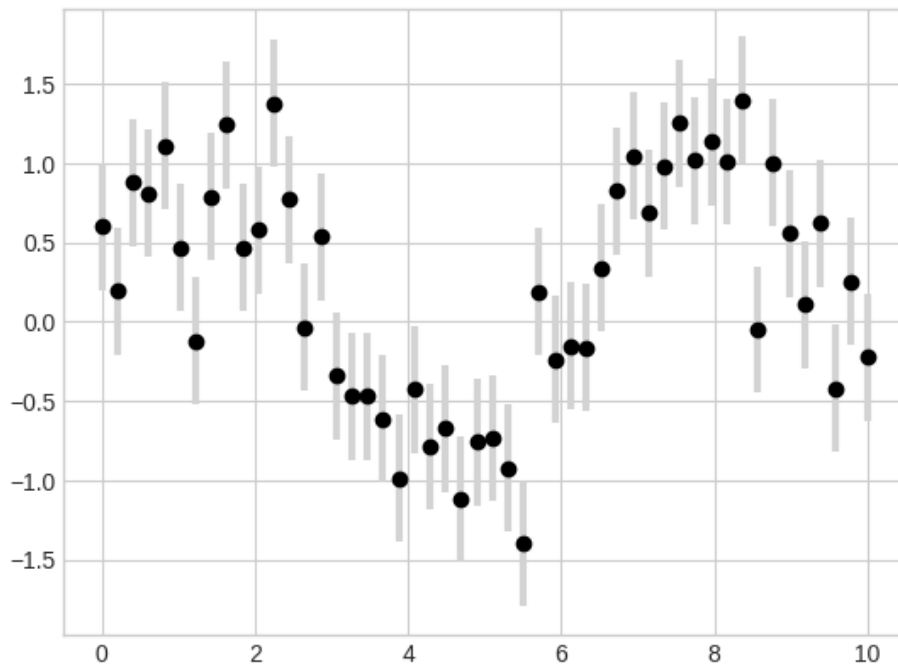
plt.errorbar(x, y, yerr=dy, fmt='.k')

<ErrorbarContainer object of 3 artists>
```



Здесь `fmt` — код форматирования, управляющий внешним видом линий и точек, его синтаксис совпадает с сокращенным синтаксисом, используемым в функции `plt.plot`, описываемой в разделах «Простые линейные графики» и «Простые диаграммы рассеяния» данной главы. Помимо этих простейших, у функции `errorbar` есть множество параметров для более тонкой настройки выводимых данных. С помощью этих дополнительных параметров вы можете легко настроить в соответствии со своими требованиями внешний вид графика планок погрешностей. Планки погрешностей удобно делать более светлыми, чем точки, особенно на насыщенных графиках.

```
plt.errorbar(x, y, yerr=dy, fmt='o', color='black',
             ecolor='lightgray', elinewidth=3, capsize=0);
```



Графики плотности и контурные графики.

Иногда удобно отображать трехмерные данные в двумерной плоскости с помощью контуров или областей, кодированных различными цветами. Существует три предназначенные для этой цели функции библиотеки Matplotlib:

- `plt.contour` — для контурных графиков;
- `plt.contourf` — для контурных графиков с заполнением;
- `plt.imshow` — для отображения изображений. В этом разделе мы рассмотрим несколько примеров использования данных функций.

Визуализация трехмерной функции.

Начнем с демонстрации контурного графика для функции вида $z = f(x, y)$, воспользовавшись для этой цели функцией `f`

```
def f(x, y):
    return np.sin(x) ** 10 + np.cos(10 + y * x) * np.cos(x)
```

Создать контурный график можно с помощью функции `plt.contour`. У нее имеется три аргумента: координатная сетка значений x , координатная сетка значений y и координатная сетка значений z . Значения x и y представлены точками на графике, а значения z будут представлены контурами уровней. Вероятно, наиболее простой способ подготовить такие данные — воспользоваться функцией `np.meshgrid`, формирующей двумерные координатные сетки из одномерных массивов:

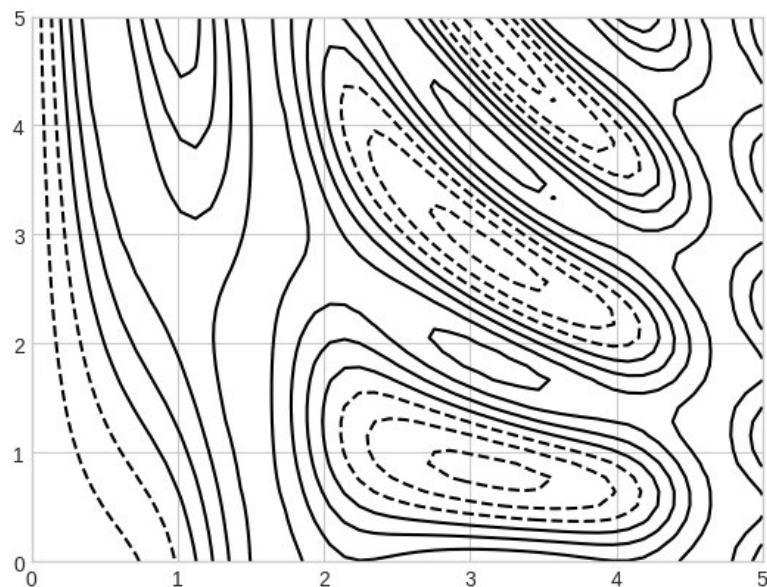
```
x = np.linspace(0, 5, 50)
y = np.linspace(0, 5, 40)
```

```
X, Y = np.meshgrid(x, y)
Z = f(X, Y)
```

Теперь посмотрим, как это отображается на обычном линейном контурном графике

```
plt.contour(X, Y, Z, colors='black')
```

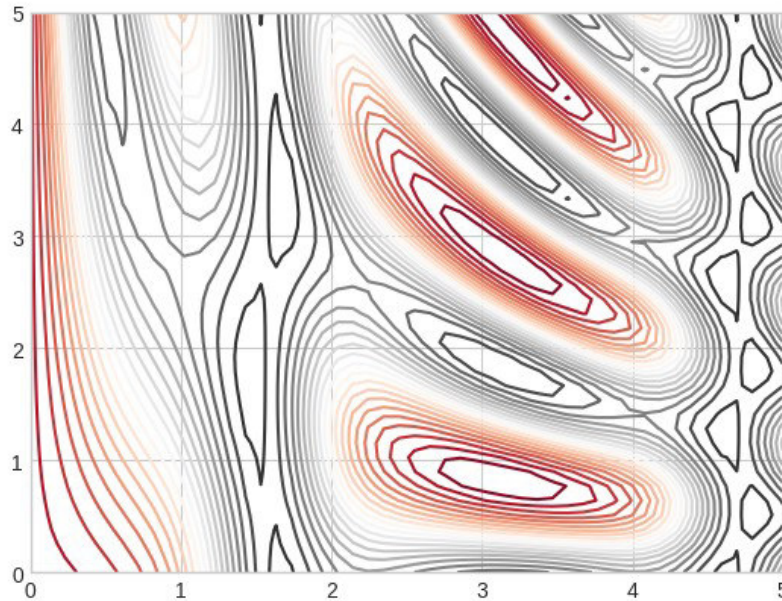
```
<matplotlib.contour.QuadContourSet at 0x785590453730>
```



Обратите внимание, что по умолчанию при использовании одного цвета отрицательные значения обозначаются штриховыми линиями, а положительные — сплошными. Можно также кодировать линии различными цветами, задав карты цветов с помощью аргумента `cmap`.

```
plt.contour(X, Y, Z, 20, cmap='RdGy')
```

```
<matplotlib.contour.QuadContourSet at 0x7855904f7e80>
```

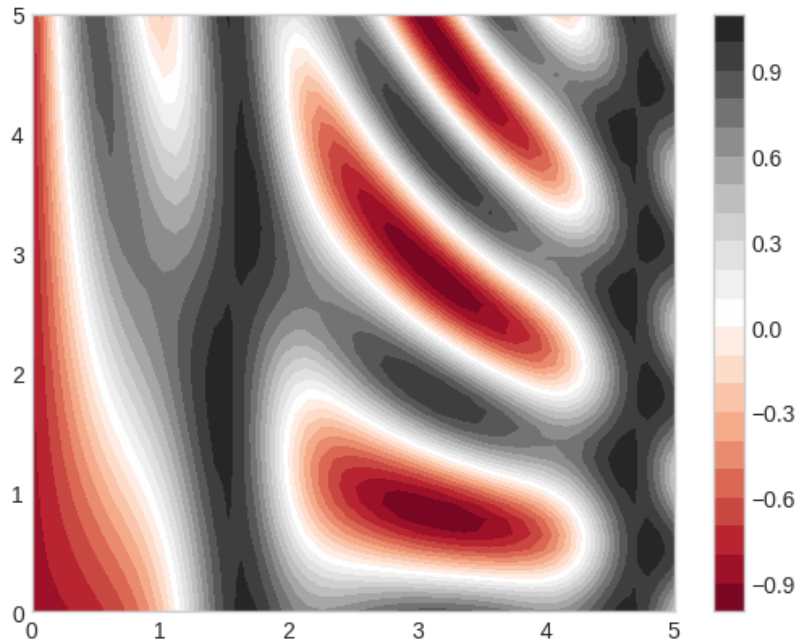


Наш график приобрел более приятный глазу вид, но промежутки между линиями несколько отвлекают внимание. Изменить это можно, воспользовавшись контурным графиком с заполнением, доступным посредством функции `plt.contourf()` (обратите внимание на букву `f` в конце ее названия), синтаксис которой не отличается от синтаксиса функции `plt.contour()`.

Вдобавок мы воспользуемся командой `plt.colorbar()`, автоматически создающей для графика дополнительную ось с маркированной информацией о цвете:

```
plt.contourf(X, Y, Z, 20, cmap='RdGy')
plt.colorbar()
```

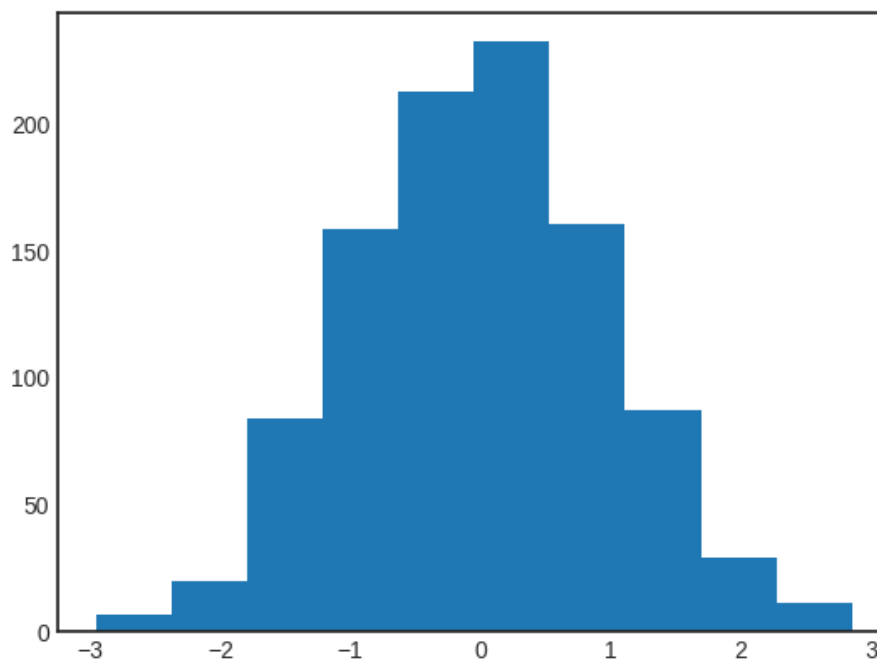
```
<matplotlib.colorbar.Colorbar at 0x78559021b9a0>
```



Гистограммы, разбиения по интервалам и плотность.

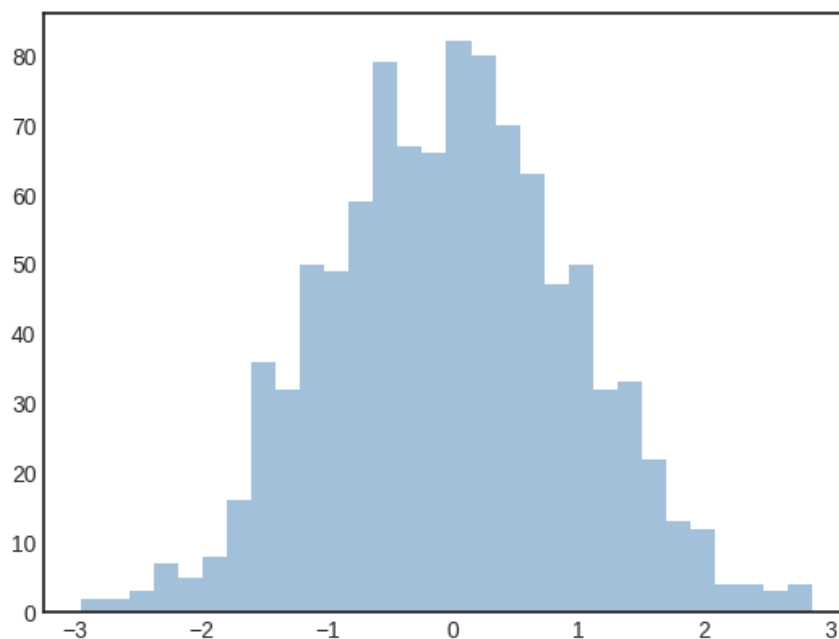
Простая гистограмма может принести огромную пользу при первичном анализе набора данных. Ранее мы видели пример использования функции библиотеки Matplotlib для создания простой гистограммы в одну строку после выполнения всех обычных импортов

```
plt.style.use('seaborn-white')
data = np.random.randn(1000)
plt.hist(data);
plt.style.use('seaborn-white')
```



У функции `hist()` имеется множество параметров для настройки как вычисления, так и отображения. Вот пример гистограммы с детальными пользовательскими настройками

```
plt.hist(data, bins=30, alpha=0.5,  
         histtype='stepfilled', color='steelblue',  
         edgecolor='none');
```



Двумерные гистограммы и разбиения по интервалам

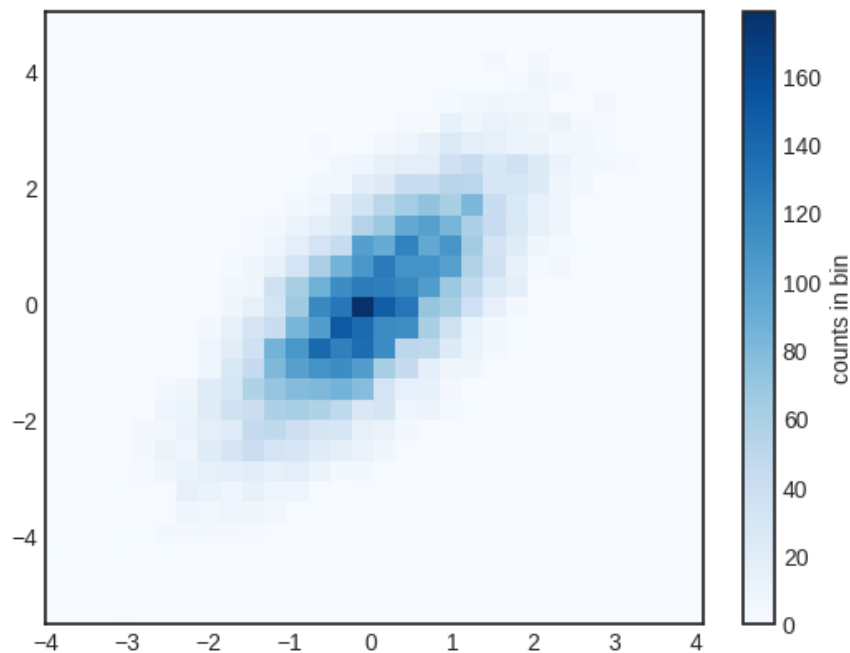
Аналогично тому, как мы создавали одномерные гистограммы, разбивая последовательность чисел по интервалам, можно создавать и двумерные гистограммы, распределяя точки по двумерным интервалам. Рассмотрим несколько способов выполнения. Начнем с описания данных массивов `x` и `y`, полученных из многомерного Гауссова распределения:

```
mean = [0, 0]  
cov = [[1, 1], [1, 2]]  
x, y = np.random.multivariate_normal(mean, cov, 10000).T
```

Функция `plt.hist2d`: двумерная гистограмма

Один из простых способов нарисовать двумерную гистограмму — воспользоваться функцией `plt.hist2d` библиотеки `Matplotlib`

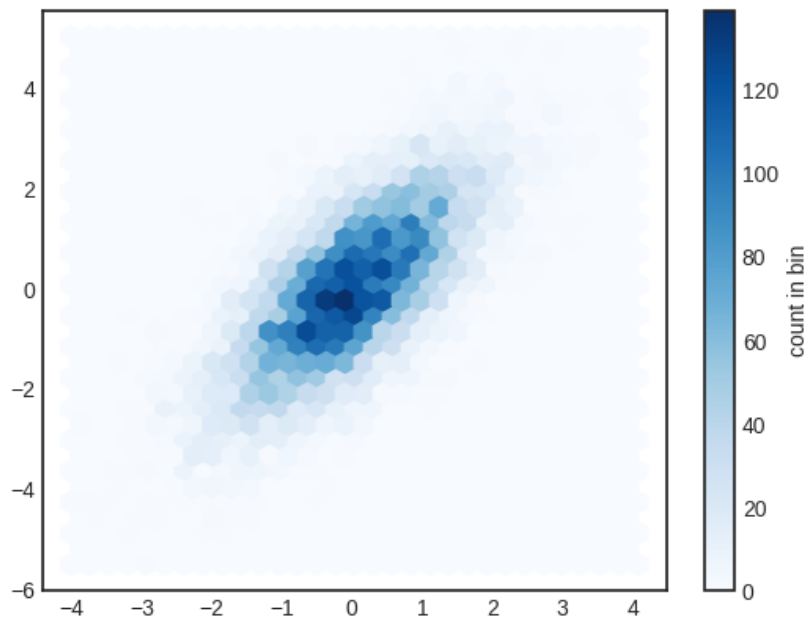
```
plt.hist2d(x, y, bins=30, cmap='Blues')
cb = plt.colorbar()
cb.set_label('counts in bin')
```



Функция `plt.hexbin`: гексагональное разбиение по интервалам

Двумерная гистограмма создает мозаичное представление квадратами вдоль координатных осей. Другая геометрическая фигура для подобного мозаичного представления — правильный шестиугольник. Для этих целей библиотека Matplotlib предоставляет функцию `plt.hexbin` — двумерный набор данных, разбитых по интервалам на сетке из шестиугольников

```
plt.hexbin(x, y, gridsize=30, cmap='Blues')
cb = plt.colorbar(label='count in bin')
```



Задания

- 1) Выполнить все примеры выше в среде Google Colab.
- 2) Построить гистограмму для по данным своего варианта. Номер варианта соответствует порядковому номеру в списке. Данные можно скачать по ссылке: https://drive.google.com/file/d/1FQy2FVNlph8dy889Gvf8yVKMNC3XLUoH/view?usp=drive_link

или по QR коду:



- 3) Построить график функции в указанном промежутке. Цвет линии – красный, тип линии – сплошной. Ось абсцисс подписать “x”, ось ординат подписать “y”. Задать метку построенной линии. Задания можно скачать по ссылке: <https://drive.google.com/file/d/1yb6rjIYma9c7v09GihhTIrnMEkQJP7AT/view?usp=sharing>

или по QR коду:



Полноценную версию блокнота Google Colab в pdf формате можно посмотреть по QR коду:



ВИЗУАЛИЗАЦИЯ ДАННЫХ С ПОМОЩЬЮ БИБЛИОТЕКИ SEABORN.

Библиотека Matplotlib зарекомендовала себя как невероятно удобный и популярный инструмент визуализации, но даже заядлые ее пользователи признают, что она зачастую оставляет желать лучшего.

Библиотека Seaborn — решение этих проблем. Seaborn предоставляет API поверх библиотеки Matplotlib, обеспечивающий разумные варианты стилей графиков и цветов по умолчанию, определяющий простые высокоуровневые функции для часто встречающихся типов графиков и хорошо интегрирующийся с функциональностью, предоставляемой объектами DataFrame библиотеки Pandas.

По принятым соглашениям Seaborn импортируется под именем sns:

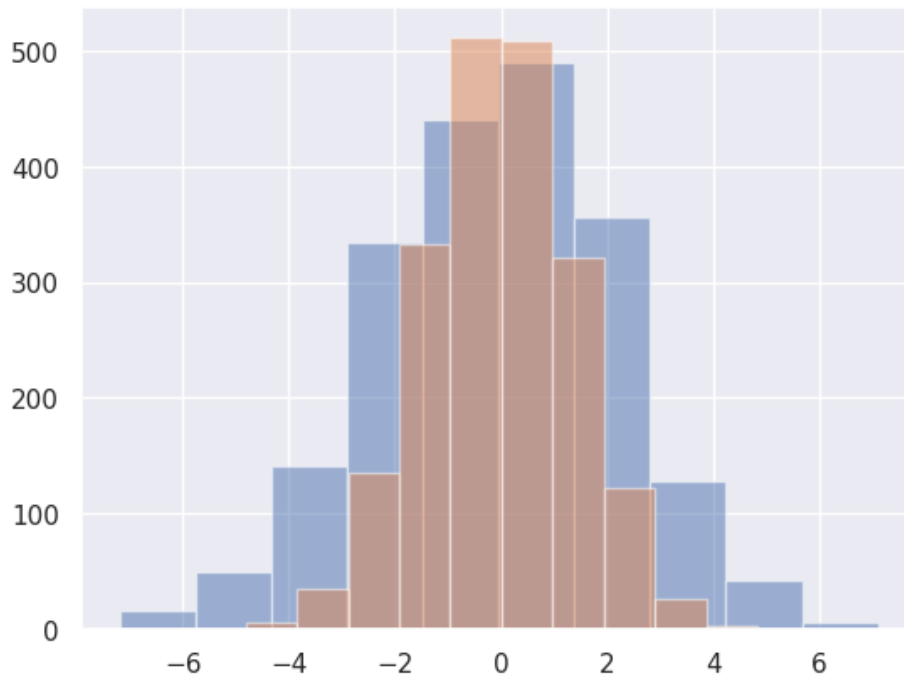
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
```

Гистограммы, KDE и плотности.

Зачастую все, что нужно сделать при визуализации статистических данных, — это построить гистограмму и график совместного распределения переменных. Мы уже видели, что в библиотеке Matplotlib сделать это относительно несложно:

```
import pandas as pd
data = np.random.multivariate_normal([0, 0], [[5, 2], [2, 2]], size=2000)
data = pd.DataFrame(data, columns=['x', 'y'])

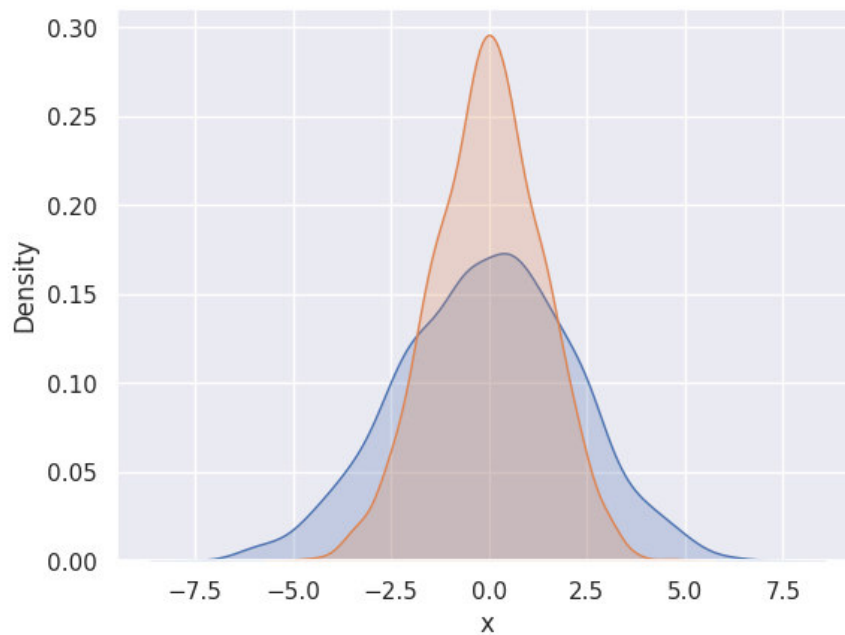
for col in 'xy':
    plt.hist(data[col], alpha=0.5)
```



Вместо гистограммы можно получить гладкую оценку распределения путем ядерной оценки плотности распределения, которую Seaborn выполняет с помощью функции `sns.kdeplot`

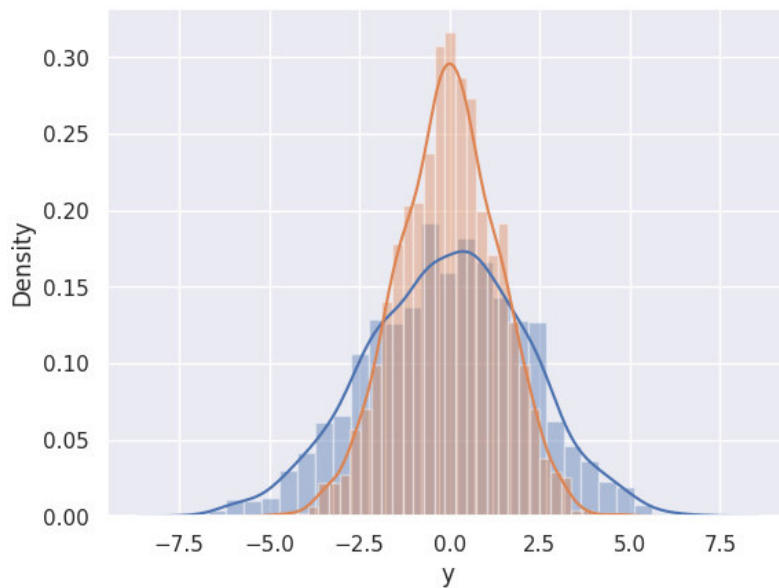
```
for col in 'xy':
    sns.kdeplot(data[col], shade=True)

sns.kdeplot(data[col], shade=True)
```



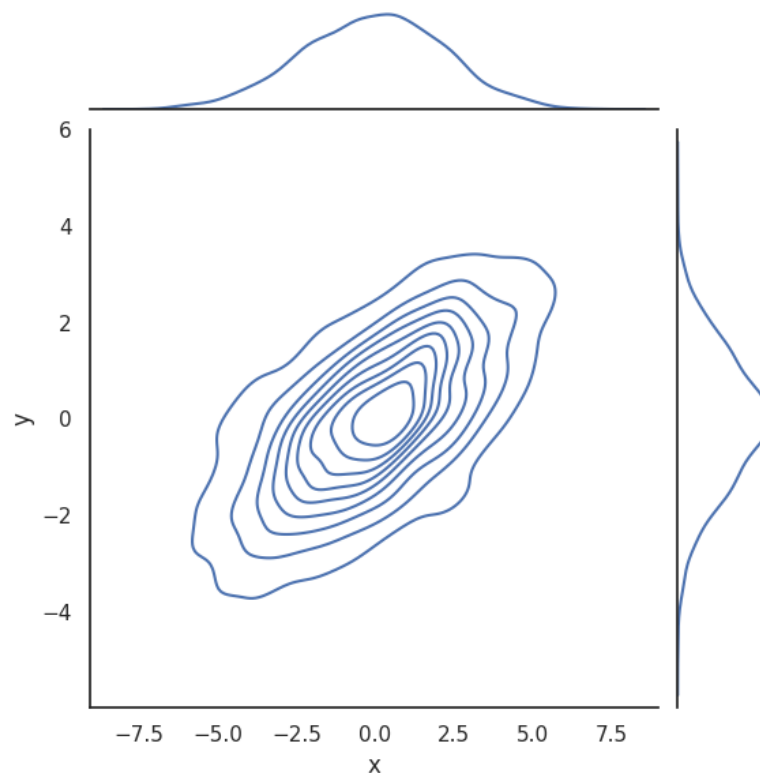
С помощью функции `distplot` можно сочетать гистограммы и KDE

```
sns.distplot(data['x'])
sns.distplot(data['y']);
```



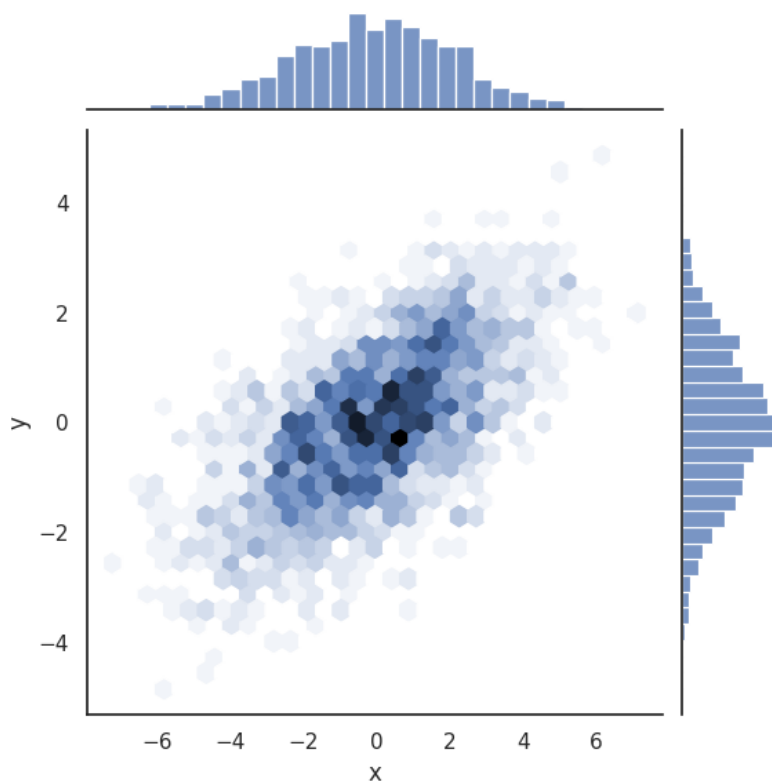
Посмотреть на совместное распределение и частные распределения можно, воспользовавшись функцией `sns.jointplot`.

with `sns.axes_style('white')`:
`sns.jointplot(data = data, x = "x", y = "y", kind='kde');`



Функции `jointplot` можно передавать и другие параметры, например можно воспользоваться гистограммой на базе шестиугольников

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

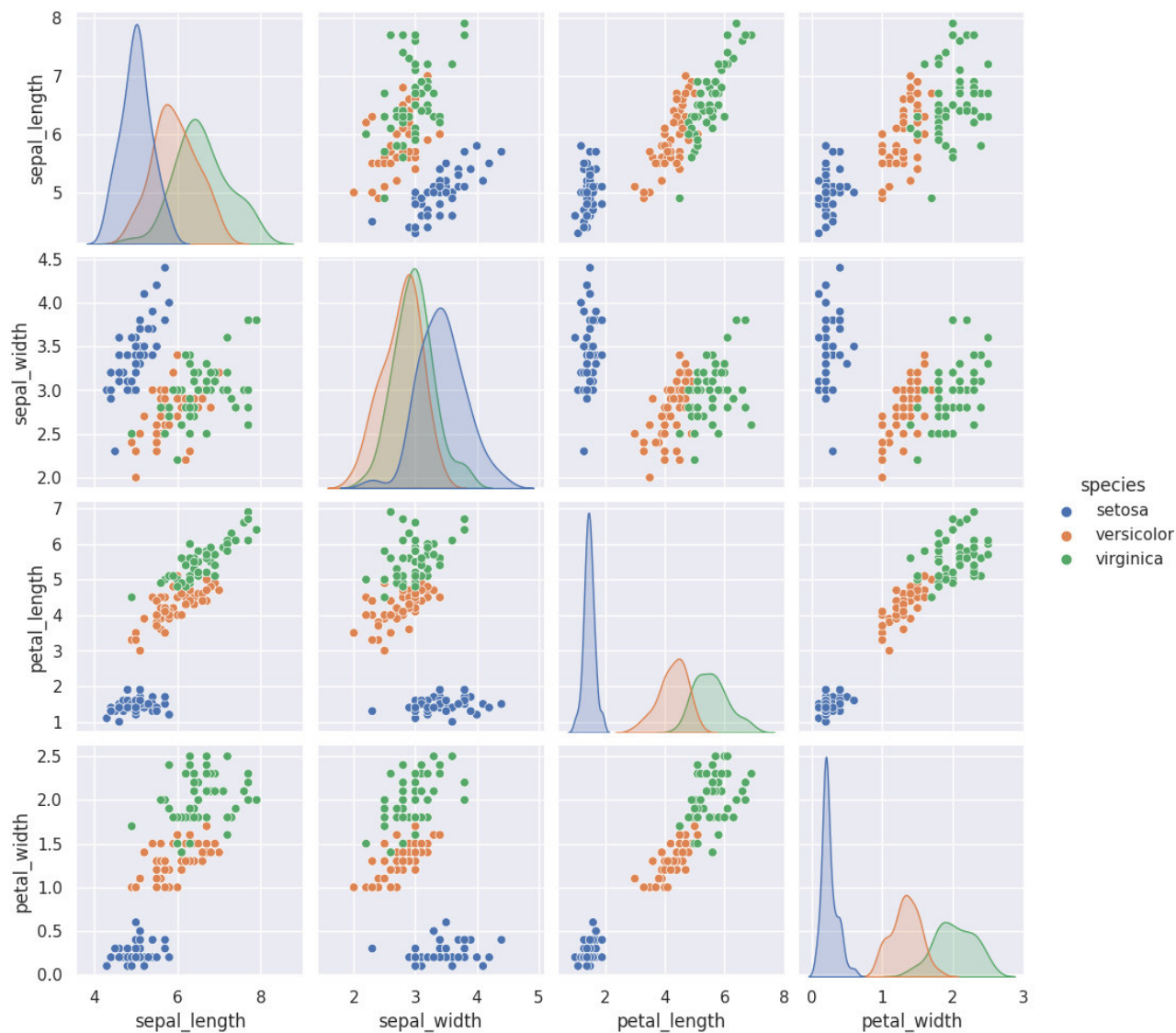


При обобщении графиков совместных распределений на наборы данных более высоких размерностей мы постепенно приходим к графикам пар (pair plots). Они очень удобны для изучения зависимостей между многомерными данными, когда необходимо построить график всех пар значений. Продемонстрируем это на наборе данных Iris, содержащем измерения лепестков и чашелистиков трех видов ирисов:

```
iris = sns.load_dataset("iris")  
iris.head()
```

Визуализация многомерных зависимостей между выборками сводится к вызову функции `sns.pairplot`:

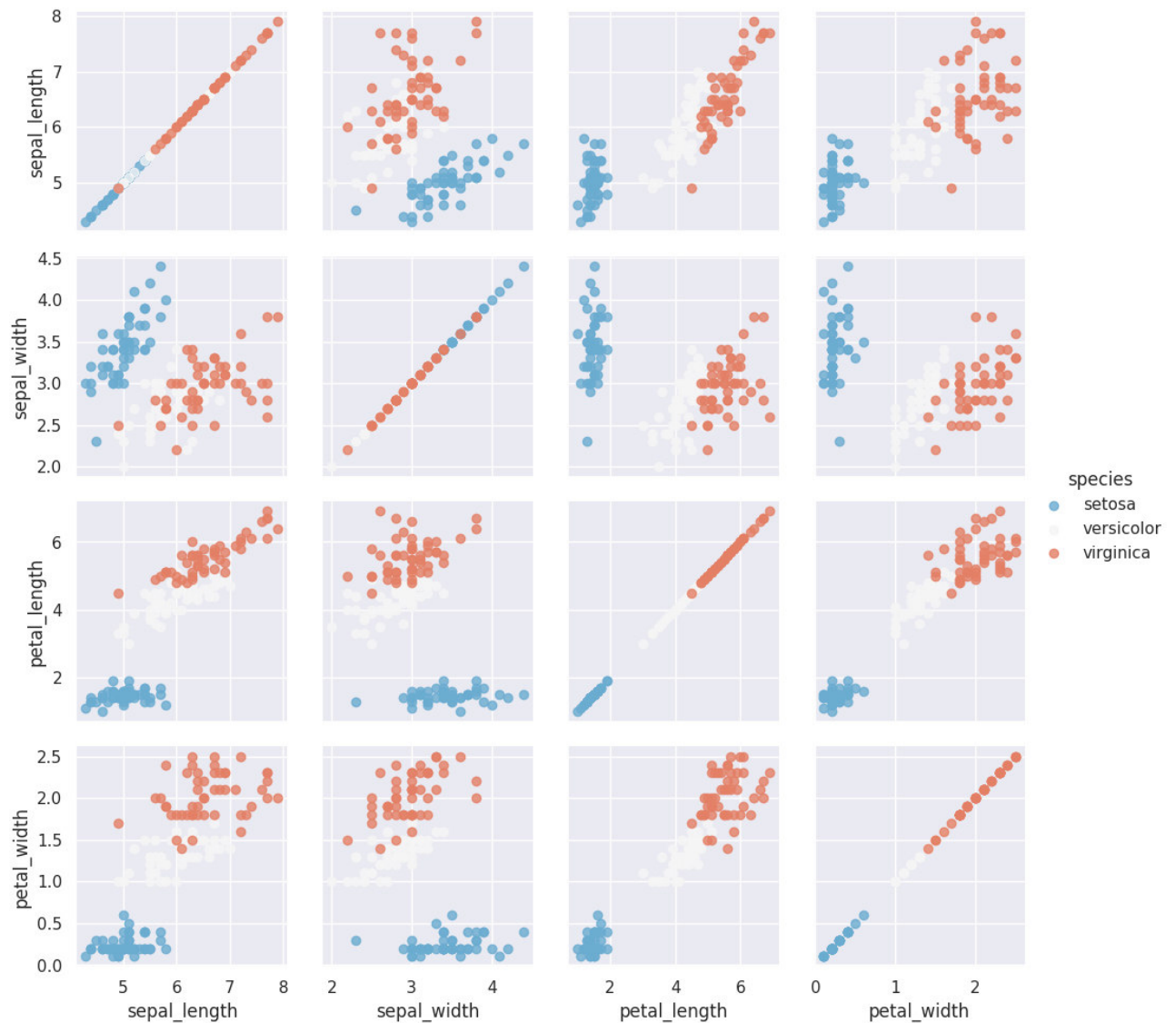
```
sns.pairplot(iris, hue='species', size=2.5);
```



Выясним, существует ли какая-либо корреляция между переменными. Для построения графиков всех этих корреляций мы воспользуемся функцией `pairgrid`:

```
iris.head(3)
```

```
g = sns.PairGrid(iris, vars=['sepal_length', 'sepal_width', 'petal_length',
                             'petal_width'],
                 hue='species', palette='RdBu_r')
g.map(plt.scatter, alpha=0.8)
g.add_legend();
```



Круговые диаграммы.

```
labels = 'Птицы', 'Кошки', 'Собаки', 'Рыбы'
```

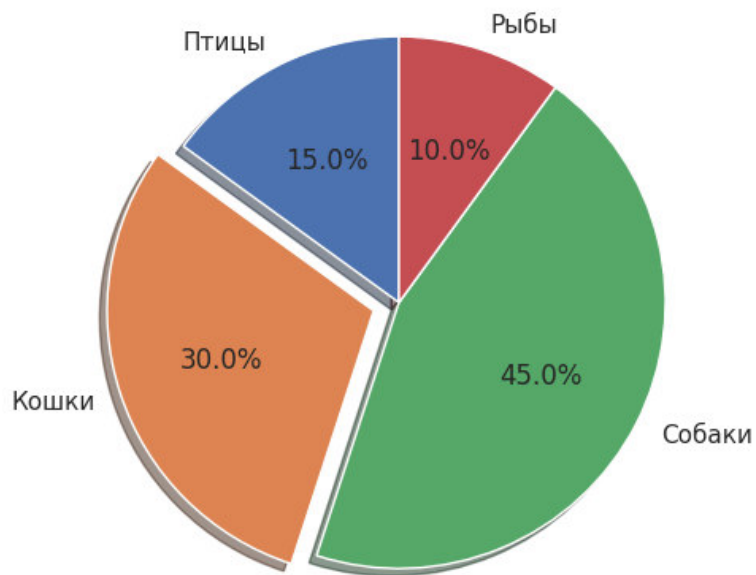
```
sizes = [15, 30, 45, 10]
```

```
explode = (0, 0.1, 0, 0) # Выделяем отдельно второй раздел
```

```
fig1, ax1 = plt.subplots()
```

```
ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',  
        shadow=True, startangle=90)
```

```
ax1.axis('equal') # Equal - Равное соотношение сторон гарантирует, что  
# пирог будет нарисован в виде круга.
```



Загрузка рисунков.

```
image = plt.imread('/content/sample_data/sibir.jpg') #указать путь к файлу  
рисунка
```

```
fig, ax = plt.subplots()  
ax.imshow(image)  
ax.axis('off') #отключить оси координат x и y  
plt.show()
```



Задания

- 1) Выполнить все примеры выше в среде Google Colab.

- 2) Скачать фотографию по ссылке:
https://drive.google.com/file/d/1vi3oD8R62cal6FfxjJuupCvrBPUUPjk3/view?usp=drive_link
или по QR коду:



и загрузить в блокнот Google Colab. Отключить отображение осей координат. Подписать фотографию.

- 3) Построить круговую диаграмму по данным своего варианта. Отделить один из разделов построенной диаграммы. Данные можно скачать по ссылке:
<https://drive.google.com/file/d/1rvVc6kEqpSzk9yKuJSpRGg6WXPFFIstS/view?usp=sharing>
или по QR коду:



Полноценную версию блокнота Google Colab в pdf формате можно посмотреть по QR коду:



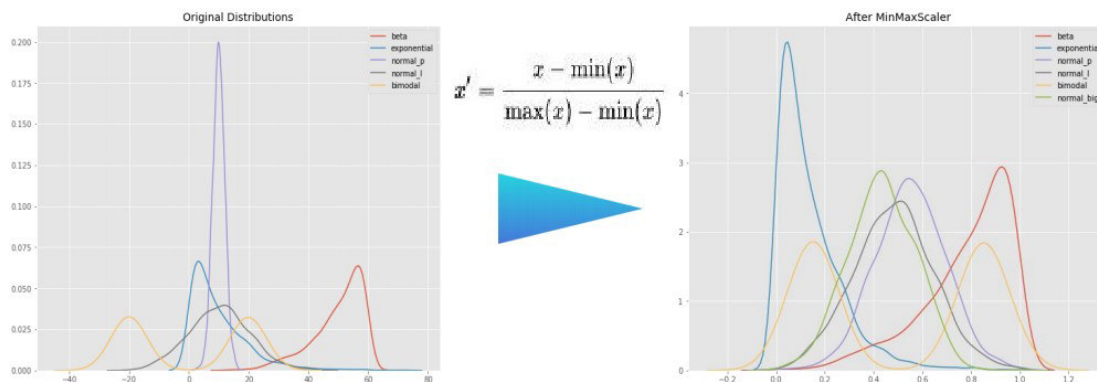
НОРМАЛИЗАЦИЯ И СТАНДАРТИЗАЦИЯ ДАННЫХ.

Нормализация (normalization) и стандартизация (standardization) являются методами изменения диапазонов значений — шкалирования. Шкалирование особенно полезно в машинном обучении (Machine Learning), поскольку разные атрибуты могут измеряться в разных диапазонах, или значения одного атрибута варьируются слишком сильно. Например, один атрибут имеет диапазон от 0 до 1, а второй — от 1 до 1000. Для задачи регрессии второй атрибут оказывал бы большое влияние на обучение, хотя не факт, что он является более важным, чем первый. Нормализация и стандартизация отличаются своими подходами:

- Нормализация подразумевает изменение диапазонов в данных без изменения формы распределения;
- Стандартизация изменяет форму распределения данных (приводится к нормальному распределению).

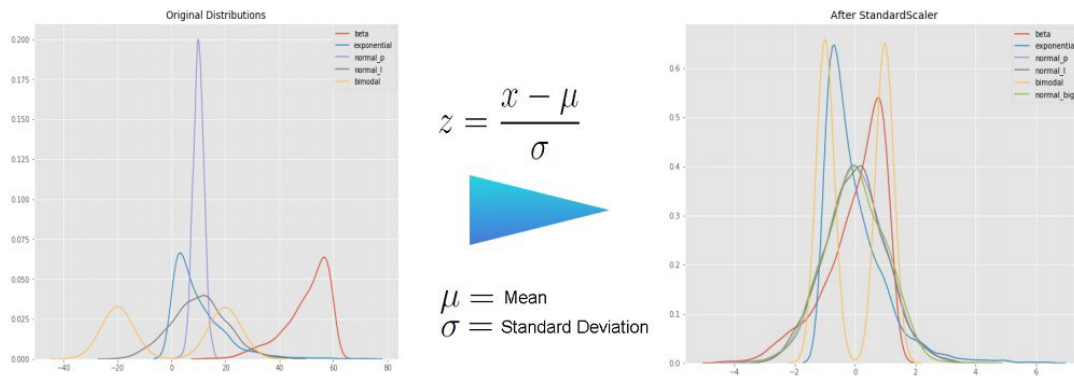
Нормализация

Нормализация – это процедура предобработки входной информации (обучающих, тестовых и валидационных выборок, а также реальных данных), при которой значения признаков во входном векторе приводятся к некоторому заданному диапазону, например, $[0 \dots 1]$ или $[-1 \dots 1]$.



Стандартизация

Стандартизация - приведения к определённому формату и представлению, которые обеспечивают их корректное применение в многомерном анализе, совместных исследованиях, сложных технологиях аналитической обработки.



Выполнение нормализации и стандартизации данных в среде Google Colab.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set()
```

Ввод данных

```
x = np.array([4, 4, 3, 3, 3.5, 4, 4.5, 3, 3.5, 5, 5, 4, 5, 3.5, 5, 4, 3.5, 4, 3.5, 5, 4, 4,
4, 4, 4])
print(x)
```

```
[4. 4. 3. 3. 3.5 4. 4.5 3. 3.5 5. 5. 4. 5. 3.5 5. 4. 3.5 4. 3.5 5. 4. 4.
3.5 5. 4. 4. 4. 4. ]
```

Нормализуем данные

```
y = (x - np.min(x)) / (np.max(x) - np.min(x))
```

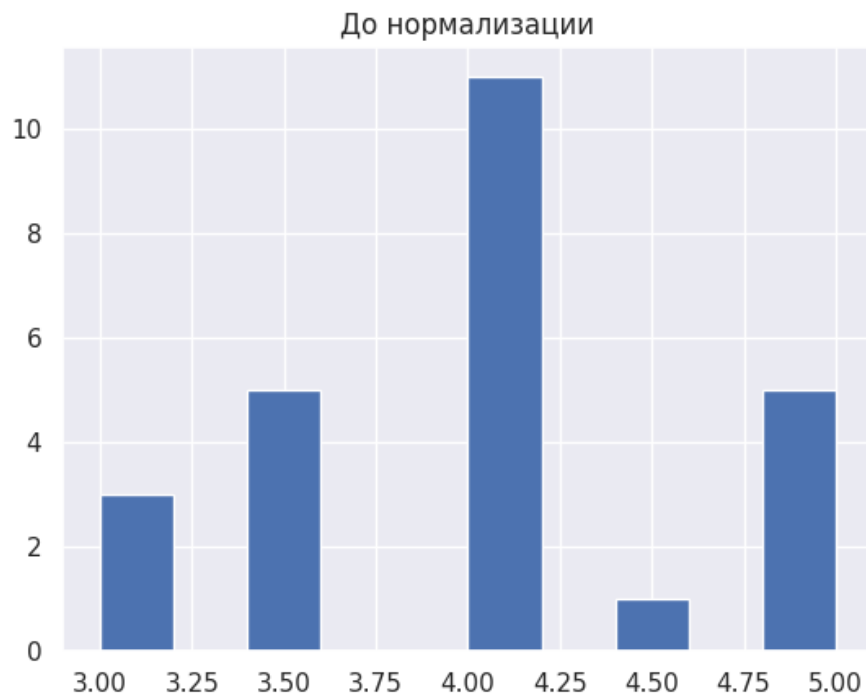
Вывод результата

```
print(y)
```

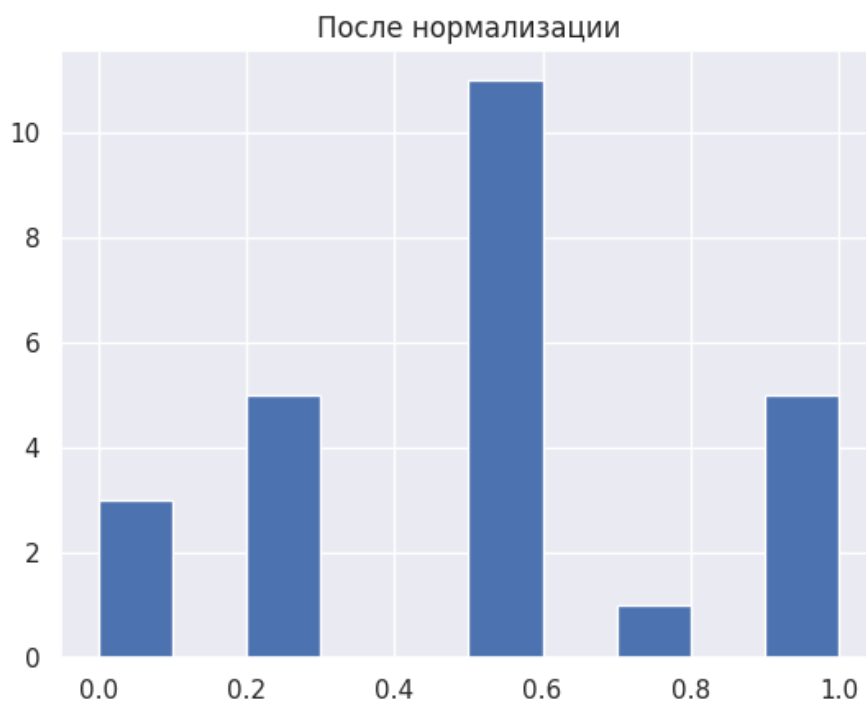
```
[0.5 0.5 0. 0. 0.25 0.5 0.75 0. 0.25 1. 1. 0.5 1. 0.25
1. 0.5 0.25 0.5 0.25 1. 0.5 0.5 0.5 0.5 0.5 ]
```

Строим гистограммы до и после нормализации

```
plt.hist(x);
plt.title('До нормализации')
plt.show()
plt.hist(y);
plt.title('После нормализации')
```



Text(0.5, 1.0, 'После нормализации')



Стандартизация данных

`print(x)`

```
[4. 4. 3. 3. 3.5 4. 4.5 3. 3.5 5. 5. 4. 5. 3.5 5. 4. 3.5 4.
 3.5 5. 4. 4. 4. 4. 4.]
```

Вычисляем среднее значение и среднее квадратическое отклонение

```
mu = np.mean(x)
sigma = np.std(x)
print('Среднее значение = ', mu)
print('стандартное отклонение = ', sigma)
```

Среднее значение = 4.0
стандартное отклонение = 0.6164414002968976

Выполняем стандартизацию

```
z = (x - mu) / sigma
```

Печатаем результат

```
print(z)
```

```
[ 0.      0.      -1.62221421 -1.62221421 -0.81110711  0.
  0.81110711 -1.62221421 -0.81110711  1.62221421  1.62221421  0.
  1.62221421 -0.81110711  1.62221421  0.      -0.81110711  0.
 -0.81110711  1.62221421  0.      0.      0.      0.
  0.      ]
```

Вычислим среднее и стандартное отклонение после нормализации т.е. для z. Если стандартизация выполнена верно, то среднее значение должно быть равно 0, а стандартное отклонение равно 1.

```
print(np.round(np.mean(z), 3))
print(np.std(z))
```

0.0
1.0

Строим гистограммы до и после стандартизации

```
plt.hist(x);
plt.title('До стандартизации')
plt.show()
plt.hist(z);
plt.title('После стандартизации')
```



Text(0.5, 1.0, 'После стандартизации')



Использование специальных библиотек

Библиотека MinMaxScaler для нормализации

from sklearn.preprocessing **import** MinMaxScaler

Загрузка данных

```
x = np.array([4, 4, 3, 3, 3.5, 4, 4.5, 3, 3.5, 5, 5, 4, 5, 3.5, 5, 4, 3.5, 4, 3.5, 5, 4, 4,
4, 4, 4])
```

Изменение размерности массива x

```
x = np.reshape(x, (-1, 1))
```

Применение библиотеки MinMaxScaler

MinMaxScaler — это преобразователь, который приводит значения признаков к заданному диапазону. Этот метод широко используется в машинном обучении для нормализации данных. В Python для работы с этим методом используется библиотека Scikit-learn.

Основная функция MinMaxScaler — это преобразование каждого значения в новое значение, которое находится в заданном интервале. Минимальное и максимальное значения можно указать вручную, либо их можно вычислить автоматически на основе имеющихся данных.

```
minmax_scaler = MinMaxScaler()
x_normal = minmax_scaler.fit_transform(x)
```

вывод результата

```
print('Результат работы библиотеки MinMaxScaler: \n', x_normal.T)
```

Результат работы библиотеки MinMaxScaler:

```
[[0.5 0.5 0. 0. 0.25 0.5 0.75 0. 0.25 1. 1. 0.5 1. 0.25
 1. 0.5 0.25 0.5 0.25 1. 0.5 0.5 0.5 0.5 0.5 ]]
```

```
print('Результат ручного преобразования: \n', y)
```

Результат ручного преобразования:

```
[0.5 0.5 0. 0. 0.25 0.5 0.75 0. 0.25 1. 1. 0.5 1. 0.25
 1. 0.5 0.25 0.5 0.25 1. 0.5 0.5 0.5 0.5 0.5 ]
```

Библиотека StandardScaler для стандартизации

StandardScaler — класс Scikit-learn, подвергающий передаваемый объект Стандартизации.

```
from sklearn.preprocessing import StandardScaler
```

Загрузка данных

```
x = np.array([4, 4, 3, 3, 3.5, 4, 4.5, 3, 3.5, 5, 5, 4, 5, 3.5, 5, 4, 3.5, 4, 3.5, 5, 4, 4,
4, 4, 4])
```

```
x.shape
```

```
(25,)
```

Изменение размерности массива x

```
x = np.reshape(x, (-1, 1))
```

```
x.shape
```

```
(25, 1)
```

Применение библиотеки StandardScaler

```
standart_scaler = StandardScaler()
```

```
x_standart = standart_scaler.fit_transform(x)
```

вывод результата

```
print('Результат работы библиотеки StandardScaler: \n', x_standart.T)
```

Результат работы библиотеки StandardScaler:

```
[[ 0.      0.     -1.62221421 -1.62221421 -0.81110711  0.
  0.81110711 -1.62221421 -0.81110711  1.62221421  1.62221421  0.
  1.62221421 -0.81110711  1.62221421  0.     -0.81110711  0.
 -0.81110711  1.62221421  0.      0.      0.      0.
  0.   ]]
```

```
print('Результат ручного преобразования: \n', z)
```

Результат ручного преобразования:

```
[ 0.      0.     -1.62221421 -1.62221421 -0.81110711  0.
  0.81110711 -1.62221421 -0.81110711  1.62221421  1.62221421  0.
  1.62221421 -0.81110711  1.62221421  0.     -0.81110711  0.
 -0.81110711  1.62221421  0.      0.      0.      0.
  0.   ]
```

Работа с реальными данными.

Пример набора данных: ежедневная погода в городе Сиэтле.

Этот набор данных содержит ежедневные метеорологические наблюдения из Сиэтла за 2012-2015 годы включительно. Файл данных с 1461 наблюдением шести переменных:

date - дата наблюдения

precipitation - количество осадков (мм)

temp_max - максимальная температура (°C)

temp_min - минимальная температура (°C)

wind - средняя скорость ветра (м/с)

weather - описание погоды.

Загрузка данных

Файл примера можно скачать по ссылке:

https://drive.google.com/file/d/1VDXCYmdcOoJHxBlvz_3Y51LPjRoJ5Mq6/view?usp=sharing

или по QR коду:



```
dataset = pd.read_csv('/content/sample_data/seattle-weather.csv')
```

Вывод первых 5 строк таблицы

```
dataset.head()
```

	date	precipitation	temp_max	temp_min	wind	weather
0	2012-01-01	0.0	12.8	5.0	4.7	drizzle
1	2012-01-02	10.9	10.6	2.8	4.5	rain
2	2012-01-03	0.8	11.7	7.2	2.3	rain
3	2012-01-04	20.3	12.2	5.6	4.7	rain
4	2012-01-05	1.3	8.9	2.8	6.1	rain

Просмотр последних 5 строк таблицы

```
dataset.tail()
```

	date	precipitation	temp_max	temp_min	wind	weather
1456	2015-12-27	8.6	4.4	1.7	2.9	rain
1457	2015-12-28	1.5	5.0	1.7	1.3	rain
1458	2015-12-29	0.0	7.2	0.6	2.6	fog
1459	2015-12-30	0.0	5.6	-1.0	3.4	sun
1460	2015-12-31	0.0	5.6	-2.1	3.5	sun

Размер таблицы

dataset.shape

(1461, 6)

Информация о файле

dataset.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1461 entries, 0 to 1460
Data columns (total 6 columns):
#   Column          Non-Null Count  Dtype
---  -
0   date             1461 non-null   object
1   precipitation     1461 non-null   float64
2   temp_max         1461 non-null   float64
3   temp_min         1461 non-null   float64
4   wind             1461 non-null   float64
5   weather          1461 non-null   object
dtypes: float64(4), object(2)
memory usage: 68.6+ KB
```

Вычисление сводных показателей

Функция `describe()` возвращает некоторые общие статистические данные данных, включая среднее значение, стандартное отклонение, минимальный элемент, максимальный элемент и некоторые другие детали. Это отличный способ получить моментальный снимок данных, с которыми вы работаете, если набор данных вам относительно неизвестен.

dataset.describe()

	precipitation	temp_max	temp_min	wind
count	1461.000000	1461.000000	1461.000000	1461.000000
mean	3.029432	16.439083	8.234771	3.241136
std	6.680194	7.349758	5.023004	1.437825
min	0.000000	-1.600000	-7.100000	0.400000
25%	0.000000	10.600000	4.400000	2.200000
50%	0.000000	15.600000	8.300000	3.000000
75%	2.800000	22.200000	12.200000	4.000000
max	55.900000	35.600000	18.300000	9.500000

Стандартизация с помощью библиотеки StandardScaler.

Преобразовываем только числовые столбцы: precipitation, temp_max, temp_min, wind

```
standard_scaler = StandardScaler()
dataset_standard = standard_scaler.fit_transform(dataset[['precipitation',
'temp_max', 'temp_min', 'wind']])
```

Преобразуем полученные результаты в таблицу

```
df1 = pd.DataFrame(dataset_standard, columns = ['precipitation', 'temp_max',
'temp_min', 'wind'])
df1.head()
```

	precipitation	temp_max	temp_min	wind
0	0.000000	0.387097	0.476378	0.472527
1	0.194991	0.327957	0.389764	0.450549
2	0.014311	0.357527	0.562992	0.208791
3	0.363148	0.370968	0.500000	0.472527
4	0.023256	0.282258	0.389764	0.626374

Вычисляем сводные показатели

```
df1.describe()
```

	precipitation	temp_max	temp_min	wind
count	1.461000e+03	1.461000e+03	1461.000000	1.461000e+03
mean	2.674870e-17	9.726800e-18	0.000000	-8.754120e-17
std	1.000342e+00	1.000342e+00	1.000342	1.000342e+00
min	-4.536499e-01	-2.455218e+00	-3.053954	-1.976672e+00
25%	-4.536499e-01	-7.947312e-01	-0.763703	-7.243529e-01
50%	-4.536499e-01	-1.142038e-01	0.012991	-1.677664e-01
75%	-3.435685e-02	7.840924e-01	0.789684	5.279666e-01
max	7.917235e+00	2.607906e+00	2.004513	4.354498e+00

Для удобства округлим полученные результаты

```
np.round(df1.describe(), 3)
```

	precipitation	temp_max	temp_min	wind
count	1461.000	1461.000	1461.000	1461.000
mean	0.000	0.000	0.000	-0.000
std	1.000	1.000	1.000	1.000
min	-0.454	-2.455	-3.054	-1.977
25%	-0.454	-0.795	-0.764	-0.724
50%	-0.454	-0.114	0.013	-0.168
75%	-0.034	0.784	0.790	0.528
max	7.917	2.608	2.005	4.354

Задания.

- 1) Выполнить все примеры выше в среде Google Colab.
- 2) Выполнить нормализацию и стандартизацию данных при помощи формул для своего варианта.
- 3) Выполнить нормализацию и стандартизацию данных при помощи библиотек MinMaxScaler() и StandardScaler() для своего варианта.

Номер варианта соответствует порядковому номеру в списке. Варианты можно скачать по ссылке:

<https://drive.google.com/file/d/1FQy2FVNlph8dy889Gvf8yVKMNC3XLUoH/view?usp=sharing>

или по QR коду:



Полноценную версию блокнота Google Colab в pdf формате можно посмотреть по QR коду:



ОБРАБОТКА КАТЕГОРИАЛЬНЫХ ДАННЫХ.

Категориальные данные — один из распространенных типов нечисловых данных. До сих пор мы работали только с числовыми значениями. Тем не менее, реальные наборы данных нередко содержат один или большее число столбцов категориальных признаков. Когда речь идет о категориальных данных, мы должны дополнительно проводить различие между *именными* и *порядковыми* признаками. Порядковые признаки можно понимать как категориальные значения, которые допускают сортировку или упорядочение. Например, размер футболки был бы порядковым признаком, потому что мы можем определить порядок **XL** > **L** > **M**. Напротив, именные признаки не подразумевают какого-либо порядка и в контексте примера с футболками мы могли бы считать именным признаком цвет футболки, т.к. обычно не имеет смысла говорить что-то вроде того, что красный больше синего.

Выполнение обработки категориальных данных в среде Google Colab

```
import numpy as np
import pandas as pd
```

Кодирование категориальных данных с помощью библиотеки Pandas.

Pandas — более новый пакет, надстройка над библиотекой NumPy, обеспечивающий эффективную реализацию класса DataFrame. Объекты DataFrame — многомерные массивы с метками для строк и столбцов, а также зачастую с неоднородным типом данных и/или пропущенными данными. Помимо удобного интерфейса для хранения маркированных данных, библиотека Pandas реализует множество операций для работы с данными хорошо знакомых пользователям фреймворков баз данных и электронных таблиц. Структура данных ndarray библиотеки NumPy предоставляет все необходимые возможности для работы с хорошо упорядоченными данными в задачах численных вычислений. Для этой цели библиотека NumPy отлично подходит, однако имеет свои ограничения, которые становятся заметными, чуть только нам потребуется немного больше гибкости (маркирование данных, работа с пропущенными данными и т. д.). Эти ограничения проявляются также при попытках выполнения операций, неподходящих для поэлементного транслирования (группировки, создание сводных таблиц и т. д.). Такие операции являются важной частью анализа данных с меньшей степенью структурированности, содержащихся во многих формах окружающего мира. Библиотека Pandas, особенно ее объекты Series и DataFrame, основана на структурах массивов библиотеки NumPy и обеспечивает эффективную работу над подобными задачами «очистки данных».

Создадим простой пример:

```
df = pd.DataFrame([['Зеленый', 'M', 10.1], ['Красный', 'L', 13.5], ['Синий', 'XL', 15.3]])
df.columns = ['Цвет', 'Размер', 'Стоимость'] # названия столбцов таблицы
df # вывод на экран результата
```

	Цвет	Размер	Стоимость
0	Зеленый	M	10.1
1	Красный	L	13.5
2	Синий	XL	15.3

В предыдущем выводе видно, что вновь созданный объект DataFrame содержит столбцы с именованным признаком (цвет), порядковым признаком (размер) и числовым признаком (стоимость).

Отображение порядковых признаков

Чтобы обеспечить корректную интерпретацию порядковых признаков алгоритмами обучения, нам нужно преобразовать строковые значения в целые числа. К сожалению, нет удобной функции, которая могла бы автоматически вывести правильный порядок меток в признаке **размер**, поэтому нам придется определить отображение вручную. В представленном ниже простом примере мы предполагаем, что нам известна числовая разница между признаками, скажем, $XL = L + 1 = M + 2$:

```
size_map = {'XL': 3, 'L': 2, 'M': 1} # Создание словаря замены
df['Размер'] = df['Размер'].map(size_map) # замена значений в столбце 'Размер'
df # Вывод результата
```

	Цвет	Размер	Стоимость
0	Зеленый	1	10.1
1	Красный	2	13.5
2	Синий	3	15.3

Выполнение унитарного кодирования на именованных признаках.

Для этих целей в библиотеке scikit-learn реализован удобный класс LabelEncoder. LabelEncoder - это инструмент, который используется для

преобразования категориальных данных в числовые значения. Это позволяет алгоритмам работать с такими данными и принимать решения на их основе.

```
from sklearn.preprocessing import LabelEncoder # импорт класса
```

Поскольку оценщики scikit-learn, предназначенные для классификации, трактуют метки классов как категориальные данные, что не подразумевает наличие любого порядка (именные данные), для кодирования строковых меток в целые числа мы использовали удобный класс LabelEncoder. Применим LabelEncoder для трансформации именного столбца цвета ('Цвет') в нашем наборе данных:

```
X = df[['Цвет', 'Размер', 'Стоимость']].values # копируем в переменную X  
только значения таблицы (без заголовка)  
le = LabelEncoder() # Создаем объект класса LabelEncoder  
X[:, 0] = le.fit_transform(X[:, 0]) # Выполнение преобразования первого  
столбца ('Цвет') таблицы,  
#результат перезаписывается в тот же столбец  
X # вывод результата  
  
array([[0, 1, 10.1],  
       [1, 2, 13.5],  
       [2, 3, 15.3]], dtype=object)
```

После выполнения предыдущего кода первый столбец в NumPy-массиве X теперь хранит новые значения цвета, закодированные следующим образом:

- **Синий** = 0
- **Зеленый** = 1
- **Красный** = 2

Если мы на этом остановимся и передадим массив нашему классификатору, то допустим наиболее распространенную ошибку при работе с категориальными данными. Заметили проблему? Хотя значения цвета не должны поступать в каком-либо порядке, алгоритм обучения теперь предполагает, что **Зеленый** больше **Синий**, а **Красный** больше **Зеленый**. Несмотря на некорректность такого предположения, алгоритм по-прежнему мог бы выдавать пригодные результаты. Однако результаты подобного рода не были бы оптимальными.

Общепринятый обход описанной проблемы предусматривает использование приема, называемого унитарным кодированием (**one-hot encoding**) или кодированием с одним активным состоянием. Идея подхода

заключается в том, чтобы создать новый фиктивный признак для каждого уникального значения в столбце именного признака. В рассматриваемом примере мы преобразовали бы признак ‘Цвет’ в три новых признака: Синий, Зеленый и Красный. Затем с помощью двоичных значений можно было бы указывать цвет (‘Цвет’) образца; скажем, образец синего цвета кодировался бы как Синий=1, Зеленый=0, Красный=0.

Для выполнения такой трансформации мы можем применить класс **OneHotEncoder**, реализованный в модуле `sklearn.preprocessing`. **OneHotEncoder** — это процесс преобразования категориальных признаков в бинарные переменные, где каждая категория принимает значение 1 или 0. При использовании **OneHotEncoder** каждая категория признака представляется отдельной колонкой, в которой только одно значение равно 1, остальные — 0. Это позволяет моделям лучше понять взаимосвязь между категориями и результатом.

```
from sklearn.preprocessing import OneHotEncoder # импортирование
X = df[['Цвет', 'Размер', 'Стоимость']].values # копируем в переменную X
# только значения таблицы (без заголовка)
ohe = OneHotEncoder() # Создаем объект класса OneHotEncoder
ohe.fit_transform(X[:, 0].reshape(-1, 1)).toarray() # Выполнение
# преобразования
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

Обратите внимание, что мы применили **OneHotEncoder** только к единственному столбцу, (`X[:, 0].reshape(-1, 1)`), чтобы избежать модификации остальных двух столбцов в массиве.

Еще более удобный способ создания таких фиктивных признаков через унитарное кодирование предполагает использование метода **get_dummies**, реализованного в **pandas**. Будучи примененным к объекту **DataFrame**, метод **get_dummies** преобразует только строковые столбцы и оставит все остальные столбцы неизменными:

```
pd.get_dummies(df)
```

	Размер	Стоимость	Цвет_Зеленый	Цвет_Красный	Цвет_Синий
0	1	10.1	1	0	0
1	2	13.5	0	1	0
2	3	15.3	0	0	1

Если признаки сильно взаимосвязаны, тогда обращаться матрицы будет трудно в вычислительном плане, что может привести к получению численно неустойчивых оценок. Чтобы сократить взаимосвязь между переменными, мы можем просто удалить один столбец признака из закодированного унитарным кодом массива. Обратите внимание, что в результате удаления столбца признака мы не теряем какую-то важную информацию.

Если мы применяем функцию *get_dummies*, то можем удалить первый столбец, указав **True** для параметра *drop_first*:

```
pd.get_dummies(df, drop_first = True)
```

	Размер	Стоимость	Цвет_Красный	Цвет_Синий
0	1	10.1	0	0
1	2	13.5	1	0
2	3	15.3	0	1

Работа с реальными данными.

Пример *Auto-mpg*.

Этот файл представляет собой список моделей автомобилей с некоторыми их характеристиками. Файл получен со страницы проекта Machine Learning Repository Калифорнийского университета. Данные были собраны в 1983 году в университете Карнеги-Мелон. Более подробная информация доступна по ссылке: <https://archive.ics.uci.edu/ml/datasets/auto+mpg>.

Файл **auto-mpg** представляет собой текстовый файл в кодировке UTF-8. Каждая строка файла описывает одну модель автомобиля.

Столбцы файла представляют собой следующие данные:

mpg: потребление топлива, в милях на галлон.

cylinders: число цилиндров в двигателе

displacement: рабочий объём двигателя (видимо, в кубических дюймах)

horsepower: мощность двигателя, в лошадиных силах.

weight: масса автомобиля, в фунтах.

acceleration: разгон до 60 миль/час, в секундах

model year: модельный год, две цифры. То есть, например, модельный год 74 означает 1974

origin: Регион производства. Значения: США, Европа, Юго-восточная Азия

car name: Название

Файл можно скачать по ссылке:

<https://drive.google.com/file/d/1LLB2kHaOI7URse2TYnyd20vg2lO98rMz/view?usp=sharing>

или по QR коду:



Загрузка данных:

```
data = pd.read_csv('/content/sample_data/Auto-mpg.csv')
```

Просмотр первых 5 строк таблицы:

```
data.head(7)
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130	3504	12.0	70	USA	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	USA	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	USA	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	USA	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	USA	ford torino
5	15.0	8	429.0	198	4341	10.0	70	USA	ford galaxie 500
6	14.0	8	454.0	220	4354	9.0	70	USA	chevrolet impala

Выведем на экран список уникальных значений в столбце ['origin']:

```
np.unique(data['origin'].values)
```

```
array(['Europe', 'Japan', 'USA'], dtype=object)
```

Закодируем столбец **‘origin’** с помощью функции **get_dummies**: с помощью параметра **columns** указываем, что необходимо кодировать только столбец **‘origin’**.

```
data_new = pd.get_dummies(data, columns = ['origin']) # выполняем
кодирование и результат сохраним в новую таблицу
data_new.head() #выводим результат
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	car name	origin_Europe	origin_Japan	origin_USA
0	18.0	8	307.0	130	3504	12.0	70	chevrolet chevelle malibu	0	0	1
1	15.0	8	350.0	165	3693	11.5	70	buick skylark 320	0	0	1
2	18.0	8	318.0	150	3436	11.0	70	plymouth satellite	0	0	1
3	16.0	8	304.0	150	3433	12.0	70	amc rebel sst	0	0	1
4	17.0	8	302.0	140	3449	10.5	70	ford torino	0	0	1

Уникальные имена в столбце **['car name']**:

```
np.unique(data['car name'].values)
```

```
array(['amc ambassador brougham', 'amc ambassador dpl',
      'amc ambassador sst', 'amc concord', 'amc concord d/l',
      'amc concord dl', 'amc concord dl 6', 'amc gremlin', 'amc hornet',
      'amc hornet sportabout (sw)', 'amc matador', 'amc matador (sw)',
      'amc pacer', 'amc pacer d/l', 'amc rebel sst', 'amc spirit dl',
      'audi 100 ls', 'audi 100ls', 'audi 4000', 'audi 5000',
      'audi 5000s (diesel)', 'audi fox', 'bmw 2002', 'bmw 320i',
      'buick century', 'buick century 350', 'buick century limited',
      'buick century luxus (sw)', 'buick century special',
      'buick electra 225 custom', 'buick estate wagon (sw)',
      'buick lesabre custom', 'buick opel isuzu deluxe',
      'buick regal sport coupe (turbo)', 'buick skyhawk',
      'buick skylark', 'buick skylark 320', 'buick skylark limited',
      'cadillac eldorado', 'cadillac seville', 'capri ii',
      'chevrolet chevelle malibu', 'chevrolet bel air',
      'chevrolet camaro', 'chevrolet caprice classic',
      'chevrolet cavalier', 'chevrolet cavalier 2-door',
      'chevrolet cavalier wagon', 'chevrolet chevelle concours (sw)',
      'chevrolet chevelle malibu', 'chevrolet chevelle malibu classic',
      'chevrolet chevette', 'chevrolet citation', 'chevrolet concours',
      'chevrolet impala', 'chevrolet malibu',
      'chevrolet malibu classic (sw)', 'chevrolet monte carlo',
      'chevrolet monte carlo landau', 'chevrolet monte carlo s',
      'chevrolet monza 2+2', 'chevrolet nova', 'chevrolet nova custom',
      'chevrolet vega', 'chevrolet vega (sw)', 'chevrolet vega 2300',
      'chevrolet woody', 'chevy c10', 'chevy c20', 'chevy s-10',
```

'chrysler cordoba', 'chrysler lebaron medallion',
 'chrysler lebaron salon', 'chrysler lebaron town @ country (sw)',
 'chrysler new yorker brougham', 'chrysler newport royal',
 'datsun 1200', 'datsun 200-sx', 'datsun 200sx', 'datsun 210',
 'datsun 210 mpg', 'datsun 280-zx', 'datsun 310', 'datsun 310 gx',
 'datsun 510', 'datsun 510 (sw)', 'datsun 510 hatchback',
 'datsun 610', 'datsun 710', 'datsun 810', 'datsun 810 maxima',
 'datsun b-210', 'datsun b210', 'datsun b210 gx',
 'datsun f-10 hatchback', 'datsun pl510', 'dodge aries se',
 'dodge aries wagon (sw)', 'dodge aspen', 'dodge aspen 6',
 'dodge aspen se', 'dodge challenger se', 'dodge charger 2.2',
 'dodge colt', 'dodge colt (sw)', 'dodge colt hardtop',
 'dodge colt hatchback custom', 'dodge colt m/m',
 'dodge coronet brougham', 'dodge coronet custom',
 'dodge coronet custom (sw)', 'dodge d100', 'dodge d200',
 'dodge dart custom', 'dodge diplomat', 'dodge magnum xe',
 'dodge monaco (sw)', 'dodge monaco brougham', 'dodge omni',
 'dodge rampage', 'dodge st. regis', 'fiat 124 sport coupe',
 'fiat 124 tc', 'fiat 124b', 'fiat 128', 'fiat 131',
 'fiat strada custom', 'fiat x1.9', 'ford country',
 'ford country squire (sw)', 'ford escort 2h', 'ford escort 4w',
 'ford f108', 'ford f250', 'ford fairmont', 'ford fairmont (auto)',
 'ford fairmont (man)', 'ford fairmont 4', 'ford fairmont futura',
 'ford fiesta', 'ford futura', 'ford galaxie 500',
 'ford gran torino', 'ford gran torino (sw)', 'ford granada',
 'ford granada ghia', 'ford granada gl', 'ford granada l',
 'ford ltd', 'ford ltd landau', 'ford maverick', 'ford mustang',
 'ford mustang cobra', 'ford mustang gl', 'ford mustang ii',
 'ford mustang ii 2+2', 'ford pinto', 'ford pinto (sw)',
 'ford pinto runabout', 'ford ranger', 'ford thunderbird',
 'ford torino', 'ford torino 500', 'hi 1200d', 'honda accord',
 'honda accord cvcc', 'honda accord lx', 'honda civic',
 'honda civic (auto)', 'honda civic 1300', 'honda civic 1500 gl',
 'honda civic cvcc', 'honda prelude', 'maxda glc deluxe',
 'maxda rx3', 'mazda 626', 'mazda glc', 'mazda glc 4',
 'mazda glc custom', 'mazda glc custom l', 'mazda glc deluxe',
 'mazda rx-4', 'mazda rx-7 gs', 'mazda rx2 coupe',
 'mercedes benz 300d', 'mercedes-benz 240d', 'mercedes-benz 280s',
 'mercury capri 2000', 'mercury capri v6',
 'mercury cougar brougham', 'mercury grand marquis',
 'mercury lynx l', 'mercury marquis', 'mercury marquis brougham',
 'mercury monarch', 'mercury monarch ghia', 'mercury zephyr',
 'mercury zephyr 6', 'nissan stanza xe',
 'oldsmobile cutlass ciera (diesel)', 'oldsmobile cutlass ls',

```

'oldsmobile cutlass salon brougham', 'oldsmobile cutlass supreme',
'oldsmobile delta 88 royale', 'oldsmobile omega',
'oldsmobile omega brougham', 'oldsmobile starfire sx',
'oldsmobile vista cruiser', 'opel 1900', 'opel manta',
'peugeot 304', 'peugeot 504', 'peugeot 504 (sw)',
'peugeot 505s turbo diesel', 'peugeot 604sl', "plymouth 'cuda 340",
'plymouth arrow gs', 'plymouth champ', 'plymouth cricket',
'plymouth custom suburb', 'plymouth duster', 'plymouth fury',
'plymouth fury gran sedan', 'plymouth fury iii',
'plymouth grand fury', 'plymouth horizon', 'plymouth horizon 4',
'plymouth horizon miser', 'plymouth horizon tc3',
'plymouth reliant', 'plymouth sapporo', 'plymouth satellite',
'plymouth satellite custom', 'plymouth satellite custom (sw)',
'plymouth satellite sebring', 'plymouth valiant',
'plymouth valiant custom', 'plymouth volare',
'plymouth volare custom', 'plymouth volare premier v8',
'pontiac astro', 'pontiac catalina', 'pontiac catalina brougham',
'pontiac firebird', 'pontiac grand prix', 'pontiac grand prix lj',
'pontiac j2000 se hatchback', 'pontiac lemans v6',
'pontiac phoenix', 'pontiac phoenix lj', 'pontiac safari (sw)',
'pontiac sunbird coupe', 'pontiac ventura sj', 'renault 12 (sw)',
'renault 12tl', 'renault 18i', 'renault 5 gtl',
'renault lecar deluxe', 'saab 99e', 'saab 99gle', 'saab 99le',
'subaru', 'subaru dl', 'toyota carina', 'toyota celica gt',
'toyota celica gt liftback', 'toyota corolla',
'toyota corolla 1200', 'toyota corolla 1600 (sw)',
'toyota corolla liftback', 'toyota corolla tercel',
'toyota corona', 'toyota corona hardtop', 'toyota corona liftback',
'toyota corona mark ii', 'toyota cressida', 'toyota mark ii',
'toyota starlet', 'toyota tercel', 'toyota corona mark ii (sw)',
'triumph tr7 coupe', 'volkswagen rabbit',
'volkswagen 1131 deluxe sedan', 'volkswagen 411 (sw)',
'volkswagen dasher', 'volkswagen jetta', 'volkswagen model 111',
'volkswagen rabbit', 'volkswagen rabbit custom',
'volkswagen rabbit custom diesel', 'volkswagen rabbit l',
'volkswagen scirocco', 'volkswagen super beetle',
'volkswagen type 3', 'volvo 144ea', 'volvo 145e (sw)',
'volvo 244dl', 'volvo 245', 'volvo 264gl', 'volvo diesel',
'vw dasher (diesel)', 'vw pickup', 'vw rabbit',
'vw rabbit c (diesel)', 'vw rabbit custom']], dtype=object)

```

При этом исходная таблица осталась без изменений:

```
data.head()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130	3504	12.0	70	USA	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	USA	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	USA	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	USA	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	USA	ford torino

Задания

- 1) Выполнить все примеры выше в среде Google Colab.
- 2) Выполнить обработку категориальных данных с помощью методов ***OneHotEncoder*** и ***get_dummies*** для данных своего варианта. Данные доступны по ссылке:

<https://drive.google.com/file/d/1qcDQcvPCSO-AX57LVwzYirgupy2Xbmxn/view?usp=sharing>

или по QR коду:



Полноценную версию блокнота Google Colab в pdf формате можно посмотреть по QR коду:



ОБРАБОТКА ОТСУТСТВУЮЩИХ ДАННЫХ

В данных часто бывают пропуски. Причины этого — неправильный ввод или сокрытие информации. Кажется, что лучший способ решить такую проблему — заполнить пробелы, но это не всегда верно. Неудачный выбор типа заполнения пропусков может ухудшить результаты. Реальные данные редко бывают очищенными и однородными. В частности, во многих интересных наборах данных некоторое количество данных отсутствует.

Типы пропусков данных.

- MCAR (Missing Completely At Random - Отсутствует совершенно случайно). В этой ситуации пропуски данных не зависят от наблюдаемых и ненаблюдаемых переменных. Пробелы в данных не ведут к систематической ошибке, но увеличивают случайную, потому что информации становится меньше. Например, при взятии медицинских анализов в записях некоторых пациентов могут отсутствовать значения из-за неправильной обработки лабораторных проб.
- MAR (Missing At Random - Отсутствует случайно). Например, на приеме терапевта у пожилых людей почти всегда измеряют давление, а у молодых — нет. Потому пропуски в графе «давление» чаще встречаются у молодых. При этом давление у них обычно ниже. Но, хотя распределение пропусков в графе «давление» не случайно, внутри каждой возрастной группы пациентов оно будет случайным.
- MNAR (Missing Not At Random - Отсутствует не случайно) Риск пропуска определяется данными. Но отличие MNAR в том, что вероятность пропуска в колонке зависит от информации в ней. Например, когда у людей спрашивают, принимали ли они наркотики, респонденты могут оставить поля пустыми, если не хотят это признавать.

Еще более затрудняет работу то, что в различных источниках данных отсутствующие данные могут быть помечены различным образом. Здесь и далее мы будем называть отсутствующие данные null, NaN или NA-значениями. Способ обработки отсутствующих данных библиотекой Pandas определяется тем, что она основана на пакете NumPy, в котором отсутствует встроенное понятие NAзначений для всех типов данных, кроме данных с плавающей точкой.

NaN: отсутствующие числовые данные

NaN, представляет собой специальное значение с плавающей точкой, распознаваемое всеми системами, использующими стандартное IEEE-представление чисел с плавающей точкой.

Выполнение обработки отсутствующих данных в среде Google Colab

```
import numpy as np
import pandas as pd
```

Пример:

```
val = np.array([1, np.nan, 3, 4, 5])
print(val)

[ 1. nan  3.  4.  5.]
```

Значение NaN в чем-то подобно «вирусу данных»: оно «заражает» любой объект, к которому «прикасается». Вне зависимости от операции результат арифметического действия с участием NaN будет равен NaN:

```
print('Сумма всех чисел в массиве: ', np.sum(val))
print('Min: ', np.min(val))
print('Max: ', np.max(val))
```

```
Сумма всех чисел в массиве: nan
Min: nan
Max: nan
```

Операции над пустыми значениями

Библиотека Pandas рассматривает значения None и NaN как взаимозаменяемые средства указания на отсутствующие или пустые значения. Существует несколько удобных методов для обнаружения, удаления и замены пустых значений в структурах данных библиотеки Pandas, призванных упростить работу с ними.

isnull() — генерирует булеву маску для отсутствующих значений.

notnull() — противоположность метода **isnull()**.

dropna() — возвращает отфильтрованный вариант данных.

fillna() — возвращает копию данных, в которой пропущенные значения заполнены или восстановлены.

```
val = pd.Series(val)
val
```

```
0    1.0
1    NaN
2    3.0
3    4.0
4    5.0
dtype: float64

vall.isnull().sum()

1
```

Выявление пустых значений

У структур данных библиотеки Pandas имеются два удобных метода для выявления пустых значений: `isnull()` и `notnull()`. Каждый из них возвращает булеву маску для данных.

Пример:

```
val2 = pd.Series([1, 2, np.nan, 3, 'hello', 4, None, 5])
print(val2)

0      1
1      2
2    NaN
3      3
4    hello
5      4
6    None
7      5
dtype: object

val2.isnull()

0    False
1    False
2     True
3    False
4    False
5    False
6     True
7    False
dtype: bool
```

Вычислим количество пустых данных:

```
val2.isnull().sum()
```

2

Вывод на экран пустых значений:

```
val2[val2.isnull()]
```

```
2    NaN
```

```
6    None
```

```
dtype: object
```

```
val2.notnull()
```

```
0    True
```

```
1    True
```

```
2    False
```

```
3    True
```

```
4    True
```

```
5    True
```

```
6    False
```

```
7    True
```

```
dtype: bool
```

Количество непустых данных:

```
val2.notnull().sum()
```

```
6
```

Вывод на экран не пустых значений:

```
val2[val2.notnull()]
```

```
0    1
```

```
1    2
```

```
3    3
```

```
4    hello
```

```
5    4
```

```
7    5
```

```
dtype: object
```

Удаление пустых значений

Существуют удобные методы: `dropna()` (отбрасывающий NA-значения) и `fillna()` (заполняющий NA-значения). Для объекта `Series` результат вполне однозначен:

```
val2.dropna()
```

```

0    1
1    2
3    3
4  hello
5    4
7    5
dtype: object

```

В случае объектов DataFrame существует несколько параметров. Рассмотрим следующий объект DataFrame:

```

df = pd.DataFrame([[1, np.nan, 3], [4, 5, 6], [np.nan, 8, 9]])
print(df)

```

```

   0  1  2
0  1.0 NaN 3
1  4.0 5.0 6
2  NaN 8.0 9

```

Нельзя выбросить из DataFrame отдельные значения, только целые строки или столбцы. В зависимости от приложения может понадобиться тот или иной вариант, так что функция `dropna()` предоставляет для случая объектов DataFrame несколько параметров. По умолчанию `dropna()` отбрасывает все строки, в которых присутствует хотя бы одно пустое значение:

```

df.dropna(axis = 0)

```

```

      0    1    2
1  4.0  5.0    6

```

В качестве альтернативы можно отбрасывать NA-значения по разным осям: задание параметра `axis=1` отбрасывает все столбцы, содержащие хотя бы одно пустое значение:

```

df.dropna(axis = 1)

```

```

      2
0  3
1  6
2  9

```

Заполнение пустых значений

Иногда предпочтительнее вместо отбрасывания пустых значений заполнить их каким-то допустимым значением. Это значение может быть фиксированным, например нулем, или интерполированным или восстановленным на основе «хороших» данных значением. Библиотека Pandas предоставляет метод `fillna()`, возвращающий копию массива с замененными пустыми значениями.

Пример:

```
val3 = pd.Series([1, np.nan, 3, 4, np.nan, 6])  
print(val3)
```

```
0    1.0  
1    NaN  
2    3.0  
3    4.0  
4    NaN  
5    6.0  
dtype: float64
```

Можно заполнить NA-элементы одним фиксированным значением, например, нулями:

```
val3.fillna(0)
```

```
0    1.0  
1    0.0  
2    3.0  
3    4.0  
4    0.0  
5    6.0  
dtype: float64
```

Можно задать параметр заполнения по направлению «вперед», копируя предыдущее значение в следующую ячейку:

```
print('Исходный массив: \n', val3)  
print('После замены:')  
val3.fillna(method = 'ffill') #заполнение по направлению «вперед»
```

Исходный массив:

```
0    1.0  
1    NaN  
2    3.0  
3    4.0  
4    NaN
```

```
5  6.0
dtype: float64
После замены:
0  1.0
1  1.0
2  3.0
3  4.0
4  4.0
5  6.0
dtype: float64
```

Или можно задать параметр заполнения по направлению «назад», копируя следующее значение в предыдущую ячейку:

```
val3.fillna(method = 'bfill') # аполение по направлению «назад»
```

```
0  1.0
1  3.0
2  3.0
3  4.0
4  6.0
5  6.0
dtype: float64
```

Для объектов DataFrame опции аналогичны, но в дополнение можно задать ось, по которой будет выполняться заполнение:

```
df[3] = np.nan
print(df)
```

```
   0  1  2  3
0  1.0 NaN 3 NaN
1  4.0 5.0 6 NaN
2  NaN 8.0 9 NaN
```

```
df.fillna(method = 'ffill', axis = 0) # значение axis = 0 идет по умолчанию
```

	0	1	2	3
0	1.0	1.0	3.0	3.0
1	4.0	5.0	6.0	6.0
2	NaN	8.0	9.0	9.0

Работа с реальными данными

База данных пассажиров парохода «Титаник».

Гибель Титаника является одним из самых печально известных кораблекрушений в истории. 15 апреля 1912 года во время своего первого плавания, Титаник затонул после столкновения с айсбергом, при этом погибло 1502 из 2224 пассажиров и членов экипажа.

Файл можно скачать по ссылке:

https://drive.google.com/file/d/1J8h1cNriTocnqPM-Lo_hrXmXACClcTaj/view?usp=sharing

или по QR коду:



Каждая строчка набора данных содержит следующие поля:

Survived — признак для каждого пассажира, обозначающий, выжил данный пассажир или нет (0 для умерших, 1 для выживших);

Pclass — класс пассажира (1 — высший, 2 — средний, 3 — низший);

Name — имя;

Sex — пол;

Age — возраст;

SibSp — количество братьев, сестер, сводных братьев, сводных сестер, супругов на борту титаника;

Parch — количество родителей, детей (в том числе приемных) на борту титаника;

Ticket — номер билета;

Fare — плата за проезд;

Cabin — каюта;

Embarked — порт посадки (С — Шербур; Q — Квинстаун; S — Саутгемптон).

В поле **Age** приводится количество полных лет. Для детей меньше 1 года — дробное.

Загрузка данных

```
data = pd.read_csv('/content/sample_data/Titanic.csv')
```

Просмотр первых 5 строк таблицы

```
data.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Посмотрим на размеры таблицы:

```
data.shape
```

```
(891, 12)
```

Общая информация о файле:

```
data.info()
```

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

```
RangeIndex: 891 entries, 0 to 890
```

```
Data columns (total 12 columns):
```

```
#   Column      Non-Null Count  Dtype
```

```
---  -
```

```
0   PassengerId  891 non-null    int64
```

```
1   Survived     891 non-null    int64
```

```
2   Pclass       891 non-null    int64
```

```
3   Name         891 non-null    object
```

```
4 Sex      891 non-null object
5 Age      714 non-null float64
6 SibSp    891 non-null int64
7 Parch    891 non-null int64
8 Ticket   891 non-null object
9 Fare     891 non-null float64
10 Cabin   204 non-null object
11 Embarked 889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

Проверим количество отсутствующих данных по всей таблице

```
data.isnull().sum()
```

```
PassengerId    0
Survived        0
Pclass          0
Name            0
Sex             0
Age            177
SibSp           0
Parch           0
Ticket          0
Fare            0
Cabin          687
Embarked        2
dtype: int64
```

Получаем, что в столбце **Embarked** 2 пропуска, в столбце **Age** у нас 177 пропусков, а в **Cabin** аж 687, что сильно больше половины. Простое удаление пропусков приведет к существенной потере информации.

Пропущенные значения в столбце **Cabin** заполним по направлению «вперед»:

```
data['Cabin'] = data['Cabin'].fillna(method = 'ffill')
```

Проверим количество отсутствующих данных по всей таблице

```
data.isnull().sum()
```

```
PassengerId    0
Survived        0
Pclass          0
Name            0
Sex             0
```

```
Age      177
SibSp    0
Parch    0
Ticket   0
Fare     0
Cabin    1
Embarked  2
dtype: int64
```

В столбце **Cabin** все еще остается одно пропущенное значение. Для его заполнения применим еще раз метод `fillna()`, но направление заполнения используем “назад”:

```
data['Cabin'] = data['Cabin'].fillna(method = 'bfill')
```

Проверим количество отсутствующих данных по всей таблице

```
data.isnull().sum()
```

```
PassengerId    0
Survived        0
Pclass         0
Name           0
Sex            0
Age           177
SibSp          0
Parch          0
Ticket         0
Fare           0
Cabin          0
Embarked        2
dtype: int64
```

Теперь все пропущенные значения столбца **Cabin** заполнены. Посмотрим еще раз на первые строки таблицы:

```
data.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	C85	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	C85	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	C123	S

Далее, обработаем пропущенные значения в столбце **Age** тремя разными способами. В *первом* способе просто удалим пропущенные значения, а результат сохраним в новую таблицу:

```
data_1 = data.dropna()
```

```
data_1.isnull().sum()
```

```
PassengerId  0
Survived     0
Pclass       0
Name         0
Sex          0
Age          0
SibSp        0
Parch        0
Ticket       0
Fare         0
Cabin        0
Embarked     0
dtype: int64
```

Посмотрим на размер полученной таблицы:

```
data_1.shape
```

```
(712, 12)
```

Количество строк в таблице уменьшилось, и тем самым потеряно часть информации.

Во *втором* способе заменим пропущенные значения на 0, а результат сохраним в новую таблицу:

```
data_2 = data.fillna(0)
```

Проверим количество отсутствующих данных по всей таблице

```
data_2.isnull().sum()
```

```
PassengerId  0
Survived     0
Pclass       0
Name         0
Sex          0
Age          0
SibSp        0
Parch        0
```

```
Ticket      0
Fare        0
Cabin       0
Embarked    0
dtype: int64
```

Размеры таблицы не изменились:

```
data_2.shape
```

```
(891, 12)
```

В *третьем* способе заменим пропущенные значения в столбце **Age** на среднее арифметическое по всему столбцу.

```
mean = np.mean(data['Age']) # вычисляем среднее арифметическое
print('среднее арифметическое: ', mean)
```

```
среднее арифметическое: 29.69911764705882
```

```
data['Age'] = data['Age'].fillna(mean) # заменяем пропущенные значения
```

Проверим количество отсутствующих данных по всей таблице

```
data.isnull().sum()
```

```
PassengerId  0
Survived     0
Pclass       0
Name         0
Sex          0
Age          0
SibSp        0
Parch        0
Ticket       0
Fare         0
Cabin        0
Embarked     2
dtype: int64
```

Размеры таблицы не изменились:

```
data.shape
```

```
(891, 12)
```

Задания

- 1) Выполнить все примеры выше в среде Google Colab.

- 2) Выполнить обработку отсутствующих значений с помощью стратегий 'ffill', 'bfill' и помощью среднего значения. Задания можно получить по ссылке:

<https://drive.google.com/file/d/1ilsHPs-chgRnshePuUHZJzUaEYcCeW3q/view?usp=sharing>

или по QR коду:



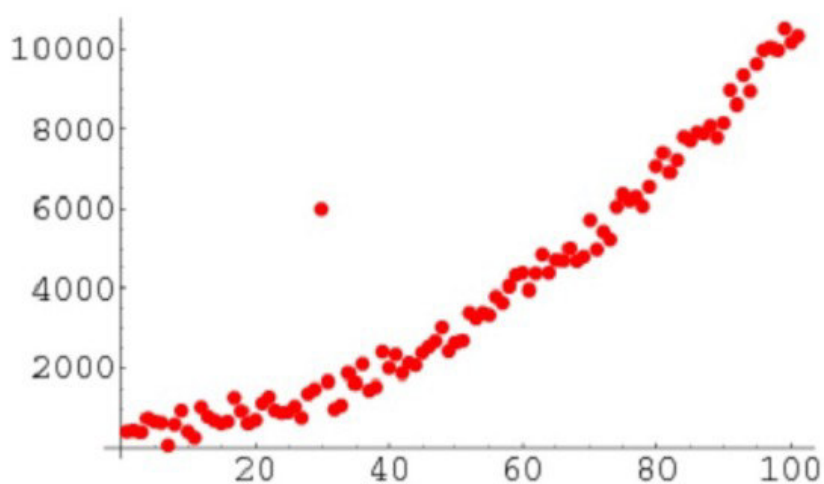
Полноценную версию блокнота Google Colab в pdf формате можно посмотреть по QR коду:

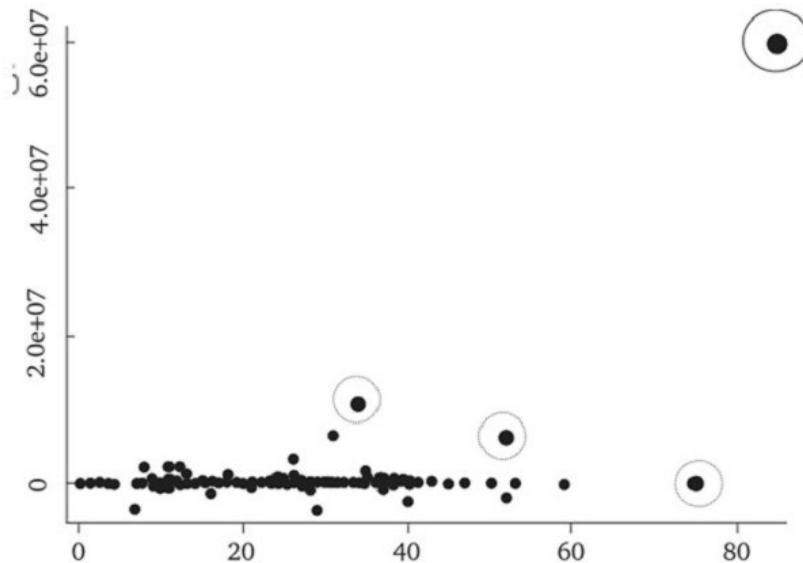


ПОИСК АНОМАЛИЙ И ВЫБРОСОВ.

В статистике, **выбросы** - это точки данных, которые не принадлежат определенной популяции. Это ненормальное наблюдение, которое находится далеко от других ценностей. Выделение - это наблюдение, которое отличается от хорошо структурированных данных. Например, вы можете ясно увидеть выбросы в этом списке: [20, 24, 22, 19, 29, 18, **4300**, 30, 18]

Это легко идентифицировать, когда наблюдения представляют собой просто набор чисел, и он одномерный, но когда у вас есть тысячи наблюдений или многомерных измерений, вам понадобятся более умные способы обнаружения этих значений.





Цели обнаружения выбросов.

Обнаружение выбросов или аномалий является одной из основных проблем интеллектуального анализа данных. Появляющееся расширение и продолжающийся рост данных и распространение устройств IoT заставляют нас переосмыслить подход к аномалиям и варианты использования, которые могут быть построены с учетом этих аномалий.

Обнаружение аномалий в данных сердечбиения может помочь в прогнозировании сердечных заболеваний. Аномалии в схемах движения могут помочь в прогнозировании аварий. Его также можно использовать для выявления узких мест в сетевой инфраструктуре и трафика между серверами. Следовательно, варианты использования и решения, основанные на обнаружении аномалий, безграничны.

Еще одна причина, по которой нам необходимо обнаруживать аномалии, заключается в том, что при подготовке наборов данных для моделей машинного обучения действительно важно обнаружить все выбросы и либо избавиться от них, либо проанализировать их, чтобы понять, почему они были у вас в первую очередь.

Бизнес:

- Обнаружение мошеннических транзакций банковских операций
- Обнаружение угроз
- Обнаружение нестандартных игроков на бирже. Обнаружение неполадок в механизмах по показаниям датчиков

- Медицинская диагностика
- Сейсмология

В технике:

- Подготовка данных
- Улучшение точности моделей
- Понимание природы данных

Виды выбросов:

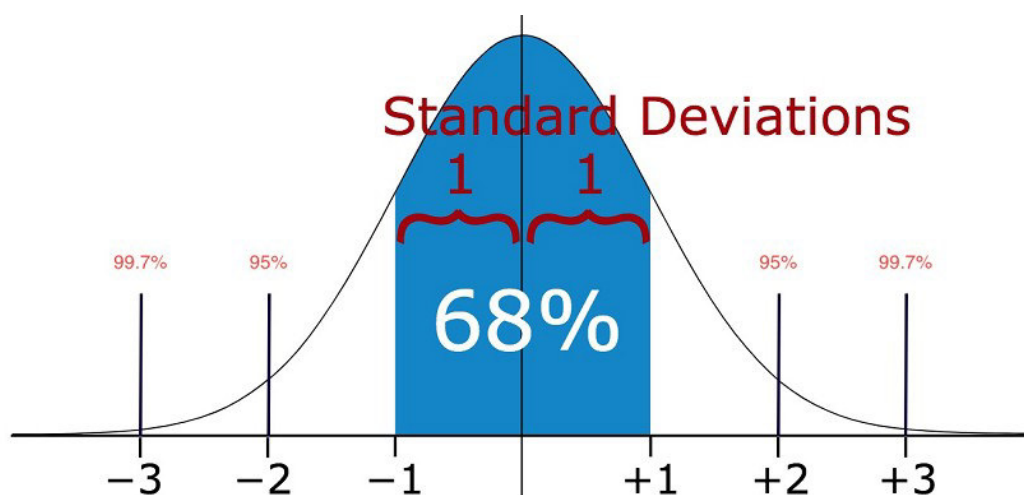
- Одномерные
- Многомерные выбросы
- Точечные выбросы
- Контекстуальные выбросы
- Коллективные выбросы

Методы обнаружения выбросов.

Использование стандартного отклонения.

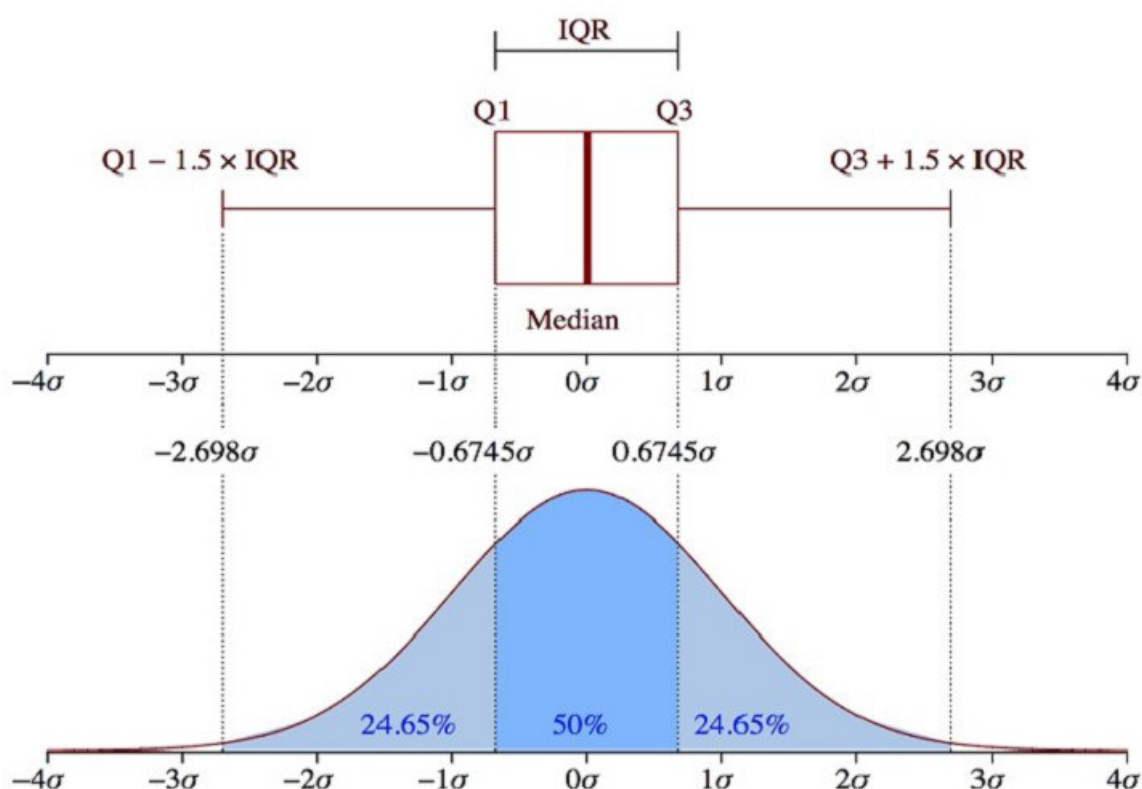
- 68% значений данных находятся в пределах одного стандартного отклонения от среднего
- 95% находятся в пределах двух стандартных отклонений
- 99,7% лежат в пределах трех стандартных отклонений

Поэтому, если у вас есть какая-либо точка данных, которая более чем в 3 раза превышает стандартное отклонение, то эти точки, скорее всего, будут аномальными или выбросами.



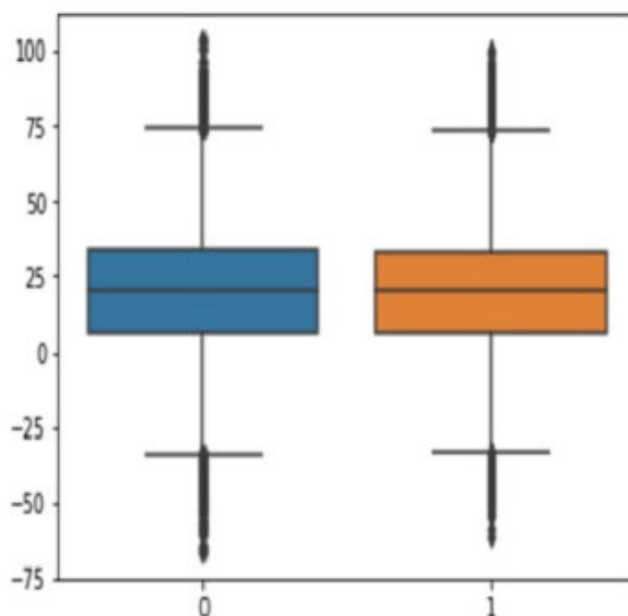
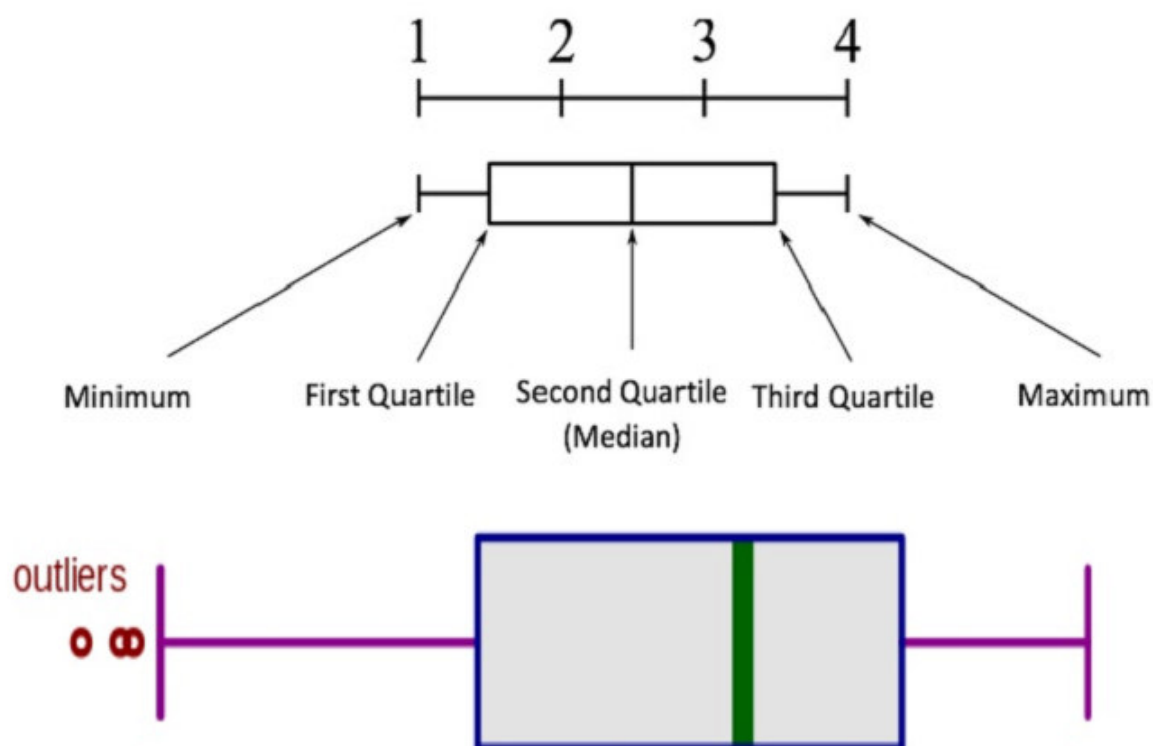
Использование межквартильного интервала.

Межквартильный интервал (IQR) — это разница между 1-м и 3-м квартилями, т.е. между 25-м и 75-м процентилями. В него входят центральные 50% наблюдений в упорядоченном наборе, где 25% наблюдений находятся ниже центральной точки и 25% — выше.



Межквартильный интервал (IQR) важен, потому что он используется для определения выбросов. Это разница между третьим квартилем и первым квартилем ($IQR = Q3 - Q1$). Выбросы в этом случае определяются как

наблюдения, которые ниже ($Q1 - 1,5 \times IQR$) или нижний усик или выше ($Q3 + 1,5 \times IQR$) или верхний усик.



Графики представляют собой графическое изображение числовых данных через их квантили. Это очень простой, но эффективный способ визуализации выбросов. Думайте о нижнем и верхнем усиках как о границах распределения данных. Любые точки данных, которые

показывают выше или ниже усов, могут считаться выбросами или аномальными.

Выполнение поиска аномалий и выбросов в среде Google Colab.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set()
```

Рассмотрим простой пример. Создадим переменную *x* с некоторыми числами. Выполним поиск аномалий с помощью рассмотренных выше методов.

```
x = np.array([4, 4, 3, 3, 3.5, 4, 4.5, 3, 3.5, 5, 5, 4, 5, 3.5, 5, 4, 3.5, 4, 3.5, 5, 4, 4,
4, 4, 4])
print(x)
```

```
[4.  4.  3.  3.  3.5 4.  4.5 3.  3.5 5.  5.  4.  5.  3.5 5.  4.  3.5 4.
 3.5 5.  4.  4.  4.  4. ]
```

Вычислим 1 и 3 квартили:

```
Q1 = np.quantile(x, 0.25)
Q3 = np.quantile(x, 0.75)
print('1 квартиль Q1 = ', Q1)
print('3 квартиль Q3 = ', Q3)
```

```
1 квартиль Q1 = 3.5
3 квартиль Q3 = 4.0
```

Вычислим межквартильный интервал (IQR):

```
IQR = Q3 - Q1
print('IQR = ', IQR)
```

```
IQR = 0.5
```

Вычисляем нижнюю ($Q1 - 1,5 \times IQR$) и верхнюю ($Q3 + 1,5 \times IQR$) границы:

```
low = Q1 - 1.5 * IQR
high = Q3 + 1.5 * IQR
print('нижняя граница ', low)
print('верхняя граница ', high)
```

```
нижняя граница 2.75
верхняя граница 4.75
```

Определим выбросы в данных:

```
print(x[x < low])  
print(x[x > high])
```

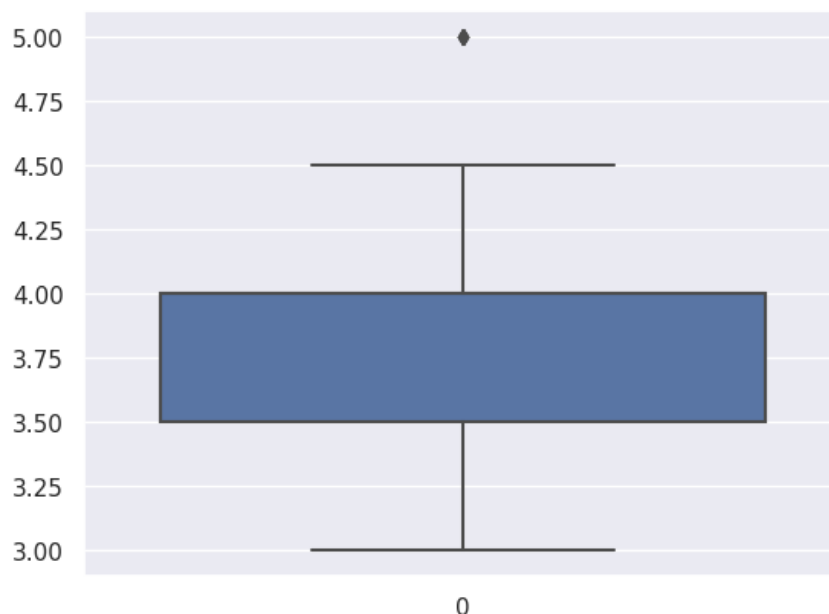
```
[]
```

```
[5. 5. 5. 5. 5.]
```

Как видно из результатов вывода, ниже значения **Q1 - 1,5x IQR** нет данных, т.е. отсутствую выбросы, а выше значения **Q1 + 1,5x IQR** попали пятерки. Значит в данном наборе данных аномалиями являются пятерки.

Построим ящик с усами (BoxPlot):

```
sns.boxplot(x);
```



На графике видно, что присутствует один выброс – 5.

Работа с реальными данными.

Набор данных о жилье в Бостоне.

Набор данных о жилье в Бостоне составлен на основе информации, собранной Службой переписи населения США о жилье в районе Бостона, штат Массачусетс.

CRIM - уровень преступности на душу населения в разбивке по городам

ZN - доля жилой земли, зонированной на участки площадью более 25 000 кв. футов.

INDUS - доля акров, не связанных с розничной торговлей, на город.

CHAS - Фиктивная переменная Чарльз-Ривер (1, если тракт ограничивает реку; 0 в противном случае)

NOX - концентрация оксидов азота (частей на 10 миллионов)

RM - среднее количество комнат в жилом помещении

AGE - доля квартир, занятых владельцами, построенных до 1940 года

DIS - взвешенные расстояния до пяти Бостонских центров занятости

RAD - индекс доступности к радиальным магистралям

TAX - ставка налога на недвижимость с полной стоимостью за 10 000 долларов США

PTRATIO - соотношение учеников и учителей в разбивке по городам

B - $1000(B_k - 0,63)^2$, где B_k - доля чернокожих по городам

LSTAT - % более низкий статус населения

MEDV - Средняя стоимость домов, занятых владельцами, в 1000 долларов

Файл можно загрузить по **ссылке**:

<https://drive.google.com/file/d/1EJ9ev8HVRgiZR3awQzdUp2d4jXprwz14/view?usp=sharing>

или по **QR коду**:



Загрузка данных:

```
data = pd.read_csv('/content/sample_data/boston_housing (3).csv')
```

Просмотр первых 5 строк таблицы:

```
data.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2

Вычислим сводные показатели:
data.describe()

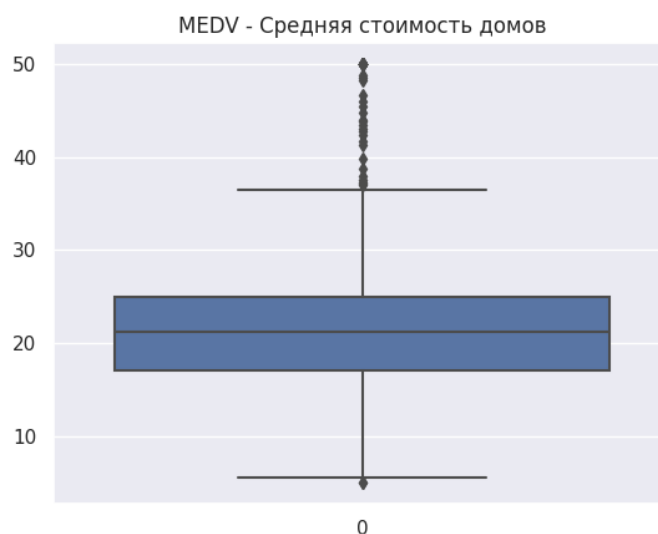
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTR
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.45
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.16
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.60
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.40
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.05
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.20
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.00

Скопируем все значения столбца 'MEDV' в отдельную переменную:
medv = np.array(data['MEDV'].values)

Пострим ящик с усами (Boxplot) для выявления выбросов:

```
sns.boxplot(medv);  
plt.title('MEDV - Средняя стоимость домов')
```

```
Text(0.5, 1.0, 'MEDV - Средняя стоимость домов')
```



Аналогичным образом, проверим остальные столбцы:

```
rm = np.array(data['RM'].values)
```

```
sns.boxplot(rm);
```

```
plt.title('RM - среднее количество комнат в жилом помещении')
```

```
Text(0.5, 1.0, 'RM - среднее количество комнат в жилом помещении')
```



```
zn = np.array(data['ZN'].values)
```

```
sns.boxplot(zn);
```

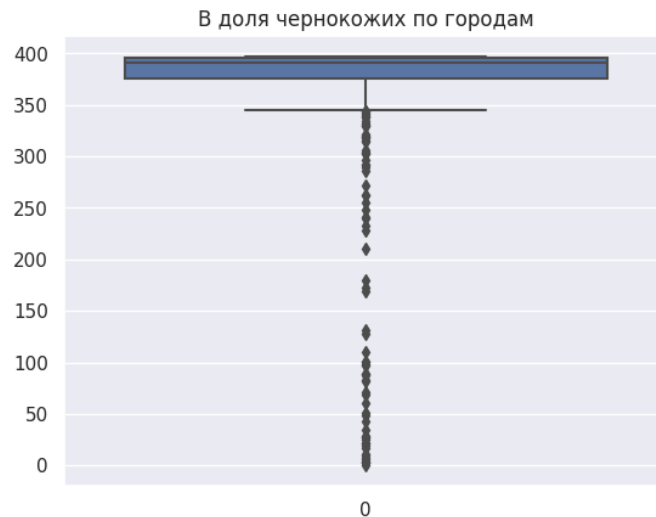
```
plt.title('ZN - доля жилой земли')
```

```
Text(0.5, 1.0, 'ZN - доля жилой земли')
```



```
b = np.array(data['B'].values)
sns.boxplot(b);
plt.title('В доля чернокожих по городам')
```

```
Text(0.5, 1.0, 'В доля чернокожих по городам')
```



```
tax = np.array(data['TAX'].values)
sns.boxplot(tax);
plt.title('TAX - ставка налога на недвижимость')
```

```
Text(0.5, 1.0, 'TAX - ставка налога на недвижимость')
```



Задания.

- 1) Выполнить поиск аномалий для столбцов 'AGE', 'RAD', 'PTRATIO'.
- 2) Скачать **по ссылке** набор данных:

<https://drive.google.com/file/d/1FQy2FVNlph8dy889Gvf8yVKMNC3XLUoH/view?usp=sharing>

или по QR коду:



Выполнить поиск аномалий и выбросов для данных своего варианта.
Номер варианта соответствует номеру в списке группы.

Полноценную версию блокнота Google Colab в pdf формате можно
посмотреть по QR коду:



ЗАКЛЮЧЕНИЕ.

Объем данных в мире растет с невероятной скоростью. По прогнозам специалистов, к 2025 году объем данных будет увеличиваться на 60% каждый год. Это связано с ростом числа устройств, генерирующих данные, таких как смартфоны, датчики, камеры и другие устройства. Кроме того, развитие технологий, таких как искусственный интеллект и машинное обучение, также способствует увеличению объема данных. На сегодняшний день накоплено огромное количество данных.

Математические методы обработки информации - это методы, которые используются для анализа и обработки данных. Они включают в себя различные виды статистики, математический анализ, теорию вероятностей и другие дисциплины.

Машинное обучение использует алгоритмы для обучения на данных, а нейронные сети имитируют работу мозга, используя слои нейронов. Оба подхода используются для обработки больших объемов данных и могут быть использованы для решения различных задач, таких как распознавание образов, классификация, прогнозирование и т.д. Современные методы обработки данных включают машинное обучение, глубокое обучение, обработку естественного языка, компьютерное зрение и анализ данных. Машинное обучение используется для предсказания будущих значений на основе исторических данных, а глубокое обучение позволяет создавать модели, которые могут обучаться на больших объемах данных. Обработка естественного языка используется для анализа текста и ответа на вопросы, а компьютерное зрение - для обработки изображений и видео.

Искусственный интеллект (ИИ) является одной из самых перспективных областей развития технологий. С каждым годом ИИ становится все более распространенным и используется в различных сферах жизни. Одной из главных задач развития ИИ является создание систем, которые могут обрабатывать и анализировать большие объемы данных. Также перспективным направлением развития ИИ является использование нейросетей. Нейросети позволяют создавать модели, способные обучаться на основе данных и принимать решения на основе этих данных. Нейросети могут использоваться для распознавания образов, классификации данных, прогнозирования и других задач. Кроме того, развитие ИИ может привести к созданию систем, которые смогут заменить человека в некоторых задачах. Например, уже сейчас существуют системы, которые могут выполнять работу юристов, врачей и других специалистов. В будущем такие системы могут стать еще более совершенными и заменить человека во многих сферах деятельности.

ЛИТЕРАТУРА

Рашка, Себастьян, Мирджалили, Вахид. Python и машинное обучение: машинное и глубокое обучение с использованием Python, scikit-learn и TensorFlow 2 3-е изд. [Текст] / Рашка, Себастьян, Мирджалили, Вахид; СПб. : ООО "Диалектика", 2020. - 848 с.: ил. – Парал. тит. англ. – ISBN 978-5-907203-57-0 (в пер.).

Плас Дж. Вандер. Python для сложных задач: наука о данных и машинное обучение [Текст] / Плас Дж. Вандер; СПб.: Питер, 2018. – 576 с.: ил. – (Серия «Бестселлеры O'Reilly»). – ISBN 978-5-496-03068-7 (в пер.).

Грас Д. Data Science. Наука о данных с нуля. [Текст] / Грас Д. – 2-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2021. - 416 с.: ил. – ISBN 978-5-9775-6731-2 (в пер.).

База научных статей, переводов, таблиц, графиков, иллюстраций, примеров кода по темам - анализ данных, машинное обучение, нейронные сети, искусственный интеллект [Электронный ресурс] – Режим доступа: <https://machinelearningmastery.ru/>