

С.Ю. Добдин, А.В. Скрипаль

**Введение в науку о данных  
с использованием языка Python  
(Введение в Data Science)**

Учебник по программированию

«Саратов»

2022

УДК [616-072.7: 612.13]

ББК 53.4 я 73

C45

**Добдин С.Ю., Скрипаль А.В.**

Введение в науку о данных с использованием языка Python (Введение в Data Science): Учебник по программированию будет полезен студентам, аспирантам и преподавателям инженерных и естественно-научных специальностей вузов, интересующихся программированием на языке Python, анализом данных и созданием алгоритмов машинного обучения. Изд. 1-е. – Саратов. 2022. – Н с.: ил.

**ISBN**

В учебнике подробно изложены теоретические основы программирования на языке Python. Рассмотрены основные приложения для анализа данных, которые очень часто используются при подготовке данных для алгоритмов машинного обучения. Представлены основные методы и подходы для анализа исследовательских данных, в том числе основанных на статистической оценке. Для визуализации результатов используются специальные графические пакеты. Основное содержание учебника посвящено предобработке данных и возможности создания новых признаков. Отдельно приведены способы обработки числовых категориальных данных. В качестве базовых алгоритмов машинного обучения рассмотрены модели регрессии, классификации и кластеризации с использование библиотеки sklearn. Обсуждаются вопросы оптимизации и настройки гиперпараметров модели, позволяющих получить наилучшие результаты. Подробно представлены способы сбора и хранения информации с использованием языка Python. Отдельное внимание уделено вопросам парсинга данных в сети интернет. В заключительной части подробно рассказано о современной системе контроля версий git.

Рекомендует к печати:

Кафедра системного анализа и автоматического управления  
Саратовского национального исследовательского  
государственного университета имени Н.Г. Чернышевского  
доктор физико – математических наук **Сысоев Илья Вячеславович**

© Добдин С.Ю., Скрипаль А.В. 2022

# Оглавление

## Предисловие

### 1. Основы Python и анализа данных

- 1.1. Целые числа, ввод-вывод, простые операции со строками
  - 1.2. вещественные числа и операции над ними
  - 1.3. Условный оператор, цикл while и цикл for
  - 1.4. Функции и рекурсия
  - 1.5. Кортежи, списки и их сортировка
  - 1.6. Множества и словари
  - 1.7. Введение в объектно-ориентированное программирование (ООП)
- Задания для самостоятельной работы

### 2. Предобработка исследовательских данных

- 2.1. Подключение и работа с библиотекой numpy
  - 2.2. Подключение и работа с библиотекой pandas
  - 2.3. Срезы данных и логическое индексирование
  - 2.4. Предобработка данных: пропуски, некорректные данные и дубликаты
  - 2.5. Изменение типов данных
- Задания для самостоятельной работы

### 3. Визуализация данных

- 3.1. Библиотека matplotlib
  - 3.2. Точечная и линейная диаграмма
  - 3.3. Построение гистограмм и столбчатых диаграмм
  - 3.4. Построение boxplot
- Задания для самостоятельной работы

### 4. Статистический анализ данных

- 4.1. Среднее значение, медиана, мода и диапазон
  - 4.2. Элементы теории вероятности: дисперсия и стандартное отклонение
  - 4.3. Группировка и агрегирование данных
  - 4.4. Взаимосвязь данных
  - 4.5. Проверка гипотезы
- Задания для самостоятельной работы

## **5. Обработка числовых и категориальных признаков**

- 5.1. Масштабирование признаков
- 5.2. Стандартизация признаков
- 5.3. Нормализация данных
- 5.4. Генерация полиномиальных признаков
- 5.5. Преобразование признаков
- 5.6. Анализ выбросов
- 5.7. Дискретизация признаков
- 5.8. Работа с пропущенными данными
- 5.9. Кодирование номинальных категориальных признаков
- 5.10. Кодирование порядковых категориальных признаков
- 5.11. Кодирование словарей признаков
- 5.12. Категориальные переменные

## **6. Исследовательский анализ данных и машинное обучение**

- 6.1. Проверка мультиколлинеарности и отбор признаков
- 6.2. Регрессионный анализ
- 6.3. Классификация
- 6.4. Деревья решений
- 6.5. Нейронные сети
- 6.6. Кластеризация

Задания для самостоятельной работы

## **7. Сбор и хранение данных**

- 7.1. Работа с файлами формата csv и excel
- 7.2. Работа с файлами формата txt
- 7.3. Чтение и запись JSON-файлов
- 7.4. Чтение html-файлов из интернета, парсинг данных, API
- 7.5. Чтение из базы данных SQL, запись в базу данных SQL
- 7.6. Операторы для работы с базой данных в библиотеке sqlite3
- 7.7. Представление результатов в Jupyter Notebook – IPython
- 7.8. Редактор кода Visual Studio Code
- 7.9. Система контроля версий git

Задания для самостоятельной работы

**Рекомендуемая литература**

## **Предисловие**

Язык Python (на русском произносится как Пайтон или Питон) был разработан в 1991 году программистом Гвидо Ван Россумом (нидерл. Guido van Rossum). Своим названием язык обязан известному шоу «Летающий цирк Монти Пайтона». Основная цель разработчиков языка – реализовать модель простого и доступного языка программирования любому пользователю. На сегодняшний день язык входит в ряд наиболее используемых языков программирования в мире. С помощью данного языка очень часто решают задачи по анализу и обработке больших данных (Big Data). Python является кроссплатформенным языком программирования. Нет большой разницы в какой операционной системе была написана программа.

Для глубокого изучения языка рекомендуется ознакомиться с дополнительной литературой посвящённой языку Python. Материал, изложенный в данных книгах, позволит получить больше информации об основах программирования на языке Python, а также дополнительно решить множество практических задач по программированию.

Для написания программ на языке программирования Python потребуется предварительная установка интерпретатора (программа для компиляции написанных программ в машинный код) и среды для программирования. На начальном этапе можно воспользоваться средой программирования, которую можно скачать вместе с интерпретатором по ссылке с официального сайта разработчика:

<https://www.python.org/downloads/>

Дополнительное рекомендуемое ПО – среда программирования JetBrains PyCharm (Community):

<https://www.jetbrains.com/pycharm/download/>

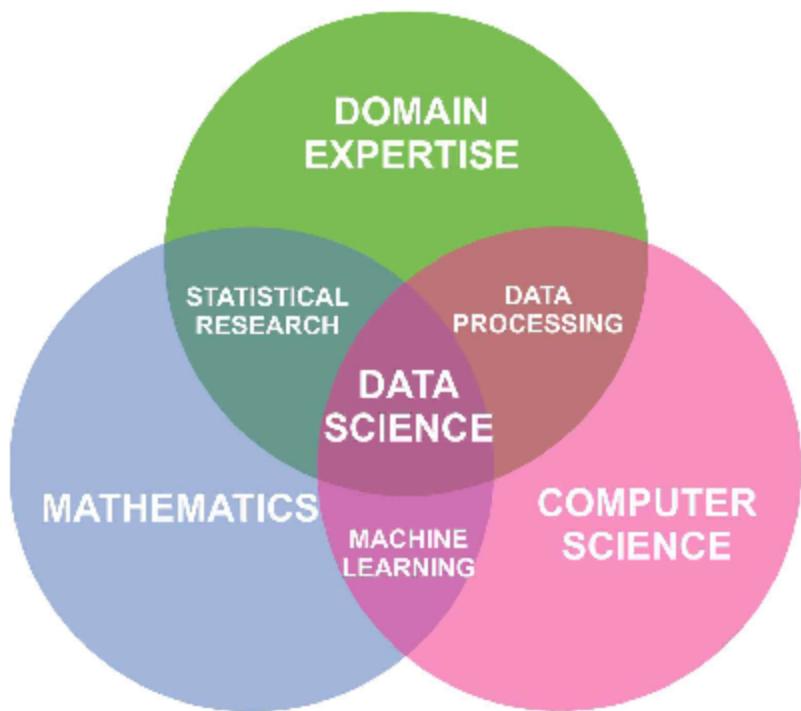
Среда программирования для начинающих – Thonny Python IDE for beginners:

<https://github.com/thonny/thonny/releases/download/>

Начинающим пользователям рекомендуется использовать среду программирования Thonny Python IDE. В среде программирования упрощён процесс написания и отладки программы. Кроме того, для среды существует дополнительное расширение, которое можно установить заранее. Расширение позволяет создавать приложения для работы с аппаратной частью микрокомпьютеров Raspberry Pi.

Для лучшего усвоения процесса обучения программирования на python рекомендуется следующая последовательность: изучение теоретического материала, использование дополнительных источников информации (книги и обучающее видео), выполнение практических задач. Основным ключом к эффективному изучению материала является решение практических задач.

После освоения ключевых понятий языка Python будут подробно рассмотрены некоторые приложения языка (библиотеки). В частности, рассмотрена несколько библиотек для графического представления данных. Визуализация является важным этапом анализа данных, позволяющим выявлять закономерности в данных. Данный этап тесно связан с аналитикой данных. Для предварительной подготовки данных используется библиотека pandas и numpy. На основе данных приложений можно реализовать статистический анализ данных и подготовку данных. Именно эти библиотеки используются на этапе исследовательского анализа данных (англ. exploratory data analysis, EDA). Анализ и подготовку данных можно назвать заключительным этапом перед созданием алгоритмов машинного обучения. Совокупность всех приведённых этапов и является основой для науки о данных (англ. Data Science, DS).

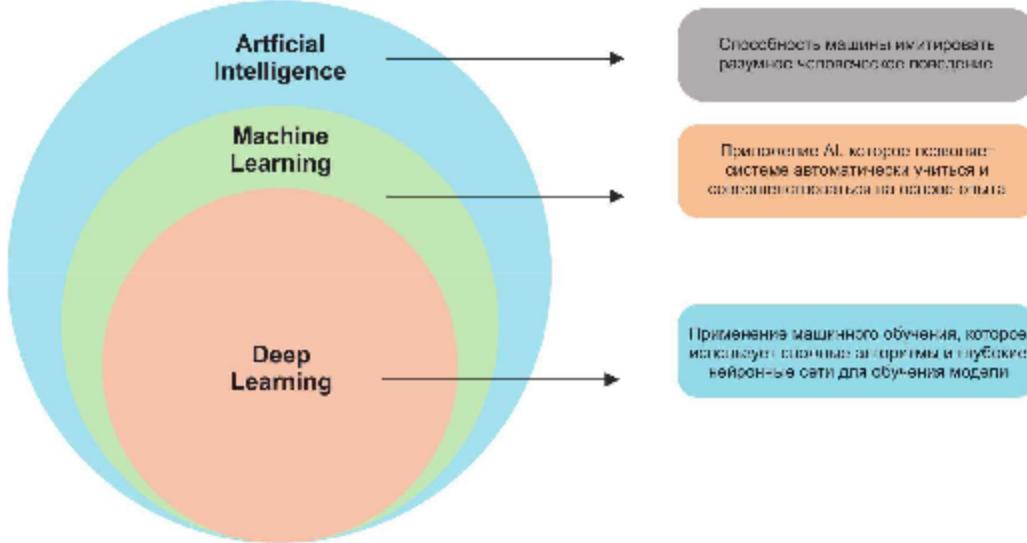


[Ист.: Palmer S. Data Science for C-Suite, New York: Digital Living Press, 2015]

Для полноценной реализации исследовательского цикла необходимо обладать тремя основными базовыми составляющими анализа данных: знание языка программирования (например, Python), знание основных приложений для анализа данных, знание основных приложений для создания алгоритмов машинного обучения. Эффективный процесс изучения дисциплины будет зависеть от последовательного изучения и закрепления всех указанных этапов.

Ещё одним важным понятием при изучении алгоритмов машинного обучения является понятие искусственный интеллект (англ. Artificial Intelligence, AI). Данный термин включает научные теории и технологии, которые позволяют реализовать компьютерную программу, имитирующую реальный интеллект. Основными составляющими искусственного интеллекта является: алгоритмы машинного обучения (англ. Machine Learning, ML) и

алгоритмы глубокого обучения (англ. Deep Learning, DL). В данном пособии будут представлены примеры некоторых алгоритмов машинного обучения.



Код некоторых программ и датасеты данного методического пособия можно скачать на следующих сайтах в сети интернет:

[https://github.com/mculab64/data\\_science/](https://github.com/mculab64/data_science/)  
[https://elteha.ru/data\\_science/data.zip](https://elteha.ru/data_science/data.zip)

Основным критерием успешного изучения теоретического материала является применение знаний на практике. В сети существует множество ресурсов, которые позволяют писать алгоритмы машинного обучения и участвовать в соревнованиях разного уровня. Кроме того, некоторые ресурсы, например, kaggle.com, позволяют скачать готовые датасеты различных данных. Приведём несколько полезных ссылок:

<https://www.kaggle.com/>

<https://colab.research.google.com/notebooks/intro.ipynb#recent=true>

<https://ods.ai/competitions>

<https://boosters.pro/>

<https://mlbootcamp.ru/ru/main/>

На сайте <https://stepik.org> или <https://www.coursera.org> можно записаться на бесплатные обучающие курсы, где можно получить дополнительные знания о библиотеках Python, алгоритмах машинного обучения и алгоритмах глубокого обучения. Изучить базы данных (реляционные базы данных) и язык структурированных запросов SQL. Подробно разобрать особенности работы с большими данными (big data).

# 1. Основы Python и анализа данных



## 1.1. Целые числа, ввод-вывод, простые операции

В данном разделе, посвящённом программированию на языке Python, будут продемонстрированы листинги программ, написанные и протестированные в программе Thonny Python IDE. Установите данную среду на компьютер и запустите. Создайте и сохраните новый файл (обязательно убедитесь, что файл имеет расширение \*.py).

Программа на языке Python – это обычный текстовый файл с набором команд. Рассмотрим процесс написания и запуска новых программ. Добавим в новый файл одну команду **print()**, которая предназначена для вывода информации. Пусть наша первая программа выводит на экран текст ‘Hello World!’. Для запуска скрипта нажмите клавишу F5 или нажмите на соответствующую кнопку на панели инструментов.

```
print('Hello World!')
```

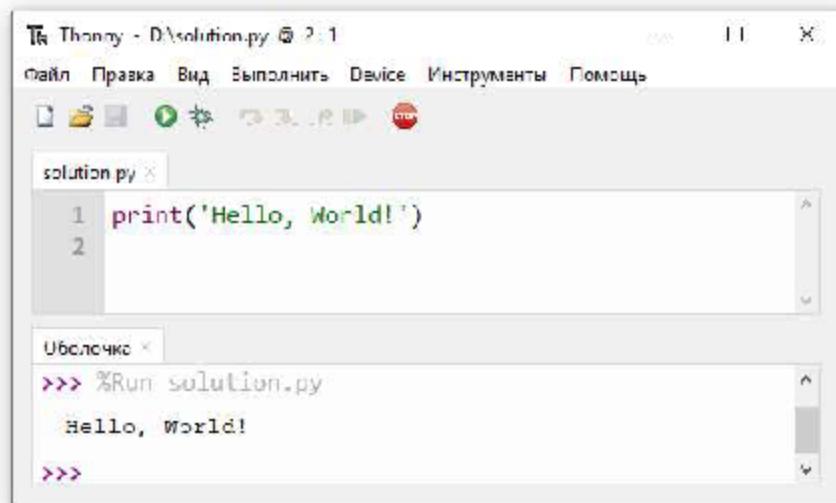


Рис.1.1. Среда программирования Thonny Python IDE

В соседнем окне можно наблюдать результат выполнения скрипта. Теперь необходимо ознакомиться с правилами обработки целочисленных значений. Из математики известно, что под данное определение попадают

следующий ряд чисел: 0, -1, 1, -2, 2 … -N, N. Посмотрим примеры программ на Python для обработки целых чисел.

---

```
print(3 + 2)                                # (1)
print('3' + '2')                             # (2)
print('3 + 2 =', 3 + 2)                      # (3)
print('3 + 2 =', 3 + 2, sep=' ')             # (4)
print(4, 3, 2, 1, sep=' + ', end='\n') # (5)
```

---

Рассмотрим подробнее приведённые примеры. Первое, что хотелось отметить это появление новой конструкции, которая начинается с символа решётки `#`. Так на языке Python принято обозначать комментарии. Размещение комментария является правилом хорошего тона при программировании. Настоятельно рекомендуется всегда использовать в своей программе комментарии. Теперь о самом коде. Пример (1) демонстрирует вывод в консоль суммы двух натуральных чисел. Пример (2) является конкатенацией (склеиванием) двух строк. Обратите внимание, что строка может быть выделена двойными или одинарными кавычками. В примере 4 используется новая для нас конструкция. Она называется именованный параметр `sep` (separator или разделитель). Его можно использовать, когда при выводе результат необходимо отделить некоторым символом. В данном примере таким разделителем является пробел. В примере 5 используется ещё одна неизвестная для нас конструкция. Она называется именованный параметр `end` (конец). Параметр определяет конец строки. В нашем примере в качестве окончания строки используется переход на новую строку.

Создайте новый файл, запишите приведённые примеры и посмотрите на результат работы данной программы. Результаты сравните с ответом.

---

```
>>> %Run example_01.py
```

```
5
32
3 + 2 = 5
3 + 2 = 5
4 + 3 + 2 + 1
```

---

В приведённом выше примере мы использовали ввод целых чисел без создания переменных. **Переменная** – это контейнер для хранения определённых значений данных. Сейчас мы говорим только о целых числах. Позднее познакомимся с другими типами данных. Рассмотрим пример использования переменных при решении простых задач. Следующий листинг позволяет рассчитать объём пирамиды.

---

```
import math # Подключение математического модуля
# Данна правильная треугольная пирамида
```

```

a = 1 # 1 сторона основания
b = 1 # 2 сторона основания
c = 1 # 3 сторона основания

h = math.sqrt(3) # Высота пирамиды
S = (a * a * math.sqrt(3)) / 4 # Площадь основания
V = 1/3 * S * h # Объём фигуры

print('Объём пирамиды = {:.2f}'.format(V))

```

---

Полученный результаты сравните со своим ответом.

---

```
>>> %Run example_02.py
```

Объём пирамиды = 0.25

---

Пример example\_02.py уже более сложная программа. В ней показано как можно подключить дополнительные модули к программе. В нашем случае использовалась библиотека **math** для нахождения квадратного корня числа. Приведён пример создания переменных через знак присваивания ‘=’ и составления выражений. Для вывода результата программы используется метод **format** с точностью до двух знаков после запятой. Данный метод будет полезен при написании программ и правильного форматирования выводимой информации.

```

print('|{: <10.2f}|'.format(123.0)) # Выравнивание по левому краю
print('|{: >10.1f}|'.format(1234.0)) # Выравнивание по правому краю
print('|{: ^10.1%}|'.format(0.12)) # Выравнивание по центру

```

---

Результат программы.

---

```
>>> %Run example_03.py
```

123.00
1234.0
12.0%

Язык Python, как и многие языки программирования, имеет ряд арифметических операций, которые можно применить для обработки значений. С некоторыми мы уже знакомы по предыдущим программам. Ряд других арифметических операций представлены в таблице.

Знак	Операция	Значение 1	Значение 2	Результат
+	Сложение	5	4	9
-	Вычитание	5	4	1
*	Умножение	5	4	20
//	Целочисленное деление	15	4	3
%	Остаток от деления	14	4	2

Теперь рассмотрим команду для ввода данных в программу. Для ввода данных в программу используется команда **input()**. Следующий листинг для расчёта площади прямоугольника демонстрирует использование этого метода.

```
# Программа для нахождения площади прямоугольника
a = int(input()) # сторона а
b = int(input()) # сторона b

S = a * b          # Площадь прямоугольника

print('Площадь прямоугольника = {:.2f}'.format(S))
```

Результат программы.

```
>>> %Run example_04.py
```

```
1
2
Площадь прямоугольника = 2.00
```

В данной программе используется неизвестная нам конструкция **int(input())**. Функция **int()** принудительно сообщает программе, что вводимые данные будут целыми числами. Вводимые данные могут быть типа **float**, для этого нужно было указать вместо **int()** функцию **float()**. Ну и, наконец, ещё один распространённый тип данных, который называется **string** (строка). Ранее мы рассмотрели данный тип данных на примере операции конкатенации строк. Функция **str()** принудительно сообщает программе, что данные будут строкой.

При работе со строками на практике очень часто используют срезы. Срез – это способ извлечения части символов или некоторых символов из строки. Создание среза приводит к созданию новой строки и к сохранению структуры старой строки. Каждый символ строки имеет свой индекс. Используется два типа нумерации:  $[0, 1, 2 \dots N]$  и  $[-N \dots -3, -2, -1]$ . Рассмотрим пример получения среза. Проанализируйте результат работы программы самостоятельно.

```
string = 'abcdefghijkl'

s_1 = string[0]
s_2 = string[0:5]
s_3 = string[3:]
s_4 = string[-1]
s_5 = string[-5:-2]
s_6 = string[:-5]
s_7 = string[::-1]
print(s_1, s_2, s_3, s_4, s_5, s_6, s_7, sep='\n')
```

Строку string можно представить следующим образом:

	<b>a</b>	<b>b</b>	<b>c</b>	<b>d</b>	<b>e</b>	<b>f</b>	<b>g</b>	<b>h</b>	<b>i</b>
[0...N]	0	1	2	3	4	5	6	7	8
[-N...-1]	-9	-8	-7	-6	-5	-4	-3	-2	-1

Методы для работы строками. Ниже представлен пример работы с методом **find**, **replace** и **count**. Первый метод позволяет найти индекс первого вхождения строки в подстроку. Второй метод осуществляет поиск вхождения подстроки в строке и её замену на новое значение. Метод **count** считает количество вхождений подстроки в строке.

```
st = '123123456'

s_1 = st.find('123', 3)          # Индекс первого вхождения от 3-го индекса
s_2 = st.replace('456', '123', 1) # Замена '456' на '123' один раз
s_3 = st.count('123')           # Подсчёт всех вхождений '123'

s_4 = len(st)                   # Нахождение длины строки

print(s_1, s_2, s_3, s_4, sep='\n')
```

Существует также метод **rfind**, который осуществляет поиск индекса первого вхождения, начиная поиск с другой стороны строки. Строковая функция **len()** используется в данном примере для нахождения длины строки. Позднее она нам пригодится при организации циклов, которые позволяют последовательно перебирать все символы строки от начала до конца.

## 1.2. Вещественные числа и операции над ними

Если число можно представить в виде  $N = M \cdot q^p$ , где  $M$  – мантисса,  $q$  – основание, а  $p$  – порядок, то такое число называется вещественным. На самом деле, вещественные числа отличаются от натуральных тем, что они могут быть представлены с плавающей точкой. Вещественных чисел значительно больше, чем натуральных, поэтому для хранения этих чисел в Python требуется ограничение точности. Рассмотрим небольшой пример сложения двух чисел.

```
a = 0.1 # число a
b = 0.2 # число b

print(a + b)
```

Результат программы.

```
>>> %Run example_05.py
```

```
0.3000000000000004
```

Результат выполнения программы показал, что в определённом знаке появилось значение, которого быть не должно. Так устроен язык программирования Python и метод хранения чисел в памяти вычислительной машины. При решении задач требующих высокую точность нужно учитывать данную особенность, чтобы исключить дополнительную погрешность вычислений.

Иногда для удобства обработки данных используются различные функции для округления. Ниже приведён пример использования этих функций.

---

```
import math          # Подключение математического модуля

print(int(2.5))    # Округление в сторону нуля
print(round(2.5))  # Округляет до ближайшего целого
print(math.floor(2.5)) # Округление в меньшую сторону
print(math.ceil(2.5)) # Округление в большую сторону
```

---

В библиотеке **math** можно найти функцию для округления **trunc()**, которая работает аналогично **int()**. Библиотека **math** содержит множество других математических функций для работы с вещественными числами.

### 1.3. Условный оператор и цикл while

В предыдущих разделах мы познакомились с двумя типами данных: целые числа и строки. Рассмотрим логический тип данных, который может принимать одно из двух возможных состояний: **True** (истина) и **False** (ложь). По аналогии с арифметическими операциями существуют логические операции. В таблице приведены наиболее распространённые из них.

Знак сравнения	Описание
<	меньше
>	больше
<=	меньше либо равно
>=	больше либо равно
==	равно
!=	не равно

Листинг для демонстрации логических типов данных.

---

```
a = 1 # значение a
b = 2 # значение b
c = 3 # значение c
d = 4 # значение d

result_1 = a <= b and c <= d
result_2 = a >= b or c >= d
```

---

```
print(result_1, result_2)
```

---

Результат программы.

---

```
>>> %Run example_05.py
```

```
True False
```

---

Наиболее часто логический тип данных используется в условных операторах. Условный оператор проверяет выполняется условие или нет. Для записи используется выражение **if** и последующий блок команд. Далее рассмотрим пример, в котором проверяется условие для введённого целого числа. Если введённое значение больше нуля, то выводится сообщение ‘Введённое число > 0’, в противном случае ‘Введённое число < 0’. Проанализируйте результат выполнения программы самостоятельно. Дополнительно используйте разные логические выражения для проверки условий оператора **if**.

---

```
num = int(input()) # Ввод целого числа  
  
if num > 0: # Если число num > 0, то выводим  
    print('Введённое число > 0')  
else: # Иначе выводим следующее  
    print('Введённое число < 0')
```

---

Условный оператор **if** допускает использование вложенных конструкций. При множественной проверке условий многократное использование оператора является не самым оптимальным решением. Если требуется проверить много условий, то в этом случае лучше использовать конструкцию **if - else** совместно с **elif** (иначе-если).

---

```
num = int(input()) # Ввод целого числа  
  
if num == 1: # Если число num = 1, то выводим  
    print('Один')  
elif num == 2: # Если число num = 2, то выводим  
    print('Два')  
else: # Иначе выводим следующее  
    print('Другое число')
```

---

Самостоятельно изучите приведённый код. Для эксперимента попробуйте добавить в эту программу дополнительные условия, используя конструкцию **elif**.

Следующую конструкцию, которую необходимо изучить для организации всевозможных циклов называется **while** (пока). При входе в цикл выполняется проверка условия **while**, далее выполняется блок команд и

возврат к проверке условия. Если условие не выполняется, то цикл завершается и выполняется следующий блок программы. Данная конструкция очень удобна для организации бесконечных циклов.

---

```
num = int(input())      # Ввод целого числа

while num != 0:          # Пока введённое число не равно нулю
    print('Вы ввели {}, программа продолжается'.format(num))
    num = int(input()) # Ввод нового числа

print('Вы ввели 0, программа завершена')
```

---

Рассмотрим ещё один пример использования конструкции **while** для организации циклов. В данном примере программа считывает введённое число  $N$ , а потом выводит последовательность чисел в одну строку от 0 до  $N$ . Попробуйте изменить программу так, чтобы значения выводились не в строку, а в столбец.

---

```
num = int(input())      # Ввод целого числа
out = 0

while num > 0:          # Пока число > нуля
    print(out, end=' ')
    out += 1
    num -= 1
```

---

Если возникла необходимость принудительного завершения цикла **while**, то можно воспользоваться оператором **break**. Его использование удобно в некоторых случаях, но зачастую лучше избегать его использования.

## 1.4. Кортежи, списки, их сортировка и цикл for

В этом разделе мы познакомимся с двумя новыми типами данных: кортеж и список. **Кортеж (tuple)** состоит из элементов произвольных типов и является неизменяемым. Кортеж выделяется круглыми скобками (). Кортеж может содержать внутри себя другие кортежи. **Список (list)** также состоит из элементов произвольных типов, но является изменяемым. Список выделяется квадратными скобками []. Рассмотрим пример создания кортежа и списка.

---

```
string = '12345'        # Ввод строки

tuplen = tuple(string) # Создание кортежа
listn = list(string)  # Создание списка

print(tuplen)           # Вывод кортежа
print(listn)            # Вывод списка

print(tuplen[0])         # Вывод элемента кортежа с индексом [0]
print(listn[4])          # Вывод элемента списка с индексом [4]
```

---

## Результат программы.

---

```
>>> %Run example_06.py

['1', '2', '3', '4', '5']
['1', '2', '3', '4', '5']
1
5
```

Список, как уже было сказано ранее, является изменяемой структурой. Рассмотрим некоторые методы для работы со списком и его отдельными элементами.

Метод	Функция
list.append(x)	Добавление элемента x к списку
list.extend(L)	Добавление списка L в конец существующего списка
list.insert(i, x)	Вставка на место i-го элемента значения x
list.remove(x)	Удаление первого элемента x в списке
list.pop([i])	Удаление i-го элемента списка
list.count(x)	Возврат количества элементов со значением x
list.sort([key=функция])	Сортирует список на основе функции
list.reverse()	Разворачивает список
list.clear()	Очищает список

Остановимся подробнее на методе `sort()`. Данный метод позволяет реализовать сортировку данных. Числа могут быть отсортированы в порядке возрастания, буквы согласно их порядку в алфавите. Использование lambda-функции в качестве параметра можно сортировать по разным столбцам списка.

---

```
# Определение списка целых чисел 3x3
listn = [[5, 3, 1],
          [4, 5, 8],
          [7, 2, 5]]

# Сортировка списка по 2 столбцу с инверсией
listn.sort(key=lambda i: i[1], reverse=True)

print(listn)           # Вывод нового списка
```

---

## Результат программы.

---

```
>>> %Run example_07.py
```

```
[[7, 2, 5], [5, 3, 1], [4, 5, 8]]
```

---

Ранее мы использовали команду `int()` для построчного добавления элементов в нашу программу. Задачу по вводу нескольких символов подряд можно упростить используя метод `split()`, который позволяет создать из строки набор элементов. Например, для создания списка из введённой строки целых чисел будет выглядеть следующим образом:

```
listn = list(map(int, input().split()))
```

---

В данном случае функция `map()` позволяет определить для каждого нового элемента из введённой строки тип данных `integer`. Существует и обратная команда для создания строки, которая называется `join()`.

```
listn = [1, 2, 3]
print(' '.join(map(str, listn)))
```

---

В последнем примере в качестве разделителя используется пробел. Теперь рассмотрим ещё одну конструкцию для реализации циклов, которая называется `for`. Цикл `for` позволяет поочерёдно перебирать элементы из чего-нибудь (`iterable`, `tuple` или `list`). Очень часто `for` используется совместно с функцией `range()`, которая позволяет генерировать элементы типа `iterable`.

```
print(list(range(10))) # Генерация списка [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

---

Рассмотрим несколько примеров использования цикла `for` для обработки данных. Перебор всех элементов списка:

```
apples = ['red', 'green', 'yellow']
for color in apples:
    print('Цвет яблока', color, sep=' ')
```

---

Результат программы.

```
>>> %Run example_07.py
```

```
Цвет яблока red
Цвет яблока green
Цвет яблока yellow
```

---

Создание нового списка, используя функцию `range()`:

```
listn = [] # Создание пустого списка
```

---

```
for i in range(1, 10):    # Цикл для i от 1 до 9
    listn.append(i**2)    # Добавления квадрата числа в список
print(listn)              # Вывод списка на экран
```

---

Результат программы.

```
>>> %Run example_07.py
[1, 4, 9, 16, 25, 36, 49, 64, 81]
```

---

Пример создания вложенных циклов. Вывод таблицы умножения на экран.

---

```
for i in range(1, 10):
    for j in range(1, 10):
        print('{: >2.0f}'.format(i*j), end=' ')
    print()
```

---

Результат программы.

```
>>> %Run example_08.py
1 2 3 4 5 6 7 8 9
2 4 6 8 10 12 14 16 18
3 6 9 12 15 18 21 24 27
4 8 12 16 20 24 28 32 36
5 10 15 20 25 30 35 40 45
6 12 18 24 30 36 42 48 54
7 14 21 28 35 42 49 56 63
8 16 24 32 40 48 56 64 72
9 18 27 36 45 54 63 72 81
```

---

В следующих разделах мы более подробно познакомимся с обработкой данных различных списков. Необходимо отметить, что использование вложенных циклов накладывает определённое ограничение на скорость выполнения программы. Использование вложенных циклов должно быть оправдано. Существуют разные парадигмы программирования. Используемые нами способы записи циклов относится к императивному программированию. Для изучения других парадигм рекомендуется обратиться к дополнительной литературе.

А теперь рассмотрим практический пример работы с текстовыми файлами, используя списки и циклы. Создайте новый текстовый документ, добавьте в него следующее содержимое:

---

```
1 2 3
2 4 6
3 6 9
4 8 12
5 10 15
```

---

Сохраните этот документ с именем input.txt. Создайте новый файл с расширением \*.py и сохраните код приведённый ниже.

---

```
print('Введите номер столбца: ')
n = int(input()) # Ввод номера столбца

infile = open('input.txt') # Открытие файла
filen = infile.readlines() # Чтение всех строк файла

columns = [] # Создание пустого списка columns
column = [] # Создание пустого списка column

for f in filen:
    line = f.split() # Построчное чтение данных
    columns.append(line) # Применение функции split() к строке
    print(columns) # Добавление строки к списку
    for i in range(len(columns)): # Вывод списка на экран
        if i == n: # Срез столбца с номером n
            column.append(columns[i][n])

print('Содержимое {} столбца: {}'.format(n))
print(column)

infile.close()
```

---

Результат программы.

---

```
>>> %Run example_09.py
```

Введите номер столбца:

0

[['1', '2', '3'], ['2', '4', '6'], ['3', '6', '9'], ['4', '8', '12'], ['5', '10', '15']]

Содержимое 0 столбца:

['1', '2', '3', '4', '5']

1 2 3 4 5

---

Проанализируем полученный результат. Программа открывает текстовый файл и построчно считывает данные в новый контейнер. В этом контейнере каждая строка хранится как единое целое. Для разделения элементов строки используется функция **split()**. Далее полученное значение новой строки добавляется к списку **columns**. В этом списке содержатся все столбцы из текстового файла. Следующий цикл **for** позволяет извлечь все элементы из конкретного столбца, который мы определили в самом начале программы. После работы с файлом его необходимо закрыть. В следующих разделах полученный результат будет использован при построении графиков по данным из файла.

## 1.5. Функции и рекурсия

Функция в Python – это часть программы, которая может быть многократно использована (не путать с математическими функциями). Использование функций позволяет разбить весь код программы на несколько частей. Функции реализуют логически выстроенную модель программы, определяют строгую конструкцию, которая легко воспринимается разными программистами. Отличительной особенностью функций является то, что они могут возвращать или не возвращать результат своей работы. Для объявления новой функции используется команда **def**. Рассмотрим пример создания функций для выполнения определённых операций.

---

```
a = int(input())
b = int(input())

def func_1(n1, n2): # Функция для нахождения суммы
    total = n1 + n2
    return total

def func_2(n1, n2): # Функция для нахождения разности
    differ = n1 - n2
    return differ

def func_3(n1, n2): # Функция для нахождения произведения
    multip = n1 * n2
    return multip

def func_4(n1, n2): # Функция для нахождения частного
    division = n1 / n2
    return division

print('Сумма двух чисел = ', func_1(a, b))
print('Разность двух чисел = ', func_2(a, b))
print('Произведение двух чисел = ', func_3(a, b))
print('Частное двух чисел = ', func_4(a, b))
```

В данной программе созданы 4 функции для реализации математических операций: сумма, разность, произведение и деление. В качестве параметров функции (формальных параметров) передаются значения n1 и n2. В качестве аргументов функции передаются значения a и b. Отличие параметров от аргументов в том, что набор параметров функции является фиксированным, а вот значение аргументов при вызове функции может отличаться. Внутри функции для возврата результата функции используется команда **return**.

Приведём ещё один пример использования функции для нахождения факториала числа ( $n!$ ). В программе будет показан метод нахождения факториала без использования функции и с использованием. Преимуществом

применения функции является то, что готовую функцию можно использовать несколько раз в программе.

---

```
n = int(input())

fakt = 1
for i in range(1, n + 1):      # Нахождение факториала
    fakt *= i
print('Факториал без функции = ', fakt)

def faktorial(n):            # Функция нахождения факториала
    fakt = 1
    for i in range(1, n + 1):
        fakt *= i
    return fakt
print('Факториал через функцию = ', faktorial(n))
```

---

Результат программы при  $n = 5$ .

---

```
>>> %Run example_10.py
```

```
Факториал без функции = 120
Факториал через функцию = 120
```

---

Теперь рассмотрим процесс запуска функции из самой себя, который в программировании принято называть рекурсией. Далее приведена программа для нахождения факториала заданного числа с использованием рекурсии.

---

```
n = int(input())

def faktorial(n):            # Функция нахождения факториала
    if n == 0:
        return 1
    return n * faktorial(n - 1) # Рекурсия функции faktorial(n)

print(faktorial(n))
```

---

Для нахождения факториала введённого числа, в отличие о рассмотренных ранее, реализуется процесс **перехода**, по которому проводится расчёт по известному результату для меньшего параметра. Например, для нахождения  $n!$  на первом шаге функция должна вернуть  $n$  умноженное на  $(n-1)!$ , и так до тех пор, пока не будет достигнута **база** (момент, когда в рекурсии не будет вызвана новая функция). Сначала проверяется условие окончания процесса рекурсии, а потом выполняются команды.

## 1.6. Множества и словари

Множества – это тип данных в языке Python, который содержит набор элементов и не имеет определённого порядка. Множества в Python напоминают множества из математики. Объекты множества можно добавлять, удалять, перебирать и проверять на принадлежность к множеству. Множества выделяются фигурными скобками, а могут быть созданы с помощью функции `set()`. Сами множества могут содержать объекты разных типов.

```
nSet_1 = {3, 1, 2, 3}                      # Создание множества
print(nSet_1)

nSet_2 = set((3, 1, 2))                     # Создание множества из кортежа
print(nSet_2)

nSet_3 = set([3, 1, 2])                      # Создание множества из списка
print(nSet_3)

nSet_4 = set(range(1, 10, 2))                # Создание множества, используя range()
print(nSet_4)

nSet_5 = {1, (1, 2, 3), 'Python'} # Создание множества из разных типов данных
print(nSet_5)
```

Проанализируйте результат работы данной программы самостоятельно. Обратите внимание на две особенности множеств. При выводе множеств функцией `print()` элементы множества выстраиваются в порядке возрастания (в случае букв и слов по алфавиту), а если какие-то элементы повторяются, то все повторы будут удалены из множества.

Теперь рассмотрим принцип работы с отдельными элементами множества. Поскольку множества являются итерируемыми объектами, то для их обработки можно использовать цикл `for`.

```
nSet = {1, '2', 3, '4', 5}                  # Создание множества
for elem in nSet:                            # Перебор всех элементов множества
    print(elem, end=' ')                    # Печать элемента на экран
```

Групповые операции для обработки множеств. В таблице ниже представлены операции, которые позволяют обрабатывать не отдельные элементы множества, а множество целиком. Логические операции:

Операция	Описание
A   B	Объединение множеств
A & B	Пересечение множеств
A - B	Множество, элементы которого входят в A, но не входят в B
A ^ B	Элементы входят в A   B, но не входят в A & B

Операции сравнения:

Операция	Описание
$A == B$	Все элементы совпадают
$A != B$	Есть различные элементы
$A <= B$	Все элементы A входят в B
$A < B$	$A <= B$ и $A != B$

Ниже представлен код программы, который демонстрирует примеры работы с множествами, используя логические операции и операции сравнения.

---

```
nSet_1 = {1, 2, 3, 4, 5}
nSet_2 = {4, 5, 6, 7, 8}

print(nSet_1 | nSet_2)
print(nSet_1 & nSet_2)
print(nSet_1 - nSet_2)
print(nSet_1 ^ nSet_2)

print(nSet_1 == nSet_2)
print(nSet_1 != nSet_2)
print(nSet_1 <= nSet_2)
print(nSet_1 < nSet_2)
```

---

Результат программы.

---

```
>>> %Run example_11.py

{1, 2, 3, 4, 5, 6, 7, 8}
{4, 5}
{1, 2, 3}
{1, 2, 3, 6, 7, 8}
False
True
False
False
```

---

Ещё одной сложной структурой данных в языке Python являются словари. Словари удобно использовать тогда, когда используется сопоставление пары элементов. Например, указание страны и её столицы. Первый элемент называется ключ, второй называется значением.

---

```
countries = {'Russia' : 'Moscow', 'Germany' : 'Berlin', 'France' : 'Paris'}
```

---

Теперь рассмотрим процесс создания словаря, добавление и удаление элементов словаря. Отметим, что словари являются итерируемыми объектами. Для обработки словарей можно использовать цикл **for**.

---

```
countries = dict()          # Создание пустого словаря

countries['Russia'] = 'Moscow' # Добавление 1 ключ-значение
```

---

```
countries['Germany'] = 'Berlin' # Добавление 2 ключ-значение
countries['France'] = 'Paris'   # Добавление 3 ключ-значение

print(countries)

del countries['Germany']      # Удаление элемента с ключом 'Germany'
print(countries)
```

---

Предлагается самостоятельно изучить результат работы программы. Следующий пример показывает использование цикла **for** для обработки словарей.

```
countries = {'Russia' : 'Moscow', 'Germany' : 'Berlin', 'France' : 'Paris'}

for country in countries:           # Обращение к элементу по индексу
    print(country, countries[country])

for country, capital in countries.items(): # Обращение через функцию items()
    print(country, capital)

for country in countries.keys():       # Обращение к ключу словаря
    print(country)

for capital in countries.values():     # Обращение к значениям словаря
    print(capital)

countries.setdefault('Spain', 'Madrid') # Добавления новой пары
print(countries)
```

---

В программе приведены два цикла **for**, которые показывают, как можно обратиться к паре ключ-значение в заданном словаре. В первом цикле для обращения к элементу используют индекс этого элемента в словаре. Второй цикл использует специальную функцию **items()**, позволяющую извлечь из словаря ключ и значение.

## 1.7. Введение в объектно-ориентированное программирование

Объектно-ориентированное программирование является одной из парадигм современного программирования, созданной на основе императивного программирования. Ранее мы уже обсуждали особенности императивного программирования. Кроме того, существует декларативное программирование и функциональное программирование. Остановимся подробнее на объектно-ориентированном программировании (ООП). Для ООП определяют три наиболее важных понятия: инкапсуляция, наследование и полиморфизм.

**Инкапсуляция** – это процесс объединения данных и методов работы с этими данными в один объект, скрытие внутреннего устройства объекта и создание интерфейса взаимодействия.

**Наследование** – это возможность создания нового класса (или объекта), на основе существующего класса с заимствованием его атрибутов.

**Полиморфизм** – это возможность использования одного и того же метода или функции в разных классах.

Под **классом** мы будем понимать описание структуры объекта (полей структуры) и методов (функций) для работы с данными в этой структуре.

**Объектом** называется конкретный экземпляр класса, в котором поля структуры заполнены значениями.

Приведём пример создания класса и нового объекта. Пусть по условию задачи нам требуется найти площадь прямоугольника со сторонами `side_a` и `side_b`. Создадим класс `Area`. Класс `Area` имеет несколько методов, которые позволяют изменять атрибуты объекта: `change_side_a()`, `change_side_b()`, `change_side_ab()`. Ещё один метод `area()` позволяет вычислить произведение.

```
class Area:  
    side_a = 5  
    side_b = 5  
  
    def change_side_a(self, new_side_a):          # Изменение атрибута side_a  
        self.side_a = new_side_a  
  
    def change_side_b(self, new_side_b):          # Изменение атрибута side_b  
        self.side_b = new_side_b  
  
    def change_side_ab(self, new_side_a, new_side_b):  
        self.side_a = new_side_a  
        self.side_b = new_side_b  
  
    def area(self):  
        return self.side_a*self.side_b  
  
obj = Area()                                     # Создание нового объекта  
print(obj.side_a, obj.side_b)                    # Вывод на экран атрибутов  
print(obj.area())                                # Площадь прямоугольника  
  
obj.change_side_a(2)                            # Изменение атрибута side_a
```

```
obj.change_side_b(2)          # Изменение атрибута side_b
print(obj.side_a, obj.side_b)  # Вывод на экран новых атрибутов
print(obj.area())              # Площадь прямоугольника

obj.change_side_ab(10, 10)     # Изменение атрибутов

print(obj.side_a, obj.side_b)  # Вывод на экран новых атрибутов
print(obj.area())              # Площадь прямоугольника
```

---

Конструкция `obj = Area()` создаёт новый объект для класса `Area`. Далее на экран выводится текущие значения атрибутов для этого объекта (атрибуты класса). Используя данные атрибуты, находят площадь прямоугольника. В программе происходит изменение атрибутов заданными методами и повторное вычисление площади прямоугольника.

Следующий пример демонстрирует возможность наследования атрибутов и методов одного класса от другого.

---

```
class Area_1:                      # Создание класса Area_1
    side_a = 5
    side_b = 5

    def change_side_a(self, new_side_a):      # Изменение атрибута side_a
        self.side_a = new_side_a

    def change_side_b(self, new_side_b):      # Изменение атрибута side_b
        self.side_b = new_side_b

    def area(self):
        return self.side_a * self.side_b

class Area_2(Area_1):               # Создание класса Area_2

    def change_side_ab(self, new_side_a, new_side_b):
        self.side_a = new_side_a
        self.side_b = new_side_b

obj = Area_2()                    # Создание нового объекта

print(obj.side_a, obj.side_b)      # Вывод на экран атрибутов
print(obj.area())                 # Площадь прямоугольника

obj.change_side_ab(3, 3)

print(obj.side_a, obj.side_b)      # Вывод на экран новых атрибутов
print(obj.area())                 # Площадь прямоугольника
```

---

Обратите внимание на запись `Area_2(Area_1)`, которая сообщает нам о том, что класс `Area_2` наследуется все атрибуты и методы родительского класса `Area_1`. Сам класс содержит метод, который позволяет изменить свойства атрибутов.

Рассмотрим конкретный пример проявления полиморфизма. В данном примере в двух разных классах существуют два разных метода, но имеющих одинаковое название.

---

```
a = int(input())
b = int(input())

class One:                      # Определение класса One
    def function(self):          # Определение функции function для класса One
        return a + b

class Two:                       # Определение класса Two
    def function(self):          # Определение функции function для класса Two
        return a - b

obj_1 = One()                   # Создание нового объекта obj_1
obj_2 = Two()                   # Создание нового объекта obj_2

print(obj_1.func())
print(obj_2.func())
```

Ранее с проявлением полиморфизма мы сталкивались, когда применяли к разным типам данных функцию `len()`, позволяющую определить длины последовательности данных.

Для классов в языке Python существует специальный метод, который позволяет автоматически при создании объекта создавать ему атрибуты. Специально вызывать данный метод не нужно, он будет запущен сразу при вызове класса. Такой метод называется **конструктором класса** и обозначается именем `__init__` (два подчёркивания до и после init).

---

```
class YesInit:                  # Класс с конструктором класса
    def __init__(self, one, two):
        self.fname = one
        self.sname = two

obj = YesInit('Elon', 'Musk')
print (obj.fname, obj.sname)      # Вывод: Elon Musk

class NoInit:                    # Класс без конструктором класса
    def names(self, one, two):
        self.fname = one
        self.sname = two

obj = NoInit()

obj.names('Elon', 'Musk')
print (obj.fname, obj.sname)      # Вывод: Elon Musk
```

В обоих случаях результат выполнения программы будет одинаковый.

Ну и, напоследок, остановимся на таком понятие как **магические методы**. Это специальные методы, которые могут выполнять стандартные операции над объектами, используя операторы. Например, магический метод `__add__` (два подчёркивания до и после `add`).

---

```
class mMet:  
    def __init__(self, x):      # Создание конструктора __init__  
        self.base = x  
  
    def __add__(self, a):        # Создание метода __add__  
        return self.base + a  
    def __sub__(self, a):        # Создание метода __sub__  
        return self.base - a  
    def __mul__(self, a):        # Создание метода __mul__  
        return self.base * a  
    def __div__(self, a):        # Создание метода __div__  
        return self.base / a  
  
res = mMet(5)  
  
print(res.__add__(3))          # Вывод: 8  
print(res.__sub__(3))          # Вывод: 2  
print(res.__mul__(3))          # Вывод: 15  
print(res.__div__(5))          # Вывод: 1.0
```

---

Такой метод использования операторов называется **перегрузкой операторов**. Особенность применения оператора зависит от типа объекта к которому он применяется. Дополнительную информацию о видах методов можно узнать из справочной литературы. Ввиду их большого количества мы не будем останавливаться на них более подробно.



## Задания для самостоятельной работы

### 1. Целые числа, ввод-вывод, простые операции со строками

**1.1.** Напишите программу, которая, используя функцию `int()`, позволяет вводить в программу имя и фамилию пользователя. На выходе программы методом `print()` вы должны получить строку вида «Здравствуйте, Иван Иванов!». Задание считается выполненным полностью, если предоставлены два варианта программы: вывод строки без использования оператора конкатенации `«+»` и с использованием данного оператора.

---

```
>>> %Run example_practice_01.py
```

```
Иван  
Иванов
```

```
Здравствуйте, Иван Иванов!
```

---

**1.2.** Напишите программу, которая принимает на вход целое число  $n$  от 0 до 9. Программа должна вывести на экран квадраты, количество которых соответствует номеру  $n$ .

Примечание: вывод строк можно упростить, если использовать один из способов работы со строками. Для вывода определённого количества строк  $n$ , строку можно умножить на число  $n$ .

---

```
>>> %Run example_practice_02.py
```

```
5
```

```
-----  
| | | | | | |  
| | | | | | |  
-----
```

---

**1.3.** Даны два целых числа  $n$  и  $k$ . Необходимо найти остаток от деления и результат целочисленного деления числа  $n$  на число  $k$ . Далее нужно сложить два результата и вывести сумму на экран.

---

```
>>> %Run example_practice_03.py
```

```
5  
2  
Результат программы: 3
```

---

**1.4.** Дано трёхзначное целое число. Необходимо определить количество сотен, десятков и единиц в этом числе. Программа должна выводить каждое значение с новой строки. Последней строкой необходимо вывести произведение всех этих чисел.

---

```
>>> %Run example_practice_04.py
```

```
234
```

```
Количество сотен = 2  
Количество десятков = 3  
Количество единиц = 4  
Произведение чисел: 24
```

---

**1.5.** Напишите программу, которая принимает на вход двузначное целое число  $n$ . Программа должна рассчитывать количество десятков  $k$  этого числа. Далее необходимо возвести число 2 в степень  $k$ . Задание считается выполненным полностью, если предоставлены два варианта программы: с использованием функции  $\text{pow}(n, k)$  и с использованием стандартного оператора « $**$ ».

**1.6.** На вход программа принимает целое трёхзначное число  $m$  – количество минут. Необходимо перевести минуты в понятный формат времени  $H:M$  и вывести результат на экран.

Примечание: для вывода времени нужно использовать оператор конкатенации « $+$ ». Т.к. переменные имеют тип данных `int`, то для выполнения операции конкатенации требуется их перевод в тип данных строка функцией `str()`.

---

```
>>> %Run example_practice_06.py
```

```
569
```

```
Текущее время 9:29
```

---

**1.7.** Дано натуральное число  $n$  в интервале от 1 до 9. Необходимо рассчитать два значения для этого числа:  $n - 1$  и  $n + 1$ . Каждое полученное число нужно будет разделить на 10 и сложить результат. Программа должна вывести на экран округлённое функцией  $\text{round}(x, n)$  полученное значение с точностью до 2 знаков после запятой.

**1.8.** Яблоко на овощном рынке стоит  $n$  рублей и  $m$  копеек. Определите, сколько всего рублей и копеек нужно заплатить за  $N$  яблок. Программа должна принять на вход три числа:  $n$ ,  $m$ ,  $N$ . На выходе программы должна быть одна строка вида “Нужно заплатить 44 рубля(ей) и 80 копеек”.

---

```
>>> %Run example_practice_08.py
```

```
5  
60  
8
```

Нужно заплатить 44 рубля(ей) и 80 копеек

---

**1.9.** Дано трёхзначное целое число  $n$ . Необходимо проверить, является ли данное число палиндромом. На выходе программа должна дать одно из значений: `True` или `False`.

Примечание: поскольку на данном этапе нам неизвестны условные операторы, то для проверки условия необходимо использовать оператор «`==`».

---

```
>>> %Run example_practice_09.py
```

```
123  
False
```

```
101  
True
```

---

**1.10.** Данна геометрическая фигура параллелепипед с высотой  $h$ , в основании которого лежит прямоугольник со сторонами  $a$  и  $b$ . Необходимо рассчитать площадь основания и объём полученной фигуры.

Примечание: для вывода результатов расчёта необходимо использовать функцию `format()`. Количество знаков после запятой не должно превышать 2. Ответ для каждого расчёта выводится с новой строки.

---

```
>>> %Run example_practice_10.py
```

```
3  
10
```

Площадь основания = 50 кв.ед.  
Объём фигуры = 150 куб.ед.

---

**1.11.** Данна строка s из последовательности натуральных чисел от 0 до 9 (без пробелов и иных символов между цифрами). Используя срезы данных, необходимо выполнить следующие преобразование строки и вывести на экран: элементы строки с 2 по 3, элемент строки с индексом [-2], элементы строки с 0 по 4, все четные элементы строки, все нечетные элементы строки, все элементы строки за исключением двух последних, все элементы в обратном порядке, все нечетные элементы в обратном порядке, все чётные в обратном порядке, длину строки функцией len().

**1.12.** Данна строка s из последовательности натуральных чисел от 0 до 9 (без пробелов и иных символов между цифрами). Необходимо найти методом find() в строке индекс цифры 5 и на основе срезов выполнить вывод строки, начиная с элемента, имеющего данный индекс.

**1.13.** Данна строка s из последовательности натуральных чисел от 0 до 9 (без пробелов и иных символов между цифрами). Необходимо в данной строке методом replace() заменить числа 1, 3, 5 и 7 на «one», «three», «five» и «seven».

**1.14.** Данна строка s из последовательности натуральных чисел от 0 до 9 (без пробелов и иных символов между цифрами). Необходимо в данной строке методом replace() заменить последовательность «567» на «1234567», методом len() определить длину новой и старой строки. На экран программа должна вывести разность двух длин строк (по модулю).

Примечание: для нахождения модуля разности двух чисел можно использовать функцию abs().

**1.15.** Данна строка s = “There is no time like the present”. Необходимо функцией count() найти какое количество раз встречается буква «е» в данной строке, методом pow() возвести данное число в квадрат. Далее нужно найти количество пробелов в данной строке и методом pow() возвести данное число в квадрат. Программа должна рассчитать разницу двух полученных чисел и вывести на экран.

## 2. Вещественные числа и операции над ними

**2.1.** Напишите программу, которая на вход принимает значение R (радиус сферы). Программа должна рассчитать площадь большого круга, площадь поверхности сферы и объём сферы.

Примечание: если в результате получается вещественное число, то его следует округлить методом round() до 2 знаков после запятой.

**2.2.** Необходимо написать программу, которая позволяет решить систему линейных алгебраических уравнений из двух неизвестных методом Крамера. На вход программа принимает 6 чисел. В результате программа должна вывести на экран корни для решения системы. Для вывода неизвестных значений необходимо использовать функцию format().

Примечание: рассмотрим следующую систему уравнений:

$$\begin{cases} 5x + 4y = 14 \\ 2x - 6y = -2 \end{cases}$$

где основная матрица имеет вид  $\begin{pmatrix} 5 & 4 \\ 2 & -6 \end{pmatrix}$ , столбец свободных членов  $\begin{pmatrix} 14 \\ -2 \end{pmatrix}$ .

Неизвестные  $x$  и  $y$  находим из следующих выражений ( $\Delta_x, \Delta_y, \Delta$  - определители матриц):

$$x = \frac{\Delta_x}{\Delta} = \frac{\begin{vmatrix} 14 & 4 \\ -2 & -6 \end{vmatrix}}{\begin{vmatrix} 5 & 4 \\ 2 & -6 \end{vmatrix}} = \frac{-84 + 8}{-30 - 8} = 2 \text{ и } y = \frac{\Delta_y}{\Delta} = \frac{\begin{vmatrix} 5 & 14 \\ 2 & -2 \end{vmatrix}}{\begin{vmatrix} 5 & 4 \\ 2 & -6 \end{vmatrix}} = \frac{-10 - 28}{-38} = 1.$$

**2.3.** Дано положительное вещественное число  $n$ . Необходимо вывести на экран квадрат его дробной части.

Примечание: при написании программы нельзя использовать условные операторы и циклы. Допустимо использование методов округления чисел из библиотеки math.

**2.4.** Цена товара в магазине  $n$  дана с точностью до двух знаков после запятой, где всё, что после запятой указано в копейках. Необходимо рассчитать сколько потребуется денег для покупки  $k$  единиц товара. Программа должны выводить отдельно количество рублей и копеек на экран.

Примечание: при написании программы нельзя использовать условные операторы и циклы. Допустимо использование методов округления чисел из библиотеки math.

**2.5.** Даны две стороны прямоугольного треугольника  $a$  и  $b$ . Необходимо найти гипотенузу треугольника и рассчитать sin(), cos() и tg() для двух острых углов.

**2.6.** Даны два вектора:  $a = \{X_1, Y_1, Z_1\}$  и  $b = \{X_2, Y_2, Z_2\}$ . Необходимо найти скалярное произведение векторов, угол между векторами и векторное произведение векторов.

Примечание: для нахождения неизвестных величин можно использовать следующие формулы:

$$a \cdot b = X_1 \cdot X_2 + Y_1 \cdot Y_2 + Z_1 \cdot Z_2, \quad (1)$$

$$\cos(\varphi) = \frac{a \cdot b}{|a| \cdot |b|} = \frac{X_1 \cdot X_2 + Y_1 \cdot Y_2 + Z_1 \cdot Z_2}{\sqrt{X_1^2 + Y_1^2 + Z_1^2} \cdot \sqrt{X_2^2 + Y_2^2 + Z_2^2}}, \quad (2)$$

$$\vec{a} \cdot \vec{b} = \begin{vmatrix} Y_1 & Z_1 \\ Y_2 & Z_2 \end{vmatrix} \begin{vmatrix} Z_1 & X_1 \\ Z_2 & X_2 \end{vmatrix} \begin{vmatrix} X_1 & Y_1 \\ X_2 & Y_2 \end{vmatrix}. \quad (3)$$

В выражении (3) для нахождения координат нового вектора требуется найти 3 определителя второго порядка.

**2.7.** Напишите программу, которая преобразует давление  $P$  из мм рт. ст. в паскали (Па). Программа должна выводить на экран давление в кПа с точностью до 3-ёх знаков после запятой.

Примечание: при составлении выражения для конвертации величин используйте соотношение 1 мм рт.ст. = 133,3 Па.

**2.8.** Напишите программу, которая принимает на вход значение температуры по шкале Цельсия с точностью до двух градусов после запятой, преобразует данное значение в градусы Фаренгейта, Кельвина, градусы Реомюра и Ранкина.

Примечание: для нахождения неизвестных величин можно использовать следующие формулы для перевода:

$$[{}^\circ C] = \frac{5}{9}([{}^\circ F] - 32), \quad (\text{по Фаренгейту})$$

$$[{}^\circ C] = [{}^\circ K] - 273.15, \quad (\text{по Кельвину})$$

$$[{}^\circ C] = \frac{5}{4}[{}^\circ R], \quad (\text{по Реомюру})$$

$$[{}^\circ C] = \frac{5}{9}[{}^\circ R_a] - 273.15 \quad (\text{по Ранкину})$$

**2.9.** Напишите программу, которая принимает два параметра на вход: масса тела  $m$  (кг) и рост  $h$  (м) человека. Необходимо рассчитать индекс массы тела (ИМТ) по заданным параметрам. На выходе программа должна вывести ИМТ и дать одно из значений: True или False. True в случае, если ИМТ строго попадает в диапазон от 18.5 до 30. False во всех остальных случаях.

Примечание: поскольку на данном этапе нам неизвестны условные операторы, то для проверки условия необходимо использовать операторы « $\ll=$ » или « $\gg=$ ».

```
>>> %Run example_practice_09.py
```

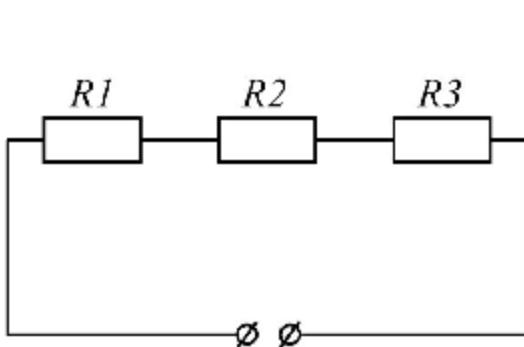
68

1.74

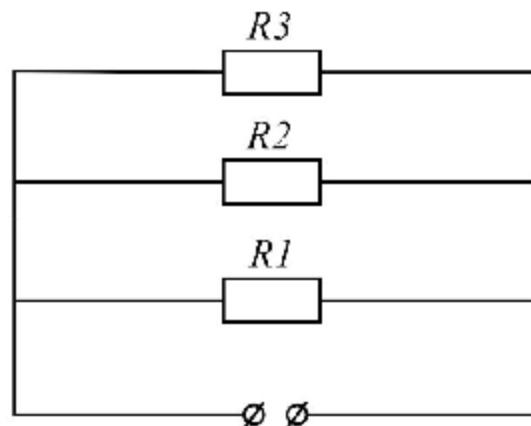
ИМТ человека = 22.46

True

**2.10.** Напишите программу, которая позволяет рассчитать неизвестные электрические величины для двух случаев подключения сопротивлений в цепи постоянного тока. В первом случае сопротивления подключены последовательно (*a*), а во втором параллельно (*b*).



*a)*



*b)*

1) Найти напряжение источника  $U$ , если известно, что ток в цепи  $I = i$  А, а  $R_1 = R_2 = R_3 = r$  Ом. Переменные  $i$  и  $r$  произвольные значения, заданные пользователем.

2) Найти ток в цепи  $I$ , если известно, что напряжение источника  $U = u$  В, а  $R_1 = R_2 = R_3 = r$  Ом. Переменные  $u$  и  $r$  произвольные значения, заданные пользователем.

Примечание: для решения задачи необходимо использовать соотношения для закона Ома при последовательном и параллельном подключении:

	последовательно	параллельно
Сила тока	$I = I_1 = I_2 = \dots = I_n$	$I = I_1 + I_2 + \dots + I_n$
Напряжение	$U = U_1 + U_2 + \dots + U_n$	$U = U_1 = U_2 = \dots = U_n$
Сопротивление	$R = R_1 + R_2 + \dots + R_n$	$\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2} + \dots + \frac{1}{R_n}$

Если в результате расчётов получается вещественное число, то его следует округлить методом `round()` до 2 знаков после запятой.

### 3. Условный оператор, цикл while и цикл for

**3.1.** Напишите программу, которая принимает на вход 2 числа  $a$ ,  $b$  и сравнивает введённые числа. Если  $a > b$ , то программа должна вывести на экран число  $a$ , иначе нужно вывести число  $b$ .

**3.2.** Напишите программу, которая принимает на вход 2 числа  $a$ ,  $b$  и сравнивает введённые числа. Если  $a > b$ , то программа должна вывести на экран строку “Число  $a > b$ ”. Если  $a = b$ , то программа должна вывести на экран строку “Число  $a = b$ ”, иначе строку “Число  $a < b$ ”.

**3.3.** Напишите программу, которая принимает на вход 3 числа  $a$ ,  $b$ ,  $c$  и сравнивает введённые числа. Если  $a \geq b$  и  $a \geq c$ , то программа должна вывести на экран число  $a$ . Если  $b \geq a$  и  $b \geq c$ , то программа должна вывести на экран число  $b$ . Если  $c \geq b$  и  $c \geq a$ , то программа должна вывести на экран число  $c$ . Во всех остальных случаях нужно вывести на экран строку “Ни одно из условий не выполнено”.

**3.4.** Напишите программу, которая принимает на вход 3 числа  $a$ ,  $b$ ,  $c$ . Необходимо вывести на экран последовательность этих чисел в порядке возрастания.

Примечание: для сортировки данных необходимо использовать свойство перестановки  $(a, b) = (b, a)$ . Никаких новых переменных создавать нельзя.

**3.5.** На вход программа принимает переменную  $year$  (год с указанием столетия). Используя условные и логические операторы, необходимо определить, является ли введённый год високосным или нет.

**3.6.** Дано натуральное трёхзначное число  $n$ . Используя цикл `while`, необходимо вывести на экран число в третьей степени последовательности  $k = 1, 2, 3\dots$ . Условием выхода из цикла `while` и окончание работы программы является нарушение условия  $k^{**3} < n$ .

Примечание: для выхода из цикла можно использовать оператор `break`. Все числа должны быть выведены в одну строку через пробел.

---

```
>>> %Run example_practice_06.py
```

```
956  
1 8 27 64 125 216 343 512 729
```

---

**3.7.** Дано натуральное число  $n$ . Используя цикл `while`, необходимо рассчитать факториал числа  $n$  (или  $n! = 1 \cdot 2 \cdot 3 \dots (n - 1) \cdot n$ ). Для проверки программы можно использовать стандартную функцию `factorial()`.

**3.8.** Допустим, что мы открыли банковский вклад в размере  $n$  рублей, который позволяет нам получать дивиденды равные 10% годовых. Необходимо рассчитать, через сколько дней после открытия вклада мы получим прибыль в размере 25% от нашей исходной суммы.

Примечание: при написании программы использовать только цикл while.

**3.9.** Напишите программу, которая суммирует введённые с клавиатуры числа. Программа должна сохранять сумму чисел и вывести её на экран в тот момент, когда пользователь ввёл 0.

Примечание: при написании программы использовать только цикл while.

**3.10.** Напишите программу, которая принимает на вход одно число. Если введённое число не равно нулю, то программа снова позволяет ввести новое число. Программа должна сохранять максимальное число из введённых и вывести это число на экран в тот момент, когда пользователь ввёл 0.

Примечание: при написании программы использовать только цикл while.

**3.11.** Напишите программу, которая позволяет найти корни для решения квадратного уравнения. Для проверки дискриминанта необходимо использовать условные операторы.

**3.12.** Дано натуральное число  $n$ . Используя цикл for, необходимо рассчитать факториал числа  $n$  (или  $n! = 1 \cdot 2 \cdot 3 \dots (n - 1) \cdot n$ ). Для проверки программы можно использовать стандартную функцию factorial().

**3.13.** Напишите программу, которая принимает на вход 2 натуральных числа  $a$  и  $b$ . Используя цикл for, необходимо вывести на экран последовательность чисел от  $a$  до  $b$  с шагом 1 в одну строку через пробел.

**3.14.** Напишите программу, которая принимает на вход целое число  $n$  от 1 до 9. Используя цикл for, программа должна вывести на экран квадраты, количество которых соответствует номеру  $n$ . В центре каждого квадрата должен быть его порядковый номер.

---

```
>>> %Run example_practice_14.py
```

5



**3.15.** Данна строка  $s = \text{"Better late than never"}$ . Необходимо в цикле for найти какое количество раз встречается буква «е» в данной строке, методом pow() возвести данное число в квадрат. Далее нужно найти количество пробелов в данной строке и методом pow() возвести данное число в квадрат. Программа должна рассчитать разницу двух полученных чисел и вывести на экран.

## 4. Кортежи, списки и их сортировка

**4.1.** Создайте кортеж из 10 натуральных чисел, используя функцию range( $n, k$ ), где  $n$  и  $k$  число вводимые пользователем. Далее посчитайте циклом for сумму чётных ( $\text{total\_even}$ ) и нечётных ( $\text{total\_odd}$ ) элементов данного кортежа. Сравните полученные суммы. Если сумма чётных больше, то программа должна вывести на экран строку вида “Сумма чётных элементов больше:  $\text{total\_even} > \text{total\_odd}$ ”, иначе строку вида “Сумма нечётных элементов больше:  $\text{total\_even} < \text{total\_odd}$ ”.

**4.2.** Напишите программу, которая принимает на вход две строки последовательных целых чисел и преобразует их в 2 кортежа. Далее необходимо найти в каждом кортеже максимальное и минимальное значение, определить разницу для максимального и минимального. Вывести на экран результаты расчётов.

Примечание: для создания кортежа из строки чисел можно использовать конструкцию: `tuple(map(int, input().split()))`. Поиск максимального и минимального значения можно выполнить циклом for или стандартными функциями.

**4.3.** Создайте два кортежа по 5 элементов в каждом. Первый кортеж содержит наименования фруктов: яблоко, банан, апельсин, груша, мандарин. Второй кортеж содержит количество фруктов из первого кортежа: 5, 8, 3, 5, 10. Напишите программу, которая должна принимать на вход название фрукта, определять индекс фрукта в первом кортеже (`fruits.index()`), а на экран выводить количество единиц из второго кортежа по найденному индексу.

**4.4.** Напишите программу, которая принимает на вход целое положительно число  $n$  от 2 до 10. Далее необходимо в цикле for создать список из случайных целых чисел в диапазоне от 0 до 100. Количество элементов списка должно соответствовать числу  $n$ . Выведите на экран три строки: не сортированный список, сортированный список и максимальное значение в списке.

Примечание: для генерации случайного числа используйте библиотеку random и функцию random(). Сортировку элементов выполнить методом sort().

**4.5.** Напишите программу, которая принимает на вход целое положительно число  $n$  от 2 до 100. Далее необходимо в цикле `for` создать список из случайных целых чисел в диапазоне от 0 до 100. Количество элементов списка должно соответствовать числу  $n$ . Программа должна найти сумму всех элементов списка и вывести на экран. Использование стандартных функций для поиска суммы не допускается.

**4.6.** Дан список следующего вида: `mylist = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]`. Необходимо к каждому списку добавить два новых элемента. Первый элемент должен быть суммой трёх предыдущих, а второй произведением. В результате программа должна вывести на экран новый список, в котором будет храниться 3 новых списка по 5 элементов в каждом.

**4.7.** Создайте в цикле `for` список из случайных целых чисел в диапазоне от 0 до 100. Для полученной последовательности чисел необходимо рассчитать среднее значение, дисперсию и стандартное отклонение (среднеквадратичное отклонение).

Примечание: для нахождения неизвестных величин можно использовать следующие формулы:

$$\bar{x} = \frac{\sum x_i}{n}, \quad (1)$$

$$\sigma^2 = \frac{\sum (x_i - \bar{x})^2}{n}, \quad (2)$$

$$\sigma = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n}}. \quad (3)$$

В данном случае считаем, что математическое ожидание в формуле (2) совпадает со средним арифметическим.

**4.8.** Создайте в цикле `for` список из 10 случайных целых чисел в диапазоне от 0 до 99. Необходимо найти минимальный и максимальный элемент последовательности и поменять их местами. Для сравнения вывести на экран две строки: исходный список и новый список.

Примечание: для сортировки данных необходимо использовать свойство перестановки  $(a, b) = (b, a)$ . Никаких новых переменных создавать нельзя.

**4.9.** Создайте в цикле `for` список из 30 случайных целых чисел в диапазоне от 0 до 99. Выполните сортировку списка таким образом, чтобы элемент с нулевым индексом списка был максимальным. Программа должна вывести максимальное значение на экран.

Примечание: использование стандартных методов для поиска максимального значения в списке не допускается.

**4.10.** Дан список сотрудников организации следующего вида:

```
staff = [['Иванов Александр', 'слесарь', 16000, 20],  
         ['Петров Евгений', 'наладчик', 17000, 22],  
         ['Сидоров Владимир', 'механик', 18000, 24]]
```

В данном списке в каждой строке элемент с индексом 2 сообщает о заработке каждого сотрудника, а элемент с индексом 3 информацию о количестве отработанных дней сотрудником в текущем месяце. Необходимо выполнить следующее:

- 1) рассчитать сумму всех заработанных денег сотрудниками;
- 2) рассчитать средний заработка сотрудников организации;
- 3) определить для каждого сотрудника какое количество денег он зарабатывает за один свой рабочий день (результат округлить до двух знаков после запятой);
- 4) добавить полученное в п.п. (3) число к списку каждого сотрудника;
- 5) выполнить сортировку данных списка по последнему столбцу и определить самого высокооплачиваемого сотрудника;
- 6) вывод данных на экран должен соответствовать формату:

```
print('Сотрудник | Специальность | Заработка | дней | руб./день |')  
print('-----')  
for row in staff[:]:  
    print('{: <18} | {: <14} | {:. ^9.2f} | {:. ^4} | {:. >9.1f} |'.format(  
        row[0], row[1], row[2], row[3], row[4]))
```

## 5. Множества и словари

**5.1.** Даны два множества элементов. Каждое множество содержит по 10 элементов произвольных целых чисел в диапазоне от 0 до 10. Необходимо выполнить следующие операции над множествами: объединение, пересечение, разность и симметричную разность.

**5.2.** Дано множество, которое содержит 10 элементов произвольных целых чисел в диапазоне от 0 до 10. Необходимо выполнить сортировку элементов и вывести на экран минимальное и максимальное значение множества.

**5.3.** Дано множество, которое содержит 10 элементов произвольных вещественных чисел в диапазоне от 0 до 10. Необходимо создать новое множество, которое будет содержать квадраты элементов исходного множества. Результат вывести на экран.

**5.4.** Даны два множества элементов. Каждое множество содержит по 10 элементов произвольных целых чисел в диапазоне от 0 до 10. Напишите программу, которая создаёт новое множество, в котором содержатся элементы, входящие в первое и второе множество. Результат необходимо вывести на экран.

Примечание: проверку принадлежности элемента к множеству необходимо выполнить в цикле for.

**5.5.** Создайте новый словарь, в котором содержится название животного. В качестве ключа использовать название на английском языке, в качестве значения название на русском языке. В исходном словаре должны быть: собака, кошка и кролик. Напишите программу, которая будет принимать на вход новую пару ключ-значение (каждое с новой строки) и добавлять в исходный словарь. Программа должна завершиться и вывести на экран полученный словарь, если пользователь ввёл слово ‘STOP’.

**5.6.** Создайте новый словарь, в котором содержится названия стран и их столиц. В качестве ключа использовать страна, в качестве значения название столицы. В словаре должны быть: Россия, США, Германия, Франция и Китай. Напишите программу, которая должна выполнять четыре цикла for:

- 1) в первом цикле программа построчно выводит строку “Столицей country является capital.”, где country и capital ключ и значение из словаря;
- 2) во втором цикле вывести пару ключ-значение, используя метод items();
- 3) в третьем цикле вывести ключи словаря, используя метод keys();
- 4) в четвёртом цикле вывести значения словаря, используя метод values().

**5.7.** Используйте исходный словарь из задания 5.6. Напишите программу, которая будет принимать на вход название страны. Если такая страна в словаре есть, то программа должна вывести название столицы, иначе выводится сообщение “Такой страны в словаре нет”.

**5.8.** Дан небольшой фрагмент текста. Необходимо подсчитать, сколько раз каждое слово встречается в этом тексте. Результаты поиска необходимо хранить в словаре, где ключом будет слово, а количество его употреблений – значением. Программа должна вывести на экран три наиболее часто используемых слова в тексте.

Примечание: для сортировки данных преобразуйте пару ключ-значение словаря в элементы списка, используя методом `items()`. Для сортировки списка используйте функцию с указанными атрибутами `sort(key=lambda i: i[1])`, где `i[1]` содержит информацию о количестве употреблений слова.

## 6. Функции и рекурсия

**6.1.** Напишите программу, которая принимает на вход два вещественных числа `a` и `b`. Программа должна содержать 4 функции: функция для нахождения суммы двух чисел, функция для нахождения разности двух чисел, функция для нахождения произведения двух чисел, функция для нахождения частного двух чисел. Далее необходимо последовательно вызвать каждую функцию с аргументами `a` и `b`, а результат вывести на экран в одну строку через пробел.

**6.2.** Напишите программу, которая содержит функцию для нахождения числа `n` возведённого в степень `k`. Пусть `n` является обязательным параметром, а `k` – необязательным. Значение `k = 2` по умолчанию.

**6.3.** Напишите функцию, которая в качестве аргумента может принимать именованный аргумент `fruit` (название фрукта). При вызове данной функции программа должна вывести на экран строку ‘Я люблю `fruit!`’, где `fruit` – строка с названием фруктом, например, ‘яблоко’.

**6.4.** Напишите функцию для нахождения факториала числа `n` (или  $n!=1 \cdot 2 \cdot 3 \dots (n-1) \cdot n$ ) с использованием цикла `for` или `while`. Функция используется в программе, которая должна принимать целое число и выводить на экран результат расчёта факториала.

**6.5.** Напишите функцию для нахождения факториала числа `n` (или  $n!=1 \cdot 2 \cdot 3 \dots (n-1) \cdot n$ ) с использованием рекурсии. Функция используется в программе, которая должна принимать целое число и выводить на экран результат расчёта факториала.

**6.6.** Используя анонимную функцию `lambda`, напишите программу для нахождения квадрата числа `n`.

**6.7.** Дан список произвольных целых чисел из 10 элементов. Напишите программу, которая находит квадрат каждого элемента списка, используя функцию `map()` и анонимную функцию `lambda`. Использование циклов не допустимо.

**6.8.** Дан список произвольных целых чисел из 10 элементов. Напишите программу, которая выполняет фильтрацию элементов списка, используя функцию `filter()` и анонимную функцию `lambda`. Условие фильтрации, например,  $n > 3$ . Использовать циклы нельзя.

**6.9.** Напишите функцию, которая возвращает случайное целое число в диапазоне от 1 до 6. Пусть данная функция имитирует результат броска игрального кубика. Далее напишите функцию, которая суммирует результат одновременного броска двух кубиков, а результат добавляет в список. Вторая функция имеет один параметр *n* – количество раз совместного броска кубиков. Необходимо подсчитать, сколько раз каждое число встречается в полученном списке. Результаты поиска необходимо хранить в словаре, где ключом будет число от 2 до 12, а количество его упоминаний – значением. Программа должна вывести на экран три наиболее часто встречающихся числа в списке.

Примечание: для генерации случайных чисел можно использовать функцию для генерации целых чисел в заданном диапазоне `randint()` из библиотеки `random`.

**6.10.** Дан список самых рейтинговых комедий США по версии сайта [kinopoisk.ru](http://kinopoisk.ru):

---

```
films = [
    ['Назад в будущее', 1985, 8.6, 116, ['фант.', 'комедия'], 19.0, 381.1],
    ['Назад в будущее 2', 1989, 8.3, 108, ['фант.', 'прикл.'], 40.0, 331.9],
    ['Малыш', 1921, 8.2, 68, ['драма', 'комедия'], 0.25, 5.45],
    ['Один дома', 1990, 8.2, 103, ['семейный', 'комедия'], 18.0, 476.68],
    ['Один дома 2', 1992, 7.9, 119, ['семейный', 'комедия'], 18.0, 358.9],
    ['Большой', 1988, 7.9, 104, ['фантастика', 'мелодрама'], 18.0, 151.6],
    ['Рыжий пёс', 2011, 7.7, 92, ['мелодрама', 'комедия'], 8.5, 21.18],
    ['Марли и я', 2008, 7.8, 115, ['драма', 'комедия'], 60.0, 255.7],
    ['Миссис Даутфайр', 1993, 7.7, 125, ['драма', 'комедия'], 25.0, 441.28],
    ['Кудряшка Сью', 1991, 7.6, 101, ['семейный', 'комедия'], 25.0, 33.6]
]
```

---

Необходимо выполнить следующее:

1) выполните сортировку данных списка по годам `films.sort(key=lambda i: i[1], reverse=True)`.

---

```
print('Название фильма | Год | Рейтинг | Длина | Бюджет | Сборы |')
print('-----')
for row in films[:]:
    print('{: <35} | {} | {:.2f} | {:.5} | {:.1f} | {:.1f} |'.format(
        row[0], row[1], row[2], row[3], row[5], row[6]))
```

---

2) напишите функцию `column_total(films, column, start_year, end_year)`, которая позволяет рассчитать сумму бюджетов (или сборов) всех фильмов в списке за выбранный период времени;

- 3) напишите функцию `column_mean(films, column, start_year, end_year)`, которая позволяет рассчитать среднее значение бюджетов (или сборов) всех фильмов в списке за выбранный период времени;
- 4) напишите функцию для расчёта стоимости одной минуты фильма, исходя из потраченного бюджета на фильм. Добавить полученное значение к исходному датасету в виде нового столбца и сохранить как новый список `new_films`;
- 5) отсортируйте новый список `new_films` по последнему столбцу и выведите новую таблицу данных на экран.



## 2. Предобработка исследовательских данных

В данном разделе мы изучим две популярные библиотеки для обработки, анализа и отображения различных массивов данных. Среди них можно выделить следующие: **numpy** и **pandas**. Библиотеки не входят в стандартный пакет интерпретатора Python. Перед работой с ними установите их на свой компьютер. Ознакомиться с документацией можно на сайте open-source разработчиков: <https://numpy.org/doc/> и <https://pandas.pydata.org/docs/>

Мы будем использовать библиотеки для загрузки данных, предварительной обработки и анализа. Дополнительно будут использованы специальные библиотеки **sklearn** и **scipy** для детального научного анализа.

### 2.1. Подключение и работа с библиотекой numpy

**Библиотека numpy** – это библиотека языка Python, позволяющая обрабатывать многомерные массивы и матрицы. Библиотека обладает множеством высокогенеральных математических функций для операций над данными. Базовым объектом **numpy** является однородный многомерный массив (`numpy.ndarray`). Многомерный массив состоит из элементов одного типа (чаще всего чисел). В таблице ниже приведены некоторые атрибуты объектов `ndarray`.

Атрибуты	Описание
<code>ndim</code>	число измерений (чаще их называют "оси") массива
<code>shape</code>	размеры массива, его форма
<code>size</code>	количество элементов массива
<code>dtype</code>	объект, описывающий тип элементов массива
<code>itemsize</code>	размер каждого элемента массива в байтах
<code>data</code>	буфер, содержащий фактические элементы массива

Наиболее простой способ создать массив – это использовать стандартную функцию **array()**. Массивы обычно создаются из списков или кортежей. Для подключения модуля к программе используются инструкция **import**.

```
import numpy as np

a = np.array([[1, 2, 3],      # Массив 3x3 из списков
              [1, 2, 3],
```

```

[1, 2, 3]])
b = np.arange(1, 10, 1)      # Массив от 1 до 10 с шагом 1
c = np.empty((3, 3))        # Массив 3x3 без заполнения
d = np.eye(5)                # Единичная матрица 5x5
e = np.linspace(0, 2, 9)     # Массив 0-2 всего 9 значений

```

---

Базовые операции для работы с массивами. К массивам можно применять следующие операции: арифметические, индексирования, срезов, итерирования, объединение, разбиение, копирование.

---

```

import numpy as np

# Арифметические операции (размер массивов одинаковый)

a = np.array([1, 2, 3, 4])
b = np.arange(1, 5)

print(a + b)      # Суммирование элементов
print(a - b)      # Вычитание элементов
print(a * b)      # Умножение элементов
print(a / b)      # Деление элементов
print(a ** b)     # Возведение в степень
print(a % b)      # Остаток от деления

np.sin(a)         # Тригонометрическая функция sin()
np.cos(a)         # Тригонометрическая функция cos()
np.arctan(b)      # Тригонометрическая функция arctan()

a = np.array([1, 2, 3, 4])

a.sum()           # Нахождение суммы элементов массива
a.min()           # Нахождения минимума массива
a.max()           # Нахождения максимума массива

a.min(axis = 0)   # Наименьшее число в каждом столбце (для строк = 1)

# Индексы, срезы, итерации

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

a[1, 2]           # Вторая строка, третий столбец (допустимо a[1][2])
a[:, 1]           # Второй столбец
a[:1]             # Первая строка
a[0:2, :]         # Первая и вторая строки

for row in a:      # Построчный вывод массива
    print(row)
for elem in a.flat: # Поэлементный вывод массива
    print(elem)

# Объединение и разделение массивов

```

```

a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6], [7, 8]])
np.vstack((a, b))      # Объединение по первым осям
np.hstack((a, b))      # Объединение по последним осям

a = np.array([1, 2])
b = np.array([3, 4])
np.column_stack((a, b)) # Объединение столбцов одномерных массивов
np.row_stack((a, b))   # Объединение строк одномерных массивов

a = np.array([[1, 2, 3], [3, 4, 5]])
np.hsplit(a, 3)         # Разбить массив на 3 части
np.hsplit(a, (2, 3))   # Разрезать а после второго и третьего столбца

d = a.copy()            # Новый объект массива с новыми данными

```

---

В библиотеке `numpy` есть специальный модуль `numpy.linalg`, позволяющий реализовать множество операций из линейной алгебры. В таблице ниже приведены некоторые из них.

Атрибуты	Описание
<code>linalg.matrix_power(M, n)</code>	возводит матрицу в степень n
<code>linalg.norm(x[, ord, axis])</code>	норма вектора или оператора
<code>linalg.det(a)</code>	определитель
<code>linalg.solve(a, b)</code>	решает систему линейных уравнений $Ax = b$
<code>linalg.tensorsolve(a, b[, axes])</code>	решает тензорную систему линейных уравнений $Ax = b$
<code>linalg.lstsq(a, b[, rcond])</code>	метод наименьших квадратов
<code>linalg.inv(a)</code>	обратная матрица

Далее мы рассмотрим конкретный пример работы с массивами, используя библиотеку `numpy`. Наша задача заключается в анализе двух массивов данных, которые содержат информацию о росте и возрасте определённой группы людей.

---

```

heights = [189, 170, 189, 163, 183, 171, 185, 168, 173, 183, 173, 173, 175,
178, 183, 193, 178, 173, 174, 183, 183, 180, 168, 180, 170, 178, 182, 180, 183,
178, 182, 188, 175, 179, 183, 193, 182, 183, 177, 185, 188, 188, 182, 185, 191]

ages = [57, 61, 57, 57, 58, 57, 61, 54, 68, 51, 49, 64, 50, 48, 65, 52, 56,
46, 54, 49, 51, 47, 55, 55, 54, 42, 51, 56, 55, 51, 54, 51, 60, 62, 43, 55,
56, 61, 52, 69, 64, 46, 54, 47, 70]

```

---

Отметим, что для обработки данных можно использовать непосредственно сам возможности самого языка Python. Ниже приведён пример использования цикла и условий для выборки определённой группы людей с ростом выше 188 см.

---

```

cnt = 0
for height in heights:
    if height > 188:

```

```
    cnt +=1
print(cnt)
```

---

А теперь подробно рассмотрим, как можно упростить обработку данных, если использовать модуль **numpy**.

---

```
import numpy as np

heights = [189, 170, 189, 163, 183, 171, 185, 168, 173, 183, 173, 173, 175,
178, 183, 193, 178, 173, 174, 183, 183, 180, 168, 180, 170, 178, 182, 180, 183,
178, 182, 188, 175, 179, 183, 193, 182, 183, 177, 185, 188, 188, 182, 185, 191]

ages = [57, 61, 57, 57, 58, 57, 61, 54, 68, 51, 49, 64, 50, 48, 65, 52, 56,
46, 54, 49, 51, 47, 55, 55, 54, 42, 51, 56, 55, 51, 54, 51, 60, 62, 43, 55,
56, 61, 52, 69, 64, 46, 54, 47, 70]

# Преобразование типа данных
heights_arr = np.array(heights)
ages_arr = np.array(ages)

# Вывод на экран кол-во элементов каждого массива
print(heights_arr.size)
print(ages_arr.size)

# Вывод на экран размера массива
print(heights_arr.shape)
print(ages_arr.shape)

print(heights_arr.dtype)
print(ages_arr.dtype)
```

---

Когда данные загружены, приведены к нужному формату и известен их размер, то можно приступать к манипуляции над ними. Все методы **numpy** могут быть использованы в нашем случае.

---

```
# Изменение формы массива (строка -> столбец)
heights_arr = heights_arr.reshape(45,1)
ages_arr = ages_arr.reshape((45,1))

# Изменение формы массива (столбец -> строка)
heights_arr = heights_arr.reshape(1,45)
ages_arr = ages_arr.reshape((1,45))

# Объединение массивов heights_arr и ages_arr
height_age_arr = np.hstack((heights_arr, ages_arr))

# Переназначение элементов массива
heights_and_ages_arr[0, 3] = 165
heights_and_ages_arr[:, 0] = [190, 58]
new_record = np.array([[180, 183, 190], [54, 50, 69]])
heights_and_ages_arr[:, 42:] = new_record
```

```
# Связывание массивов heights_arr и ages_arr
height_age_arr = np.concatenate((heights_arr, ages_arr), axis=1)

# Применение математических операций (*, /, +, -)
height_age_arr[:,0]*0.0328084

# Применение методов numpy
height_age_arr.sum()
height_age_arr.sum(axis=0)

# Использование логических условий
height_age_arr[:, 1] < 55
height_age_arr[:, 1] == 51

# Использование маски для среза данных
mask = height_age_arr[:, 0] >= 182
mask.sum()
mask = (height_age_arr[:, 0]>=182) & (height_age_arr[:,1]<=50)
height_age_arr[mask, ]
```

---

Вернёмся в начало и посмотрим, как можно было решить задачу по выбору людей из массива, рост которых больше 188 см. При этом, мы не будем использовать стандартный цикл **for**.

```
import numpy as np
heights_arr = np.array(heights)
print((heights_arr > 188).sum())
```

---

Программа стала проще для понимания. В следующей части мы рассмотрим, каким образом можно ещё загружать данные в программу, приводить к удобному формату и проводить анализ.

## 2.2. Подключение и работа с библиотекой pandas

**Библиотека pandas** – это библиотека языка Python, позволяющая обрабатывать многомерные массивы, которые могут содержать различные типы данных. Возможность обработки различных типов данных отличает **pandas** от **numpy**. Библиотека представляет возможность работы с данными двух форматов: Series (одномерный массив) и DataFrame (многомерный массив). Для подключения модуля к программе используются инструкция **import**.

**Series** – это одномерный массив, который может содержать данные любого типа (целочисленные, строковые, с плавающей точкой, объекты Python и т.д.), аналогично столбцу в электронной таблице excel. Метки осей в совокупности называются индексами.

---

```
import pandas as pd
```

```
# Создание Series из словаря
pd.Series({'a': 1, 'b': 2, 'c': 3})

# Создание Series из списка
pd.Series([1, 2, 3], index=['a', 'b', 'c'])

# Создание Series, используя numpy
pd.Series(np.array([1, 2, 3]), index=['a', 'b', 'c'])
series['a']
```

---

Доступ к значению по его индексу, а не целочисленному значению, очень удобен, когда набор данных состоит из большого кол-ва данных. Series может быть структурной единицей для создания DataFrame.

**DataFrame** – это двумерный массив, который может содержать данные любого типа (целочисленные, строковые, с плавающей точкой, объекты Python и т.д.), аналогично столбцам в электронной таблице excel. Метки осей в совокупности называются индексами. DataFrame можно создать из различных элементов, например, из словаря. В данном случае DataFrame можно рассматривать как совокупность отдельных Series.

---

```
import pandas as pd

# Создание DataFrame из словаря
block = {
    'block_1':[1, 2, 3],
    'block_2':[4, 5, 6]
}
# pd.DataFrame(data = data, columns = columns)
df = pd.DataFrame(block, index=["a", "b", "c"])
print(df)
```

---

DataFrame можно загрузить в программу из внешнего файла. Для этого можно использовать, например, метод **read\_csv**. Загрузка других форматов файлов подробно рассмотрена в документации к библиотеке.

---

```
import pandas as pd

# Загрузка таблицы из txt, lvm, csv и xlsx
df1 = pd.read_csv('data.txt', sep='\t')
df2 = pd.read_csv('data.lvm', sep='\t')
df3 = pd.read_excel('data.xlsx')
df4 = pd.read_csv('data.csv')
```

---

В таблице ниже приведены некоторые атрибуты объектов DataFrame.

Атрибуты	Описание
<b>df.columns</b>	вывод названий столбцов
<b>df.size</b>	кол-во элементов в таблице
<b>df.shape</b>	размеры таблицы

<code>df.dtypes</code>	получение информации о типах данных в таблице
<code>df.info()</code>	просмотр сводной информации о таблице

Если нет необходимости просмотра таблицы целиком, то можно воспользоваться двумя методами: `head()` и `tail()`. Первый метод позволяет показать первые 5 строк таблицы, а второй последние 5 строк. Количество отображаемых строк можно изменить, указав нужное кол-во в скобках.

```
import pandas as pd

numbers = [[1, 8, 15],
           [2, 9, 16],
           [3, 10, 17],
           [4, 11, 18],
           [5, 12, 19]]

header = ['first_col', 'second_col', 'third_col']

df = pd.DataFrame(data = numbers, columns = header)

print(df.head(3)) # 3 первых строки
print(df.tail(3)) # 3 последних строки
```

## 2.3. Срезы данных и логическое индексирование

Иногда возникает ситуация, когда нет необходимости использовать всю таблицу целиком. Требуется использование определённых столбцов, строк или только определённых значений. В этом случае будет полезным использование срезов данных. Мы рассмотрим два типа срезов: с использованием метода `loc()` и логическое индексирование. Метод `loc()` даёт доступ к данным по строке и столбцу. Рассмотрим несколько примеров.

```
# df.loc[строка, столбец]

df.loc[7, 'column']                                # Одна ячейка
df.loc[:, 'column']                                 # Один столбец
df.loc[:, ['column_1', 'column_3']]                 # Несколько столбцов
df.loc[:, 'column_2': 'column_4']                  # Несколько столбцов подряд
df.loc[1]                                         # Одна строка
df.loc[1:]                                       # Все строки, начиная с заданной
df.loc[:3]                                       # Все строки до заданной
df.loc[2:5]                                     # Несколько строк подряд (срез)
```

Альтернативой метода `loc()` является логическое индексирование. Метод является сложным для понимания, но более универсальным при обработке данных и извлечении нужных значений из таблицы.

```
# Строки, удовлетворяющие условию
df.loc[df['column'] == 'X']
df[df['column'] == 'X']
```

```

# Столбец, удовлетворяющий условию
df.loc[df.loc[:, 'column'] == 'X']['column']
df[df['column'] == 'X']['column']

# С применение метода
df.loc[df.loc[:, 'column'] == 'X']['column'].count()
df[df['column'] == 'X']['column'].count()

```

---

Аналогичный подход срезов данных можно использовать, если необходимо работать с Series. Разница заключается в том, что в случае с Series срез делается только по строкам.

## 2.4. Предобработка данных

Предварительная обработка данных выполняет коррекцию названия столбцов в таблице, поиск пропусков, удаление или замену некорректных значений, поиск и удаление дубликатов. Другими словами, обработка позволяет осуществить чистку таблицы от значений, которые могут привести к ошибке при выполнении анализа данных. Любой анализ данных начинается именно с предобработки данных.

В таблице ниже приведены некоторые методы для подготовки данных.

Метод	Описание
<code>df.set_axis()</code>	изменение названий столбцов
<code>df.isnull(), isna()</code>	определение пропущенных значений
<code>df.fillna()</code>	заполнение пропущенных значений
<code>df.dropna()</code>	удаление пропущенных значений
<code>df.drop_duplicates()</code>	удаление дубликатов
<code>df.replace()</code>	замена значений в столбце или таблице

Рассмотрим примеры использования данных методов.

```

# Назначение новых названий для столбцов
df.set_axis(['first', 'second'], axis = 'columns', inplace = True)

# Поиск и сумма всех пустых значений в таблице
df.isnull().sum()
df.isna().sum()

# Замена всех пустых значений на нули
df = df.fillna(0)

# Удаление указанных столбцов из таблицы
df.dropna(subset = ['first', 'second'], inplace = True)

# Поиск и сумма всех дубликатов в таблице
df.duplicated().sum()

# Удаление дубликатов и новое индексирование данных
df.drop_duplicates().reset_index(drop = True)

```

```
# Поиск уникальных значений в таблице
df['column'].unique()

# Сортировка значений по заданному столбцу
df.sort_values(by=['column'])

# Количество уникальных значений в столбце
df['column'].value_counts(dropna=False)
df['column'].value_counts(normalize=True) # значения разделены на сумму

# Замена значения f_value на s_value
df.replace('f_value', 's_value')
```

---

В качестве заполнителей пустых или ошибочных значений чаще всего будут заменены плейсхолдеры типа n/a, na, NA, и N.N. либо NN. Предобработка данных позволяет выявить данные значения и заменить их на те, которые будут обработаны программой без ошибок.

## 2.5. Изменение типов данных

Изменение типа данных используется тогда, когда программе не удалось правильно определить тип данных в столбце или таблице. Числовые значения можно переводить в строки и обратно. Строки с датой можно переводить в специальный формат, который позволяет отдельно использовать время (часы, минуты, секунды) и дату (год, месяц, число). Подобный процесс разложения удобен при работе с большими массивами временных рядов. Ниже представлены 3 метода, позволяющие изменять тип данных: to\_numeric(), astype(), to\_datetime().

---

```
import pandas as pd

# Создание таблицы из внешнего файла excel
df = pd.read_excel('file.xlsx', sheet_name='Лист1')

# 1. Перевод значений столбца из типа str в вещественный float

pd.to_numeric(df['column'], errors='raise')
# если errors='raise', то в случае ошибок преобразование будет прервано
pd.to_numeric(df['column'], errors='coerce')
# если errors='coerce', то значения заменяются на NaN
pd.to_numeric(df['column'], errors='ignore')
# если errors='ignore', то значения игнорируются, но остаются

# 2. Изменение типа данных на заданный (int32/64, float32/64, str)
df['column'].astype('type')
df['column'] = df['column'].astype('type')

# 3. Перевод из строки в дату и время
pd.to_datetime(df['date_time_column'], format='%d.%m.%Y %H:%M:%S')
***
```

Формат строится с использованием следующих обозначений:

- %d - день месяца (от 01 до 31)

- %m - номер месяца (от 01 до 12)
  - %Y - год с указанием столетия (например, 2019)
  - %H - номер часа в 24-часовом формате
  - %I - номер часа в 12-часовом формате
  - %M - минуты (от 00 до 59)
  - %S - секунды (от 00 до 59)
  - %b - сокращённое название месяца
  - %B - полное название месяца
  - %a - день недели (коротко)
  - %A - день недели (полностью)
  - %p - до полудня (AM)
- \*\*\*

```
# Получение отдельных данных из столбца с датой и временем
```

```
pd.DatetimeIndex(df['time_time_column']).year # год  
pd.DatetimeIndex(df['time_time_column']).month # месяц  
pd.DatetimeIndex(df['time_time_column']).day # день  
pd.DatetimeIndex(df['time_time_column']).hour # час  
pd.DatetimeIndex(df['time_time_column']).minute # минут  
pd.DatetimeIndex(df['time_time_column']).second # секунд
```

---

Для получения дополнительной информации о способах преобразования типов данных и особенностях их использования следует обратиться к документации библиотеки pandas на сайте разработчика.



## Задания для самостоятельной работы

### 1. Подключение и работа с библиотекой numpy

#### 1.1. Даны два массива:

---

```
import numpy as np

a = np.array([[2.3, 5.1, 4.7],
              [3.5, 6.7, 1.5],
              [8.4, 3.1, 9.2]])

b = np.array([[4.3, 8.1, 6.1],
              [3.7, 6.2, 1.5],
              [2.4, 5.7, 4.7]])
```

---

Необходимо выполнить следующие операции:

- 1) выведите на экран количество строк и количество столбцов массива a и массива b;

---

```
>>> %Run example_practice_01.py
```

Массив a содержит \_ строк и \_ столбцов.  
Массив b содержит \_ строк и \_ столбцов.

---

- 2) для a и b найдите сумму, разность, произведение и частное элементов массивов;
- 3) найдите новый массив, элементами которого являются элементы массива a после возведения в квадрат;
- 4) найдите новый массив, элементами которого являются элементы массива b после нахождения остатка от деления на 2;
- 5) найди cos() каждого элемента массива a, найдите sin() каждого элемента массива b, а потом найдите сумму двух новых массивов.

**1.2.** Дан массив:

---

```
a = np.array([1.2, 2.4, 3.6, 3.5, 6.7, 1.5, 8.4, 3.1, 9.2])
```

---

Необходимо выполнить следующие операции:

- 1) выведите на экран количество строк и количество столбцов массива а;
- 2) преобразуйте массив таким образом, чтобы получился массив из 9 строк и 1 столбца;
- 3) преобразуйте массив таким образом, чтобы получился массив из 3 строк и 3 столбцов;
- 4) для массива из пункта 3 необходимо найти и вывести на экран максимальное значение в строке (столбце);
- 5) для массива из пункта 3 необходимо найти и вывести на экран минимальное значение в строке (столбце);
- 6) для массива из пункта 3 необходимо найти и вывести на экран сумму элементов строк (столбцов).

**1.3.** Даны два массива:

---

```
a = np.array([[4, 2], [9, 1]])  
b = np.array([[5, 3], [2, 5]])
```

---

Необходимо выполнить следующие операции:

- 1) объедините два массива таким образом, чтобы получился массив из 4 строк и 2 столбцов;
- 2) найдите срез массива из пункта 1, который содержит элементы строк [1-3] (включительно) для [1] столбца;
- 3) для массива из пункта 2 вычислить сумму элементов, максимальное и минимальное значение;
- 4) объедините два массива таким образом, чтобы получился массив из 2 строк и 4 столбцов;
- 5) найдите срез массива из пункта 4, который содержит элементы 1 строки и [1-3] (включительно) столбца;
- 6) для массива из пункта 5 вычислить сумму элементов, максимальное и минимальное значение.

**1.4.** Даны система линейных алгебраических уравнений:

$$\begin{cases} 5x + 4y = 14 \\ 2x - 6y = -2 \end{cases}$$

Необходимо найти решение для данной системы, используя модуль **linalg** из библиотеки **numpy**.

**1.5.** Даны следующие матрицы:

$$\begin{pmatrix} 2 & 8 \\ 1 & -6 \end{pmatrix}, \begin{pmatrix} 3 & 2 & 7 \\ 4 & 1 & 8 \\ 6 & 3 & 7 \end{pmatrix}, \begin{pmatrix} 4 & 3 & 2 & 7 \\ 6 & 1 & 1 & -2 \\ 7 & 5 & 8 & 1 \\ 9 & 5 & -3 & -5 \end{pmatrix}.$$

Используя модуль **linalg** из библиотеки **numpy** необходимо найти.

- 1) транспонированную матрицу;
- 2) обратную матрицу;
- 3) определитель, полученный из элементов матриц;
- 4) найдите норму векторов, полученных из строк матрицы 3x3;
- 5) найдите норму векторов, полученных из столбцов матрицы 4x4.

Для выделения строк и столбцов можно использовать метод `split(a, 3, axis=0)`.

**1.6.** Создайте массив `array`, который содержит 100 случайных элементов из целых чисел в диапазоне от 0 до 100.

```
import numpy as np
import random

a = np.array([random.randint(0, 100) for i in range(0, 100, 1)])
```

Найдите количество элементов массива, которые больше 50. Решить задачу необходимо двумя способами: с использованием цикла `for` и без использования цикла.

## 2. Подключение и работа с библиотекой **pandas**

**2.1.** Создайте `DataFrame`, который состоит из 3-ёх столбцов. Столбцы содержат информацию: рост (см), масса (кг) и возраст человека. При создании таблицы можно использовать готовые списки данных:

```
import pandas as pd
```

```
heights = [188, 172, 187, 161, 183, 172, 185, 163, 173, 183, 174, 174, 175,
178, 183, 195, 178, 173, 174, 183, 174, 181, 162, 180, 170, 175, 182, 180, 183,
178, 182, 188, 175, 179, 184, 193, 182, 183, 175, 185, 182, 183, 156, 185, 199]

weights = [86, 71, 89, 64, 83, 71, 86, 68, 73, 84, 73, 72, 75, 78, 85, 93, 78,
70, 74, 80, 83, 82, 68, 80, 73, 78, 82, 81, 83, 79, 82, 86, 76, 77, 80, 103,
82, 83, 77, 89, 80, 85, 82, 85, 110]

ages = [58, 61, 54, 54, 58, 57, 63, 54, 37, 51, 34, 26, 50, 48, 45, 52, 56,
46, 54, 28, 52, 47, 55, 55, 54, 42, 51, 56, 55, 51, 54, 32, 21, 62, 43, 55,
56, 61, 53, 22, 64, 45, 54, 47, 38]
```

---

Названия столбцов: heights, weights и ages. После создания таблицы выведите на экран сводную информацию методом info() и определите тип данных для каждого столбца. Выведите на экран первые и последние 3 строки таблицы.

---

```
>>> %Run example_practice_01.py
```

```
    heights  weights  ages
0        188       86      58
1        172       71      61
2        187       89      54

    heights  weights  ages
42       156       82      54
43       185       85      47
44       199       110     38
```

---

**2.2.** Для выполнения следующего задания используйте таблицу, которая была создана в задании (2.1). Необходимо известными методами для работы с таблицами pandas найти для каждого столбца следующие величины:

- 1) максимальное и минимальное значение в столбце;
- 2) среднее значение для каждой величины (используя метод mean());
- 3) среднее значение для каждой величины (без использования метода mean());
- 4) значение медианы для каждой величины;
- 5) стандартное отклонение в каждом столбце.

**2.3.** Для выполнения следующего задания используйте таблицу, которая была создана в задании (2.1). Необходимо методом describe() найти минимальное, максимальное, среднее и стандартное отклонение величин в каждом столбце. Результаты сравнить с ответами в задаче (2.2).

### 3. Срезы данных и логическое индексирование

**3.1.** Для выполнения следующего задания используйте таблицу, которая была создана в задании (2.1). На основе существующей таблицы создайте новую,

используя срезы данных. В новой таблице должны быть все столбцы. Необходимо выбрать первые 40 строк.

**3.2.** Из полученной таблицы в (3.1), используя логическое индексирование, необходимо выбрать те строки, для которых выполняется условие `heights > 180` см. На экран нужно вывести количество строк, удовлетворяющих данному условию.

**3.3.** Из полученной таблицы в (3.1), используя логическое индексирование, необходимо выбрать те строки, для которых выполняется условие `weights < 80` кг. На экран нужно вывести количество строк, удовлетворяющих данному условию.

**3.4.** Из полученной таблицы в (3.1), используя логическое индексирование, необходимо выбрать те строки, для которых выполняется условие `30 < ages < 50`. На экран нужно вывести количество строк, удовлетворяющих данному условию.

**3.5.** Из полученной таблицы в (3.1), используя логическое индексирование, необходимо выбрать те строки, для которых выполняется двойное условие: `heights > 170` см и `weights > 80` кг. Для новой таблицы рассчитайте минимальное, максимальное, среднее и стандартное отклонение величин в каждом столбце. Полученные результаты выведите на экран.

#### **4. Предобработка данных: пропуски, некорректные данные и дубликаты**

**4.1.** Для выполнения следующего задания необходимо загрузить датасет из интернета. Обратите внимание на разделитель данных.

---

```
import pandas as pd  
  
df = pd.read_csv('precious_metal.csv', sep=';')
```

---

В датасете представлена динамика изменения цен за грамм драгоценных металлов: золото, серебро, платина и палладиум. Загрузите набор данных в программу и методом `info()` определите тип данных в каждом столбце таблицы. Выведите таблицу на экран.

Примечание: все последующие задания в (4) выполняются с обработкой данных загруженного датасета.

**4.2.** Обратите внимание на название столбцов в таблице: GOLDA, Silver, platinu, Palla, Date. Данные названия в программе не будут определены как ошибочные, но правилами рекомендуется привести названия столбцов к следующему виду: gold, silver, platinum, palladium, date. Методом `set_axis()` измените названия столбцов.

**4.3.** Найдите количество пропущенных данных в каждом столбце таблицы. Замените все пропущенные значения на нули. Выведите новую таблицу на экран.

**4.4.** Обратите внимание, что при загрузке данных программа считает данные как объект. Для обработки данных различными методами необходимо преобразовать объект в число. В данном случае столбцы содержат числа, у которых в качестве разделителя используется запятая. Замените во всех числовых столбцах запятую на точку функцией apply(). Выведите новую таблицу на экран.

---

```
def replace_symbol(x):
    x = str(x)
    x = float(x.replace(',', '.'))
    return x

df['gold'] = df['gold'].apply(replace_symbol)

# или используйте lambda-функцию

df['gold'] = df['gold'].apply(lambda x: float(str(x).replace(',', '.')))
```

---

**4.5.** Найдите среднее значение цены в каждом столбце. Сохраните полученные переменные как gold\_mean, silver\_mean, platinum\_mean, palladium\_mean. Замените во всех столбцах нули на среднее значение цен методом apply(). Выведите новую таблицу на экран.

**4.6.** Создайте новые признаки: df['gold\_usd'] или df['gold\_eu'] и т.д.. В данных столбцах содержится информация о цене на металл в валюте (доллар, евро). Для заполнения новых столбцов необходимо использовать данные исходных столбцов и актуальный курс рубля по отношению к доллару и евро (по данным ЦБ РФ). Новая таблица должна содержать девять столбцов. Столбец с датой нужно исключить.

---

```
>>> %Run example_practice_01.py
```

	gold	silver	platinum	...	silver_usd	platinum_usd	palladium_usd
0	4784.40	58.70	2183.31	...	0.74	27.46	71.42
1	4759.17	60.67	2195.94	...	0.76	27.62	71.22
2	4729.52	60.93	2179.19	...	0.77	27.41	73.11
3	4667.69	59.68	2145.92	...	0.75	26.99	73.12
4	4673.64	60.85	2189.85	...	0.77	27.55	0.00
..	...	...	...	...	...	...	...
124	4023.90	35.70	1794.33	...	0.45	22.57	54.40
125	4040.41	35.85	1795.58	...	0.45	22.59	53.10
126	4014.11	35.86	1819.12	...	0.45	22.88	58.56
127	4041.80	35.93	1847.95	...	0.45	23.24	58.35

---

Сохраните полученный датасет под названием precious\_metal\_new.csv.

## 5. Изменение типов данных

**5.1.** Загрузите данные из файла precious\_metal\_new.csv в программу. Измените порядок столбцов так, чтобы столбец date был последним.

---

```
import pandas as pd

df = pd.read_csv('precious_metal_new.csv')

df = df.loc[:,['gold', 'gold_usd', 'silver', 'silver_usd',
               'platinum', 'platinum_usd', 'palladium', 'palladium_usd',
               'date']]
```

---

**5.2.** Преобразуйте столбец с датой из object в тип данных datetime.

---

```
df['date'] = pd.to_datetime(df['date'], format='%d.%m.%Y %H:%M')
```

---

Обратите внимание, что столбец времени не имеет секунд.

**5.3.** Создайте новый признак, который будет содержать информацию о месяце. Назовите новый признак df['month']. Поскольку информация будет храниться в виде числа, то вам нужно преобразовать число в строку. Для этого напишите функцию, которая выполняет замену числа от 1 до 12 на строку с названием месяца в году: January, February, March, April, May, June, July, August, September, October, November, December. Полученную таблицу выведите на экран.

**5.4.** Используя логическое индексирование и категориальный признак df['month'], выполните следующий анализ:

- 1) определите среднюю стоимость всех драгоценных металлов в рублях в мае (результат в виде словаря, где ключ – металл, а значение – средняя стоимость);
- 2) определите среднюю стоимость всех драгоценных металлов в рублях в октябре (результат в виде словаря, где ключ – металл, а значение – средняя стоимость);
- 3) рассчитайте изменение средней стоимости драгоценных металлов в % за июнь и июль (результат в виде списка).

**5.5.** Выберите один из признаков в таблице df и выполните его нормализацию. Для стандартной нормализации используйте формулу:

$$y = \frac{x - x_{\min}}{x_{\max} - x_{\min}},$$

где  $x_{\min}$  - минимальное значение в столбце,  $x_{\max}$  - максимальное значение в столбце. Для min-max нормализации используйте формулу:

$$y = \frac{x - x_{\text{mean}}}{x_{\text{std}}},$$

где  $x_{\text{mean}}$  - среднее значение в столбце,  $x_{\text{std}}$  - стандартное отклонение.



### 3. Визуализация данных

Визуализация данных – ещё один инструмент для эффективного анализа данных. Визуализация данных позволяет выявить определённые закономерности и эффективно передать информацию о данных. В этой части мы рассмотрим применение библиотеки **matplotlib**, одного из самых популярных инструментов визуализации данных, работающий с массивами **numpy** и таблицами **pandas**.

---

```
# Подключение библиотеки matplotlib
import matplotlib as mpl

# Подключение библиотеки matplotlib.pyplot
import matplotlib.pyplot as plt
```

---

Pyplot – это набор функций, который позволяет строить графики в Python аналогично графикам в математическом пакете Matlab. Каждая функция способна вносить изменения в график, например, создает область построения на фигуре, строит оси координат, определяет заголовок, метки и легенду.

#### 3.1. Библиотека matplotlib

Для построения любых графиков в **matplotlib** сначала создают графическое окно (метод `figure()`) и объект оси (метод `axes()`). Для вывода окна на экран используют метод `show()`. Рисунок должен содержать все объекты, включая оси, графики, тексты и метки.

---

```
import matplotlib.pyplot as plt
fig = plt.figure()
ax = plt.axes()
plt.show()
```

---

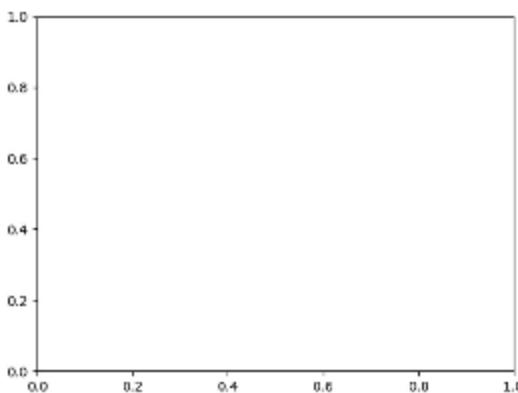


Рис.3.1. Графическое окно в matplotlib.pyplot (без графика)

Далее мы рассмотрим, каким образом можно настроить результаты отображению графика на экране, используя методы из модуля pyplot.

### 3.2. Линейный график

Для построения линейного графика мы будем использовать тригонометрическую функцию  $\cos(x)$  и метод `linspace()` из библиотеки numpy.

---

```
import matplotlib.pyplot as plt
import numpy as np

x_data = np.linspace(0, 20, 1000) # от 0 до 20 с шагом 0.01
y_data = np.cos(x_data)

fig = plt.figure()
ax = plt.axes()
ax.plot(x_data, y_data)
plt.show()
```

---

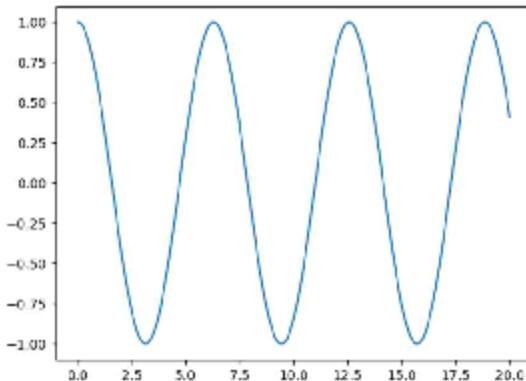


Рис.3.2. График  $y = \cos(x)$

Запись можно немного сократить, используя синтаксис `plt.plot()`. В этом случае в явном виде указывать графическое окно и оси нет необходимости.

---

```
import matplotlib.pyplot as plt
import numpy as np

x_data = np.linspace(0, 20, 1000)
y_data = np.sin(x_data)

plt.plot(x_data, y_data)
plt.show()
```

---

Важным компонентом любой фигуры является заголовок графика. Задача заголовка состоит в том, чтобы точно передать назначение графика. Ещё одним важным структурным элементом графика является метка оси. Можно указывать метки для оси абсцисс, оси ординат и заголовка с помощью plt.xlabel(), plt.ylabel() и plt.title().

---

```
import matplotlib.pyplot as plt
import numpy as np

x_data = np.linspace(0, 20, 1000)
y_data = np.cos(x_data)

plt.plot(x_data, y_data)
plt.xlabel('x_data')
plt.ylabel('y_data')
plt.title('function cos(x)')
plt.show()
```

---

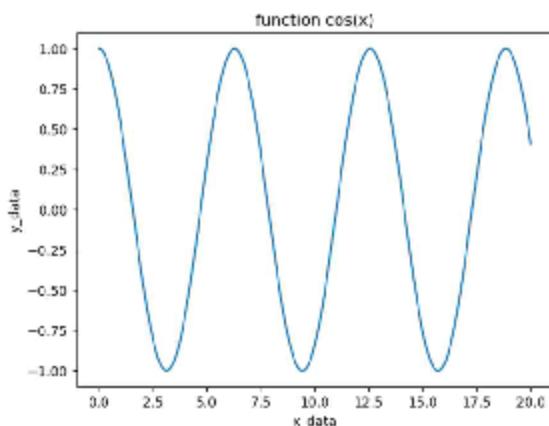


Рис.3.3. График  $y = \cos(x)$  с указанием заголовка и меток осей

Можно также установить пределы осей с помощью plt.xlim() и plt.ylim(), соответственно. Иногда возникает необходимость сравнения нескольких наборов данных на одном графике. Можно построить несколько линейных графиков на одной и той же фигуре. В этом случае функция plot() может быть вызвана несколько раз.

```

import matplotlib.pyplot as plt
import numpy as np

x_data = np.linspace(0, 20, 1000)
y_1_data = np.cos(x_data)
y_2_data = np.sin(x_data)

plt.plot(x_data, y_1_data, color='r', linestyle = '--')
plt.plot(x_data, y_2_data)

plt.xlabel('x_data')
plt.ylabel('y_data')
plt.title('function')
plt.show()

```

---

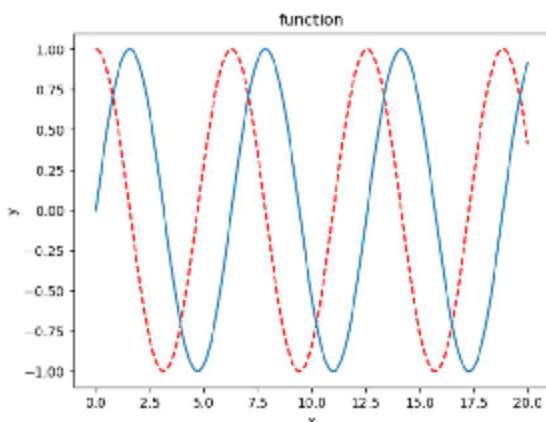


Рис.3.4. Построение нескольких фигур на одном графике

Для получения более подробной информации об настройки цвета графика (атрибут color) и стилей линий (linestyle) в matplotlib необходимо обратиться к документации [https://matplotlib.org/2.0.2/api/colors\\_api.html](https://matplotlib.org/2.0.2/api/colors_api.html).

### 3.3. Точечный график

Другим простым примером построения графика является точечный график. В этом случае отдельные значения не будут соединены отрезками. Точечный график удобен, если набор данных имеет сильный разброс. Рассмотрим несколько примеров построения точечных графиков. Обработка данных может быть выполнена модулем numpy.

---

```

# Построение графика с использованием модуля numpy
import matplotlib.pyplot as plt
import numpy as np

x_data = np.linspace(0, 20, 1000)
y_data = np.cos(x_data)
plt.scatter(x_data, y_data, color='r', marker='>')
plt.xlabel('x')
plt.ylabel('y')
plt.title('function')
plt.show()

```

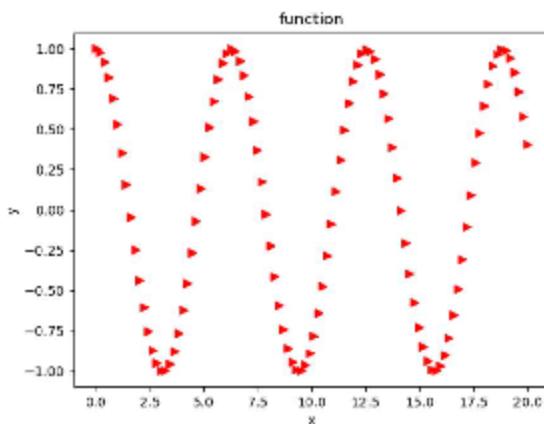


Рис.3.5. Построение графика с использованием модуля питчу

Библиотека pandas также позволяет использовать отдельные столбцы таблицы для построения графика функции.

---

```
# Построение графика с использованием модуля pandas
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

numbers = [[1, 5], [2, 3], [3, 6], [4, 7], [5, 4]]

header = ['x_data', 'y_data']

df = pd.DataFrame(data = numbers, columns = header)

plt.scatter(df['x_data'], df['y_data'], marker='o')
# df.plot(kind = 'scatter', x = 'x_data', y = 'y_data', marker='o')
plt.show()
```

---

Точечный график позволяет проанализировать взаимосвязь между двумя объектами, в то время как линейная диаграмма даёт информацию о скорости изменения величины (наклон кривой).

### 3.4. Построение гистограмм и столбчатых диаграмм

Гистограмма – это диаграмма, состоящая из прямоугольников с площадью пропорциональной частоте возникновения переменной. Рассмотрим на примере анализа двух массивов данных, которые содержат информацию о росте и возрасте определённой группы людей (см. раздел 2.1).

---

```
# Построение гистограмм
import matplotlib.pyplot as plt
import numpy as np
heights = [-]
ages = [-]
```

```

# Создание общего графического окна с nrows = 1, ncols = 2
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(8, 4))

ax1.hist(heights, bins = 10, color='b')
ax2.hist(ages, bins = 10, color='r')

plt.show()

```

---

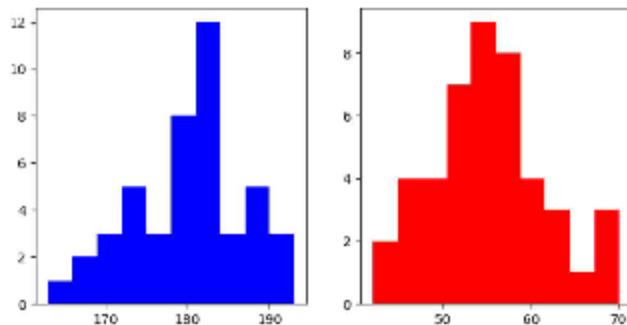


Рис.3.6. Построение гистограммы для двух типов данных: рост и возраст

Гистограмма показывает частотное распределение набора непрерывных данных, позволяя проверять данные на их форму, выбросы и асимметрию. Попробуйте самостоятельно проанализировать эти две диаграммы и определить возраст и рост, которые чаще всего встречаются в двух предложенных выборках. Здесь начинается путь настоящего аналитика данных.

Очень часто гистограмму путают со столбчатым графиком. Столбчатые графики показывают распределение данных по группам в том случае, когда необходимо провести анализ категориальных данных. Рассмотрим пример для numpy.

---

```

import matplotlib.pyplot as plt
import numpy as np

x = np.arange(1, 8)
y = np.random.randint(1, 20, size = 7)
plt.style.use('ggplot')
plt.bar(x, y, color='b')
plt.show()

```

---

Рассмотрим пример для pandas.

---

```

# Определение уникальных значений в столбце name
name_cnt = df['name'].value_counts()

plt.style.use('ggplot')
name_cnt.plot(kind ='bar')
plt.show()

```

---

В результате выполнения программы мы получим столбчатую диаграмму, которая показывает зависимость уникальных имён в столбце от кол-ва раз, которое данное имя встречается.

### 3.5. Построение boxplot (ящик с усами)

Напоследок рассмотрим пример построения boxplot (ящик с усами), который позволяет извлекать много полезной информации из набора данных.

```
# Определение уникальных значений в столбце name
import matplotlib.pyplot as plt
import numpy as np

heights = [-] # people_dataset

plt.style.use('classic')
plt.boxplot(heights)
plt.show()
```

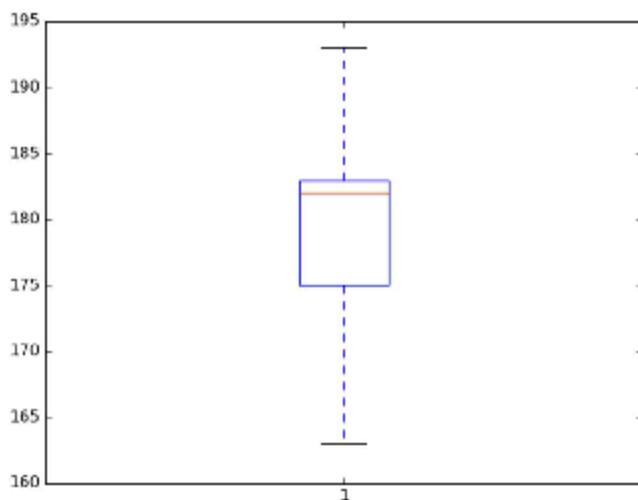


Рис.3.6. Построение boxplot для набора данных

Синий прямоугольник указывает на межквартильный диапазон (IQR), границу между первым и третьим квартилем. Т.е. 50% данных попадают в этот диапазон. Красная полоса показывает медиану. Черные “усы” сверху и снизу графика указывают на минимум и максимум величин, входящих в набор данных. Аналогичную информацию можно получить методом `describe()`.



## Задания для самостоятельной работы

### 1. Визуализация данных и библиотека matplotlib

**1.1.** Используя библиотеку matplotlib.pyplot и numpy, постройте линейный график тригонометрической функции  $\sin()$  на заданном интервале: 0 до 10 с шагом 0.01.

---

```
import matplotlib.pyplot as plt
import numpy as np

x_data = np.linspace(0, 20, 100)
y_data = np.sin(x_data)
```

---

График должен иметь заголовок, метку оси абсцисс, метку оси ординат, сетку ('--', серого цвета, толщина 0.8 ед.), легенду, размер figsize=(10, 5) и красный цвет линии толщиной 1.0 ед..

Примечание: для построения графика использовать два подхода: структурированный (ориентированный на структуру) и неструктурный (ориентированный на объект).

---

```
# Структурированный подход (ориентированный на структуру)
```

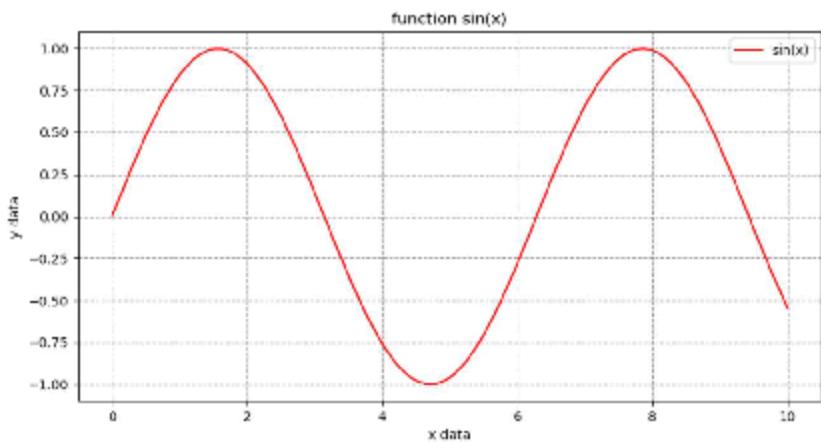
```
fig = plt.figure()
ax = plt.axes()
ax.plot(x_data, y_data)
plt.show()
```

---

```
# Неструктурированный подход (ориентированный на объект)
```

```
plt.plot(x_data, y_data)
plt.show()
```

---



**1.2.** Используя библиотеку matplotlib.pyplot и numpy, постройте точечный график тригонометрической функции cos() на заданном интервале: 0 до 10 с шагом 0.01.

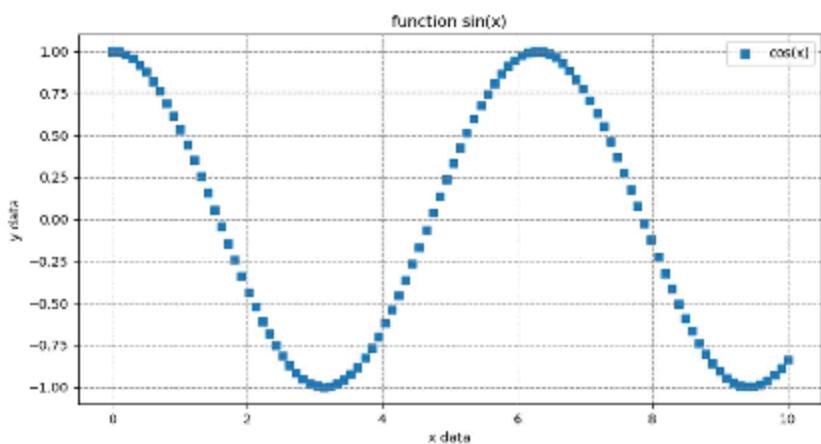
---

```
import matplotlib.pyplot as plt
import numpy as np

x_data = np.linspace(0, 10, 100)
y_data = np.cos(x_data)
```

---

График должен иметь заголовок, метку оси абсцисс, метку оси ординат, сетку ('--', серого цвета, толщина 0.8 ед.), легенду, размер figsize=(10, 5) и круглый маркер голубого цвета.



Примечание: для настройки формы маркера необходимо воспользоваться справкой на сайте разработчика библиотеки matplotlib: <https://matplotlib.org/>.

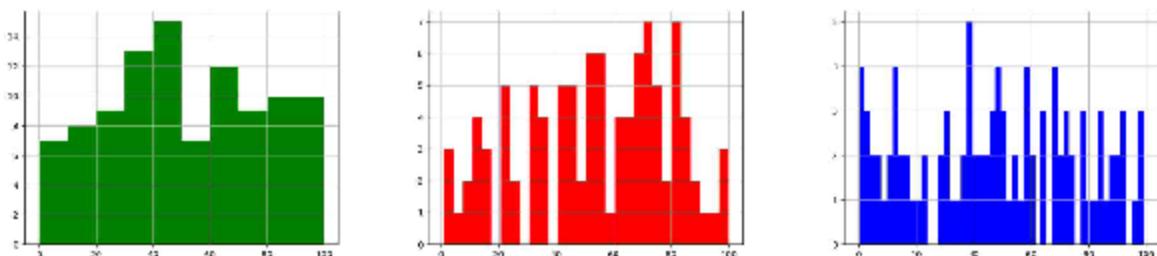
**1.3.** Создайте список из 100 целых чисел в диапазоне от 0 до 100.

---

```
import matplotlib.pyplot as plt
import random
x_data = [random.randint(0, 100) for x in range(100)]
```

---

Для полученной последовательности необходимо построить гистограмму распределения данных для bins=10, 30, 50.



Все гистограммы необходимо вывести в одном графическом окне (nrows=1, ncols=3), как это показано на рисунке.

**1.4.** Для выполнения следующего задания необходимо создать DataFrame, который содержит информацию: рост (см), масса (кг), возраст и пол человека (м/ж). Исходный датасет можно скачать здесь:

---

```
'https://github.com/mculab64/data_science/blob/main/ds_datasets/people.txt'
```

---

Используя библиотеку matplotlib.pyplot и pandas, постройте следующие графики:

1) зависимость heights/weights, heights/ages, weights/ages;

---

```
import matplotlib.pyplot as plt
import pandas as pd

df.plot(kind = 'scatter',
        x = 'heights',
        y = 'ages',
        title = 'heights/ages')
```

---

2) гистограммы распределения heights, weights, ages;

---

```
df['heights'].plot(kind='hist',
                    title = 'heights',
                    bins=5)
plt.show()
```

---

3) используя категориальный признак gender, постройте столбчатый график;

---

```
gender_cnt = df['gender'].value_counts()
party_cnt.plot(kind = 'bar')
plt.show()
```

---

**1.5.** Используя DataFrame из предыдущего задания, выполните следующее:

- 1) методом `describe()` найдите для каждого столбца (кроме `gender`): количество элементов в столбце, среднее значение, минимальное значение, максимальное значение, стандартное отклонение и межквартильное расстояние (разность данных между квантилями 25% и 75%);
- 2) для каждого столбца (кроме `gender`) постройте `boxplot`, определите наличие или отсутствие выбросов данных.

Примечание: под выбросами необходимо понимать значения, которые существенно выделяются из общей выборки.



## 4. Статистический анализ данных

### 4.1. Среднее значение, медиана, мода и диапазон

Для нахождения среднего значения необходимо найти сумму всех значений в наборе и разделить ее на количество значений. Существует специальный метод `mean()`, который позволяет рассчитывать среднее значение. Рассмотрим пример нахождения среднего значения.

```
# Нахождение среднего средствами numpy
import numpy as np

nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
nums_arr = np.array(nums)

print(np.mean(nums))

# Нахождение среднего средствами pandas
import pandas as pd

data = {'nums':[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]}
df = pd.DataFrame(data)

print(df['nums'].mean())
```

Далее рассмотрим понятие медиана. Медиана – это такое число, что половина элементов выборки меньше его, а вторая половина больше. Медиана является важным статистическим показателем. В том случае, если выборка имеет сильные выбросы, то для описания выборки лучше использовать медиану, а не среднее значение.

```
# Нахождение медианы средствами numpy
import numpy as np

nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
nums_arr = np.array(nums)

print(np.median(nums))
```

```
# Нахождение медианы средствами pandas
import pandas as pd

data = {'nums':[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]}
df = pd.DataFrame(data)

print(df['nums'].median())
```

---

Следующим рассмотрим статистический показатель мода. Мода – это значение, которое встречается наиболее часто в выборке значений. Для нахождения моды будем использовать библиотеку scipy.

```
# Нахождение моды средствами stats
from scipy import stats

data = [2, 2, 5, 4, 2, 6, 7, 8, 9, 5]

print(stats.mode(data))
```

---

В приведенном примере будут определена одна мода = 2.

Рассмотрим методы, которые позволяют определить минимальное значение min() и максимальное значение max().

```
# Нахождение min, max средствами numpy
import numpy as np

nums = [2, 2, 5, 4, 2, 6, 7, 8, 9, 5]
nums_arr = np.array(nums)

print(np.min(nums))
print(np.max(nums))

# Нахождение min, max средствами pandas
import pandas as pd

data = {'nums':[2, 2, 5, 4, 2, 6, 7, 8, 9, 5]}
df = pd.DataFrame(data)

print(df['nums'].min())
print(df['nums'].max())
```

---

Методы, которые были приведены для DataFrame, могут быть использованы и для объекта Series.

Если для аналитика данных определена задача группировки случайной величины, то в этом случае он может воспользоваться квантилями. **Квантиль** – это значение, которое заданная случайная величина не может превышать с заданной вероятностью (точка среза). Вероятность может быть задана в процентах, тогда квантиль называется процентилем или перцентилем.

Используя данное распределение, набор данных можно разделить на 4 группы, каждая из которых называется **квартиль**.

---

```
import pandas as pd

data = {'nums':[2, 2, 5, 4, 2, 6, 7, 8, 9, 5]}
df = pd.DataFrame(data)

# Деление на квартили
print(df['nums'].quantile([0.25, 0.5, 0.75, 1]))
```

---

Отметим, что медиана совпадает с одной из точек квантилей. Если рассмотреть методы median() и quantile(0.5), то результат работы программы будет одинаковый.

#### 4.2. Элементы теории вероятности: дисперсия и стандартное отклонение

В теории вероятности и статистике **дисперсия случайной величины** – это мера разброса случайной величины относительно её **математического ожидания**. Математическое ожидание случайной величины – это сумма произведений всех величин на вероятности этих значений. Иногда математическое ожидание путают со средним значением. Они могут совпадать лишь в том случае, если вероятности значений равны.

---

```
import numpy as np

nums = [2, 2, 5, 4, 2, 6, 7, 8, 9, 5]

print(np.var(nums))
print(np.std(nums))
```

---

На практике чаще всего используют не саму дисперсию, а корень квадратный из этой величины. Данный параметр называется **стандартным отклонением** или среднеквадратическим отклонением (СКО). СКО показывает, насколько значения случайной величины отличаются от среднего значения.

Метод describe() позволяет вывести почти всю сводную статистику рассмотренную выше, за исключением дисперсии. Кроме того, он подсчитывает все ненулевые значения каждого столбца.

---

```
import pandas as pd

data = {'nums':[2, 2, 5, 4, 2, 6, 7, 8, 9, 5]}
df = pd.DataFrame(data)

print(df['nums'].describe())
```

---

Функция `describe()` игнорирует нулевые значения, например, такие как "NaN" (not a number) и генерирует таблицу с описательной статистикой, которая содержит среднее, дисперсию (точнее стандартное отклонение) и форму (min, max значения и квантили) набора данных. Рассматриваемая функция может быть использована для анализа категориальных данных, которые хранятся в таблице в виде строки. Для анализа категориальных данных чаще используют метод `value_counts()`, показывающий все уникальные значения в столбце или таблице.

---

```
import pandas as pd

data = {'nums':['one', 'two', 'three', 'two', 'one', 'three', 'two', 'one',
'two']}
df = pd.DataFrame(data)

print(df['nums'].value_counts())
```

---

Сводная статистика предоставляет нам много информации о данных в доступном виде. Медиана является более надежной метрикой, чем среднее значение для непрерывных переменных, т.к. последняя чувствительна к критически большим выбросам.

### 4.3. Группировка и агрегирование данных

Сводная статистика дает хорошее общее представление о наборе анализируемых данных, но на практике очень часто приходится выполнять вычисления, имеющих связь с определённой категорией. Для анализа значений в этом случае мы можем использовать метод `groupby()`, выполняющий группировку данных. Операция `groupby` выполняется в три шага: разделение данных, применение методов и последующее объединение. Шаг разделения разбивает таблицу данных на несколько простых столбцов данных на основе значения указанного ключа. Шаг применения методов заключается в выполнении операции внутри каждого столбца данных по отдельности. Последний шаг объединяет модифицированные столбцы в одну большую таблицу. Рассмотрим простой пример группировки данных. В качестве исходных данных для анализа создадим таблицу, в которой будут представлены две категории мужчина (`man`) и женщина (`woman`) со своими признаками рост (`height`) и масса тела (`weight`).

---

```
import pandas as pd

data = {'human':['man', 'woman', 'man', 'man', 'woman', 'man', 'woman', 'man',
'man'],
'height':[178, 165, 190, 174, 160, 180, 170, 182, 170],
'weight':[70, 60, 90, 67, 58, 75, 65, 80, 65]}

df = pd.DataFrame(data)
```

```
print(df.groupby('human'))
print(df.groupby('human').mean())
```

---

В результате мы получим сгруппированную таблицу данных, в которой будет представлено среднее значение по росту и массе тела в группе мужчин и женщин. Метод mean() является примером использования функций для выполнения операций внутри отдельных столбцов. Практически все методы, которые мы рассмотрели при изучении библиотек pandas и numpy, могут быть использованы совместно с groupby().

Как поступать в том случае, если необходимо использовать несколько методов совместно с groupby(). Для этого была придуман специальный метод объединения agg(). Рассмотрим пример агрегации методов на группе мужчин и женщин.

---

```
import pandas as pd
import numpy as np

data = {'human': ['man', 'woman', 'man', 'man', 'woman', 'man', 'woman', 'man',
'man'],
        'height':[178, 165, 190, 174, 160, 180, 170, 182, 170],
        'weight':[70, 60, 90, 67, 58, 75, 65, 80, 65]}

df = pd.DataFrame(data)

print(df.groupby('human')['height'].agg([min, np.median, max]))
print(df.groupby('human')['weight'].agg([min, np.median, max]))
```

---

В результате выполнения программы были созданы две таблицы, в которых представлена минимальное, медиана и максимальное значение двух параметров всей группы людей: рост и масса тела. Обратите внимание, что методы указаны без круглых скобок.

#### 4.4. Взаимосвязь данных

Под корреляцией (корреляционной зависимостью) мы будем понимать статистическую взаимосвязь 2-ух или более случайных величин. При этом изменение одной из величин может сопутствовать изменению других случайных величин. Рассмотрим процесс корреляции двух величин группы людей: рост и масса тела. Для предварительной оценки будем использовать библиотеку matplotlib для построения графика.

---

```
import matplotlib.pyplot as plt
import pandas as pd

data = {'human': ['man', 'woman', 'man', 'man', 'woman', 'man', 'woman', 'man',
'man'],
        'height':[178, 165, 190, 174, 160, 180, 170, 182, 170],
        'weight':[70, 60, 90, 67, 58, 75, 65, 80, 65]}
```

---

```
df = pd.DataFrame(data)
plt.scatter(df['height'], df['weight'], alpha = 0.8)
# df.plot(x = 'weight', y = 'height', kind='scatter', grid=True)
plt.show()
```

---

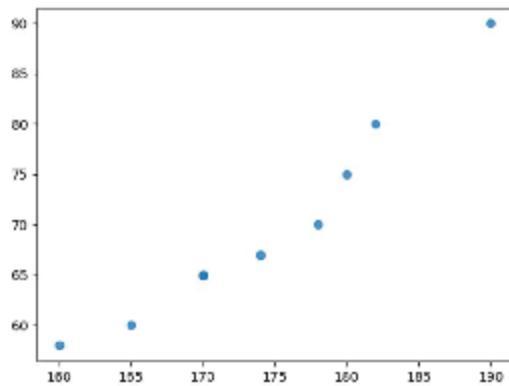


Рис.4.1. Построение графика зависимости двух случайных величин

На первый взгляд зависимость есть, но её оценка вызывает определённые вопросы. Исследователи для оценки взаимосвязи используют коэффициент корреляции Пирсона. Коэффициент может принимать разные значения в диапазоне от -1 до 1. Если рост одной из величин сопровождается ростом другой, то значение ККП принимает положительное значение. Если постоянство одной из величин сопровождается ростом другой, то значение ККП может быть определено как 0. И последний случай, если убывание одной из величин сопровождается ростом другой, то значение ККП принимает отрицательное значение. Как же коррелируют наши данные рост и вес?

---

```
import matplotlib.pyplot as plt
import pandas as pd

data = {'human':['man', 'woman', 'man', 'man', 'woman', 'man', 'woman', 'man',
'man'],
'height':[178, 165, 190, 174, 160, 180, 170, 182, 170],
'weight':[70, 60, 90, 67, 58, 75, 65, 80, 65]}

print(df['height'].corr(df['weight'])) # 0.970201081860922
print(df['weight'].corr(df['height'])) # 0.970201081860922
```

---

Нами получен КПП равный 0.97. Результат говорит о том, что случайные величины действительно очень сильно коррелируют. Можно сделать вывод о том, что если человек имеет высокий рост, то у него, как правило, будет большая масса. Аналогично получается для небольшого роста. Но мы однозначно не можем утверждать, что если мы прибавим в весе, то у нас увеличится рост. При этом нужно учитывать набор тех данных, который у нас имеется перед анализом. Для разных групп данных могут быть получены разные результаты.

В предыдущем примере мы получили численное значение результата корреляции. Если мы хотим продемонстрировать корреляцию на графике, то можно воспользоваться методом `scatter_matrix()`.

```
import matplotlib.pyplot as plt
import pandas as pd

data = {'height':[178, 165, 190, 174, 160, 180, 170, 182, 170],
        'weight':[70, 60, 90, 67, 58, 75, 65, 80, 65],
        'sex':[1, 2, 1, 1, 2, 1, 2, 1, 2]}

df = pd.DataFrame(data)

pd.plotting.scatter_matrix(df, alpha = 0.8)
print(df.corr())
plt.show()
```

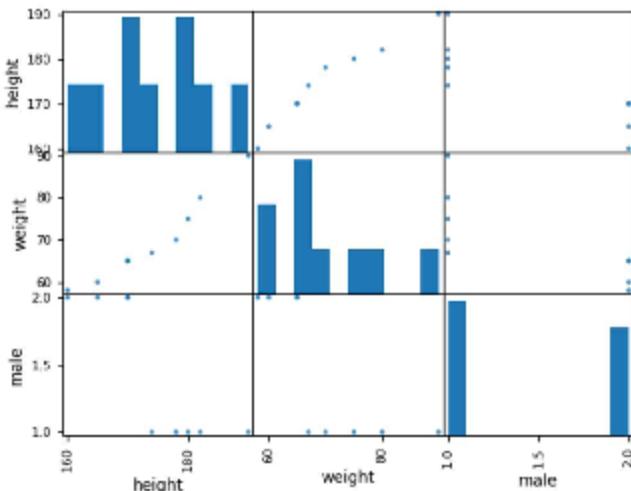


Рис.4.2. Построение матрицы диаграммы рассеяния двух случайных величин

В верхней левой ячейке связь роста с ростом. Аналогично с массой тела. Это не диаграмма рассеяния, а обычная гистограмма распределения по росту и массе тела. В таблицу для анализа дополнительно были добавлена информация о принадлежности к мужчинам – 1 и женщинам – 2. Попробуйте самостоятельно проанализировать остальные графики самостоятельно.

#### 4.5. Проверка гипотезы

Перед тем, как рассмотреть несколько примеров для проверки гипотезы, выделим несколько понятий. Любой исследователь постоянно сталкивается с большим объёмом данных, которые в статистике принято называть **генеральной совокупностью**. Для анализа данных нет необходимости использовать все эти данные. Можно выбрать определённую часть, которая называется **выборка**. По выборке делают вывод о генеральной совокупности. Существуют множество подходов с выделением разных выборок, определения

среднего значения и последующего сравнения этих данных для всех выборок. Среднее значение в этом случае принято называть **выборочным средним**. Таким образом можно определить эффективность выборки случайной величины.

Теперь о том, как формулируется и проверяется гипотеза. **Статистическая гипотеза** – некоторое утверждение о случайной величине. Начинается всё с формулировки нулевой гипотезы ( $H_0$ ). В Python для проверки гипотезы нужно выбрать уровень значимости и выяснить, попадает ли значение в критические интервалы или нет. Предлагаемый метод возвращает **статистику разности** между средним и тем значением, с которым вы его сравниваете и уровень значимости для этой статистики (р-значение). **Р-значение (p-value)** – вероятность получить для данной модели распределения значений случайной величины (среднего арифметического, медианы и др.) близкое к ранее наблюдаемым, при условии, что нулевая гипотеза верна. **Уровень значимости** – это суммарная вероятность того, что измеренное эмпирически значение окажется далеко от среднего. Если расчётный уровень значимости больше критического статистического уровня (может быть 5%, 1%, 0.01%), то отвергать нулевую гипотезу ( $H_0$ ) не рекомендуется.

Если значений в выборке мало, то для оценки используют **t-распределение Стьюдента** (распределение Уильяма Госсета), которое при определённых условиях стремится к нормальному распределению. Для проверки гипотезы о равенстве среднего используется метод `scipy.stats.ttest_1samp(array, popmean)`. Здесь в качестве атрибутов массив `array` (выборка) и `popmean` (среднее значение генеральной совокупности). Рассмотрим пример проверки нулевой гипотезы. Допустим, что у нас есть массив, который содержит значения роста определённой группы людей. Наша нулевая гипотеза заключается в предположении того, что средний рост в этой группе составляет 182 см.

---

```
from scipy import stats as st
import pandas as pd

data = {'height':[178, 165, 190, 174, 160, 180, 170, 182, 170, 176, 154, 167,
187, 192, 160, 156, 154, 188, 178]}

df = pd.DataFrame(data)

inter_value = 182 # предполагаемый средний рост

alpha = .05 # критический уровень статистической значимости 5%

results = st.ttest_1samp(data['height'], inter_value)

print('р-значение: ', results.pvalue)

if (results.pvalue < alpha):
    print('Отвергаем гипотезу H0')
else:
```

```
print('Нельзя отвергнуть гипотезу H0')

mean_value = df.mean(axis = 0)

print('Средний рост: ', mean_value.round(2))
```

---

Уровень значимости p-value составляет  $\sim 0.39\%$ , что меньше заданного критического уровня статистической значимости в 5%. Нулевая гипотеза была отвергнута. Дополнительно мы провели расчёт среднего значения роста методом mean() и получили, что это значение равно 172.68 см.

Необходимо отметить ещё две полезные функции для проверки гипотез: гипотеза о равенстве средних двух генеральных совокупностей st.ttest\_ind(sample\_1, sample\_2) и гипотеза о равенстве средних для зависимых (парных) выборок st.ttest\_rel(before, after). Функции позволяют проверить гипотезу о равенстве средних для двух выборок.



## Задания для самостоятельной работы

### 1. Среднее значение, медиана, мода и диапазон

Для выполнения следующих заданий необходимо скачать датасет, который содержит информацию о погоде в городе Екатеринбург в 2019 году. Подготовленный датасет можно скачать здесь:

---

```
# Исходный датасет был сгенерирован на сайте rp5.ru  
'https://rp5.ru/Архив\_погоды\_в\_Екатеринбурге'
```

```
# Подготовленный датасет для выполнения задания находится здесь  
'https://github.com/mkulab64/data\_science/tree/main/ds\_datasets'
```

---

Используя библиотеку matplotlib.pyplot и pandas, выполните следующие задания.

#### 1.1. Загрузите датасет в программу и выведите на экран первые 10 строк.

---

```
import matplotlib.pyplot as plt  
import pandas as pd  
  
df = pd.read_csv('ekb_weather.csv')
```

---

Проанализируйте полученные данные.

---

```
      date  temp  pressure  humidity  
0  31.12.2019  23:00    -9.2     743.1       76  
1  31.12.2019  20:00    -8.8     744.1       75  
2  31.12.2019  17:00    -7.9     744.5       82  
.  
.  
7  31.12.2019  02:00   -20.9     761.5       79  
8  30.12.2019  23:00   -21.4     764.2       78  
9  30.12.2019  20:00   -20.9     766.8       79
```

---

**1.2.** Методом `info()` определите тип данных в каждом столбце таблицы. Обратите внимание, что столбцы, которые содержат тип данных `object` имеют сложную структуру. Для обработки подобных данных лучше выполнить преобразование к тому типу данных, который можно обрабатывать библиотекой `pandas`.

Примечание: обратите внимание на столбец `date`.

**1.3.** Создайте новый признак `df['month']`, который будет содержать информацию о месяце. Напишите функцию, которая выполняет замену числа от 1 до 12 на строку с названием месяца в году: January, February, March, April, May, June, July, August, September, October, November, December. Полученную таблицу выведите на экран.

**1.4.** Используя логическое индексирование и категориальный признак `df['month']`, выполните следующий анализ:

- 1) определите среднюю температуру за весь приведённый период;
- 2) определите среднее давление за весь приведённый период;
- 3) определите среднюю влажность за весь приведённый период;
- 4) рассчитайте и сравните среднюю температуру в январе и декабре (выведите на экран название месяца, в котором средняя температура будет ниже);
- 5) рассчитайте и сравните среднее давление в апреле и ноябре (выведите на экран название месяца, в котором среднее давление будет выше);
- 6) рассчитайте и сравните среднюю влажность в июне и августе (выведите на экран название месяца, в котором средняя влажность будет выше).

**1.5.** Напишите программу, которая рассчитывает среднее значение температуры в каждом месяце. В программе должен быть создан список, содержащий 12 значений средних температур. Используя список, постройте столбчатую диаграмму распределения средней температуры по месяцам. Получите аналогичную зависимость для медианы и моды.

**1.6.** Напишите программу, которая рассчитывает среднее значение давления в каждом месяце. В программе должен быть создан список, содержащий 12 значений средних значений давления. Используя список, постройте столбчатую диаграмму распределения среднего давления по месяцам. Получите аналогичную зависимость для медианы и моды.

**1.7.** Напишите программу, которая рассчитывает среднее значение влажности в каждом месяце. В программе должен быть создан список, содержащий 12 значений средней влажности. Используя список, постройте столбчатую

диаграмму распределения средней влажности по месяцам. Получите аналогичную зависимость для медианы и моды.

## **2. Элементы теории вероятности: дисперсия и стандартное отклонение**

Для выполнения заданий используйте модифицированный датасет (с новым признаком df['month']) из предыдущего задания (1.1).

**2.1.** Напишите программу, которая рассчитывает дисперсию и стандартное отклонение температуры в каждом месяце. Результаты необходимо хранить в виде двух словарей, где ключ – название месяца, а значение – дисперсия или стандартное отклонение температуры.

**2.2.** Напишите программу, которая рассчитывает дисперсию и стандартное отклонение давления в каждом месяце. Результаты необходимо хранить в виде двух словарей, где ключ – название месяца, а значение – дисперсия или стандартное отклонение давления.

**2.3.** Напишите программу, которая рассчитывает дисперсию и стандартное отклонение влажности в каждом месяце. Результаты необходимо хранить в виде двух словарей, где ключ – название месяца, а значение – дисперсия или стандартное отклонение влажности.

**2.4.** Методом `describe()` найдите для каждого столбца (кроме `date` и `month`): количество элементов в столбце, среднее значение, минимальное значение, максимальное значение, стандартное отклонение и межквартильное расстояние (разность данных между квантилями 25% и 75%).

**2.5.** Для каждого столбца (кроме `date` и `month`) постройте `boxplot`, определите наличие или отсутствие выбросов данных.

Примечание: под выбросами необходимо понимать значения, которые существенно выделяются из общей выборки.

## **3. Взаимосвязь данных**

Для выполнения заданий используйте модифицированный датасет (с новым признаком df['month']) из предыдущего задания (1.1).

**3.1.** Используя признак в таблице данных (`temp`, `pressure`, `humidity`), найдите коэффициенты корреляции по Пирсону следующих пар:

- 1) `df['temp'] / df['pressure']`;
- 2) `df['temp'] / df['humidity']`;
- 3) `df['pressure'] / df['humidity']`;
- 4) корреляцию данных во всей таблице (`df.corr`).

Определите какие пары имеют наибольшую и наименьшую связь.

**3.2.** Используя признак в таблице данных (temp, pressure, humidity), постройте методом `scatter_matrix()` матрицу диаграммы рассеяния для всех величин. Проанализируйте полученные графики и сравните с результатами задания (3.1).

#### 4. Группировка и агрегирование данных

Для выполнения заданий используйте модифицированный датасет (с новым признаком `df['month']`) из предыдущего задания (1.1).

**4.1.** Выполните группировку и агрегацию данных температуры по месяцам. Необходимо рассчитать минимальное, максимальное значение и медиану. Выведите на экран полученную таблицу.

---

```
df.groupby('month')['temp'].agg([min, np.median, max])
```

---

**4.2.** Выполните группировку и агрегацию данных давления по месяцам. Необходимо рассчитать минимальное, максимальное значение и медиану. Выведите на экран полученную таблицу.

**4.3.** Выполните группировку и агрегацию данных влажности по месяцам. Необходимо рассчитать минимальное, максимальное значение и медиану. Выведите на экран полученную таблицу.

#### 5. Проверка гипотезы

Для выполнения заданий используйте датасет из задания 1.1. Используя библиотеку `scipy` и `pandas`, выполните проверку гипотез.

**5.1.** Рассмотрим массив `df['temp']`. Нулевая гипотеза заключается в предположении того, что среднее значение температуры за весь период наблюдения составляет +5 градусов. Необходимо методом `st.ttest_1samp()` проверить нулевую гипотезу, если критический уровень статистической значимости равен 5%.

**5.2.** Рассмотрим массив `df['pressure']`. Нулевая гипотеза заключается в предположении того, что среднее значение давления за весь период наблюдения составляет 761 мм рт.ст.. Необходимо методом `st.ttest_1samp()` проверить нулевую гипотезу, если критический уровень статистической значимости равен 5%.

**5.3.** Рассмотрим массив `df['humidity']`. Нулевая гипотеза заключается в предположении того, что среднее значение влажности за весь период наблюдения составляет 66%. Необходимо методом `st.ttest_1samp()` проверить нулевую гипотезу, если критический уровень статистической значимости равен 5%.

## 5. Обработка числовых и категориальных признаков



Предварительную обработку данных можно назвать одним из важных этапов при создании алгоритмов машинного обучения. На данном этапе выполняется различная математическая обработка числовых данных и преобразование категориальных признаков, с последующей обработкой. Результаты обработки могут оказывать существенное влияние на подгонку алгоритмов. В данном разделе мы подробно рассмотрим различные методы и подходы, позволяющие выполнить подготовку числовых и категориальных признаков для машинных алгоритмов. Вторым важным этапом настройки алгоритмов является оптимизация атрибутов при создании экземпляра класса: оптимизация расчётной сетки, перекрёстная валидация результатов, конвеер и пайплайны. Более подробно поговорим об этом в следующих разделах.

Перед изучением методов обработки давайте познакомимся с возможностью генерации тестовых наборов данных, позволяющих смоделировать набор значений для разных задач. На практике чаще всего вам предстоит создавать машины алгоритмы одной из задач: регрессии, классификации и кластеризации. Первые две задачи относятся к способу обучения с учителем (англ. *supervised learning*). Последнее относится к способу обучения без учителя (англ. *unsupervised learning*). Иногда при создании алгоритма нет возможности использовать реальные данные. В этом случае можно использовать тестовый набор данных, используя специальную библиотеку `sklearn.datasets`: `make_regression`, `make_classification` и `make_blobs`. По ссылке, которая представлена ниже, вы можете ознакомиться с примерами использования данной библиотеки.

<https://scikit-learn.ru/>

В рамках данной главы мы не будем останавливаться на их рассмотрении подробно. Рекомендуется обратиться к литературе на данную тему (описание библиотеки `sklearn`) и самостоятельно их изучить.

## 5.1. Масштабирование признаков

В большинстве случаев числовые данные требуют дополнительной обработки, т.к. использование "сырых данных" не позволит реализовать эффективный алгоритм машинного обучения. Давайте познакомимся с возможностью масштабирования признаков. Масштабирование данных - возможность изменения диапазона значений, в котором находятся данные. В большинстве случаев используется шкала [0,1] или [-1,1]. Наиболее распространённым вариантом масштабирования является min-max масштабирование. Для применения данного масштабирования можно использовать библиотеку sklearn класс MinMaxScaler().

---

```
import numpy as np
from sklearn import preprocessing

# Создать признак
X_data = np.array([[-1.0], [-1.5], [2.0], [3.5], [8.0],
                   [5.0], [-2.0], [2.5], [3.5], [6.0]])

# Создание экземпляра класса MinMaxScaler(), диапазон {0,1}
minmax_scaler = preprocessing.MinMaxScaler(feature_range=(0,1))

# Масштабирование признака
X_data_scaled = minmax_scaler.fit_transform(X_data)

# Данные до и после масштабирования
print('До масштабирования: ', X_data[:5].T)
print('После масштабирования:', X_data_scaled[:5].T)

# Min и Max значения до и после масштабирования
print('min:', X_data.min(), 'max:', X_data.max())
print('min:', X_data_scaled.min(), 'max:', X_data_scaled.max())
```

---

Кроме того, данный класс позволяет реализовать масштабирование для разных диапазонов, что может быть полезно в иных задачах. Для этого существует специальный атрибут feature\_range. Алгоритм масштабирования использует два метода: fit() и transform(). Метод fit() позволяет вычислить минимальное и максимальное значение, а transform() выполнить преобразования данных. Отметим, что можно использовать метод, который объединяет два вышеуказанных метода. Для этого можно воспользоваться методом fit\_transform(). Попробуйте самостоятельно изучить ещё один способ масштабирования MaxAbsScaler().

Существуют способы масштабирования данных на основе статистического анализа, которые учитывают различные выбросы. Для этого можно использовать класс RobustScaler().

---

```

# Масштабирование данных, имеющих сильные выбросы
# Создать признак
X_data = np.array([[-100.0], [5.0], [2.0], [-3.5], [5.0],
                   [8.0], [-20.0], [3.0], [3.5], [60.0]])

# Создание экземпляра класса RobustScaler()
robust_scaler = preprocessing.RobustScaler()

# Масштабирование признака
X_data_robust = robust_scaler.fit_transform(X_data)

# Данные до и после масштабирования
print('До масштабирования: ', X_data[:5].T)
print('После масштабирования: ', X_data_scaled[:5].T)

```

---

При данном масштабировании будет удалена медиана, а преобразование выполнено в соответствии с диапазоном квантилей (по умолчанию используется значение IQR: межквартильный диапазон). IQR - это диапазон между 1-м квартilem (25-й квантиль) и 3-м квартilem (75-й квантиль). После удаления выбросов можно будет выполнить нормализацию и стандартизацию данных.

## 5.2. Стандартизация признаков

Альтернативой масштабирования является использование стандартизации данных. После преобразования данные будут распределены стандартно: нормальное гауссовское распределение, среднее значение равно нулю, стандартное отклонение равно единице. Преобразованный признак представляет собой количество стандартных отклонений, на которое исходное значение отстоит от среднего значения (Z-оценка в статистике).

---

```

import numpy as np
from sklearn import preprocessing

# Создать признак
X_data = np.array([[10.0], [-1.5], [2.0], [3.0], [4.0],
                   [3.0], [-5.0], [2.5], [3.5], [2.0]])

# Создание экземпляра класса StandardScaler()
standard = preprocessing.StandardScaler()

# Стандартизация признака
X_data_standard = standard.fit_transform(X_data)

# Данные до и после стандартизации
print('До стандартизации: ', X_data[:5].T)
print('После стандартизации: ', X_data_scaled[:5].T)

# Данные до и после стандартизации
print('Среднее:', round(X_data_standard.mean()))

```

```
print('Стандартное отклонение:', X_data_standard.std())
```

---

Стандартизацию признаков можно назвать общим требованием для алгоритмов машинного обучения. Применение конкретного метода масштабирования, прежде всего, будет связано с разработкой конкретного типа алгоритма. Если сравнить стандартизацию и масштабирование, то на практике чаще используется первое. Всегда необходим предварительный анализ, который позволит понять целесообразность использования вида нормализации данных. Например, для анализа главных компонент необходимо использовать стандартизацию, а для нейронных сетей лучше использовать min-max масштабирование. По умолчанию рекомендуется использовать стандартизацию.

Необходимо помнить, что на результат стандартизации существенное влияние оказывает наличие выбросов. Перед преобразованием признаком требуется выполнить предобработку. Для борьбы с выбросами можно использовать анализ межквартильного диапазона или специальный класс RobustScaler().

### 5.3. Нормализация данных

Ещё одним важным этапом предварительной обработки является нормализация данных или получение единичной нормы (общая длина равна единице). Для этого можно использовать класс Normalizer(), который позволяет масштабировать значения в отдельных наблюдениях, приводя их к единичной норме. Данный вид масштабирования удобно использовать, например, если необходимо обработать много эквивалентных признаков (классификация текста). С помощью класса Normalizer() можно реализовать несколько вариантов нормы. Чаще всего применяют евклидову норму ( $l_2$ ) и манхэттенскую норму ( $l_1$ ).

```
import numpy as np
from sklearn.preprocessing import Normalizer

# Матрица признаков
X_data = np.array([[0.7, 0.5],
                   [1.5, 2.8],
                   [1.5, 20.2],
                   [2.5, 26.4],
                   [8.2, 6.3]])

# Создание экземпляра класса Normalizer()
# L-квадрат норма (евклидова норма)
#  $L^2 = \sqrt{x_1^2 + x_2^2 + \dots + x_n^2} = 1$ 

normalizer = Normalizer(norm="l2")
X_norm_l2 = normalizer.transform(X_data)
```

```
# Преобразовать матрицу признаков
print(X_norm_12)

# L' норма (манхэттенская норма)
# L' = x1 + x2 ... xn = 1

normalizer = Normalizer(norm="l1")
X_norm_l1 = normalizer.transform(X_data)

# Преобразовать матрицу признаков
print(X_norm_l1)
```

---

Евклидову норму можно рассматривать как кратчайшее расстояние между двумя точками пространства. Манхэттенская норма напоминает расстояние пройденного пути. Обратите внимание, что норма 'l1' масштабирует значения наблюдений так, что в сумме они равны единице.

#### 5.4. Генерация полиномиальных признаков

При нехватке признаков в датасете можно воспользоваться генерацией полиномиальных признаков. В этом случае можно использовать класс `PolynomialFeatures()`. Рассматриваемый класс может быть полезен, если мы точно знаем о связи признака и цели. Возникает возможность кодирования неконстантного эффекта для признака, генерируя формы этого признака высокого порядка (n-ой степени). Может возникнуть ситуация, при которой признак вообще не оказывает влияния на целевую переменную, но сочетание признаков (произведение отдельных признаков) может повлиять.

```
import numpy as np
from sklearn.preprocessing import PolynomialFeatures

# Матрица признаков
X_data = np.array([[0, 1],
                  [2, 3],
                  [4, 5]])

# Создание экземпляра класса PolynomialFeatures
polynom_data = PolynomialFeatures(degree=2, interaction_only=False)

# Создать новую матрицу признаков
polynom_data.fit_transform(X_data)

# out -> [1, x1, x2, x1^2, x1*x2, x2^2]
```

---

При создании экземпляра класса `PolynomialFeatures()` важными атрибутами является значение степени `degree` и ограничение создаваемых признаков только признаками взаимодействия. Последнее определяется атрибутом `interaction_only` и значением `True`.

## 5.5. Преобразование признаков

Для преобразования признаков можно использовать специально предназначенные классы. Многие из этих классов мы разобрали ранее. В этом пункте мы рассмотрим использование универсального подхода, позволяющего выполнить предобработку и преобразование признаков. Рекомендуется перед изучением материала ещё раз подробно разобрать раздел python, в котором рассматривается функциональное программирование, т.к. на этом основан предлагаемый подход. Использование данного подхода позволит, например, создать новый признак на основе существующего, который является логарифмом исходного числа. Для этой цели в библиотеке sklearn существует класс FunctionTransformer(). Рассмотрим пример реализации данного преобразования.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import FunctionTransformer

# Матрица признаков
X_data = np.array([[0, 1, 3],
                  [3, 4, 5],
                  [10, 7, 8],
                  [2, 5, 10]])

# Функция для обработки данных
def transform_func(row):
    return row**2

# Создание экземпляра класса FunctionTransformer()
transform_data = FunctionTransformer(transform_func)

# Создать новую матрицу признаков
transform_data.transform(X_data)
```

Для преобразования признака была создана функция transform\_func(), которая позволяет получить квадрат исходного числа. Далее функция используется как объект. В результате применения метода transform() мы получим новый преобразованный массив, все элементы которого возведены в квадрат. Альтернативной данного подхода можно назвать использование метода apply(). Метод apply() является универсальным решением часто применяемым на практике. Особенность метода заключается в том, что можно выполнить преобразование всего массива или только его части. В случае с датафреймом можно изменять конкретный столбец с данными.

```
# Создание датафрейма и использование метода apply()
df = pd.DataFrame(X_data)
```

```
df_new = df.apply(transform_func)
df_new.rename(columns=lambda x: 'feature_' + str(x), inplace=True)
```

---

В примере была использована ранее созданная функция `transform_func()`. Можно использовать и более сложную функцию. При реализации предлагаемого подхода можно использовать `lambda`-функцию.

## 5.6. Анализ выбросов

При анализе данных важно знать о распределении. В общем случае принимают, что данные распределены нормально. В реальности распределение данных может быть очень сложным. Очень сильное влияние на распределение данных оказывает наличие выбросов. Анализ и обработка признаков при наличии выбросов можно назвать одной из ключевых задач предобработки данных. Далее мы обсудим наиболее распространённые подходы анализа выбросов, которые позволяют привести данные к нормальному распределению. Необходимо отметить, что нормализация данных позволяет подготовить данные для эффективной реализации алгоритмов машинного обучения.

Существует множество способов для анализа данных. Условно их можно разделить на две группы. В первом случае используется анализ без графического представления. Второй подход основан на графическом представлении данных. Для предварительной статистической оценки можно воспользоваться стандартным методом `describe()`.

---

```
# Использование метода describe()
df_new.describe()
```

---

Метод `describe()` позволяет рассчитать значение межквартильного размаха. Межквартильный размах (IQR) - это разница между первым (25%) и третьим квартилями (75%) набора данных. Предполагается, что 50% данных находится именно в этом диапазоне. Для фильтрации данных выполняется срез согласно нижней и верхней границы выбросов.

---

```
# Вывод 25-го и 75-го перцентиля данных
df_new.describe()['feature_0'][['25%', '75%']]
```

---

Очень часто на практике визуализируют распределение данных, используя график `boxplot` и функции плотности вероятности. На рис.5.1 представлен пример нормально распределённых данных с указанием диапазона сигм (стандартное отклонение).

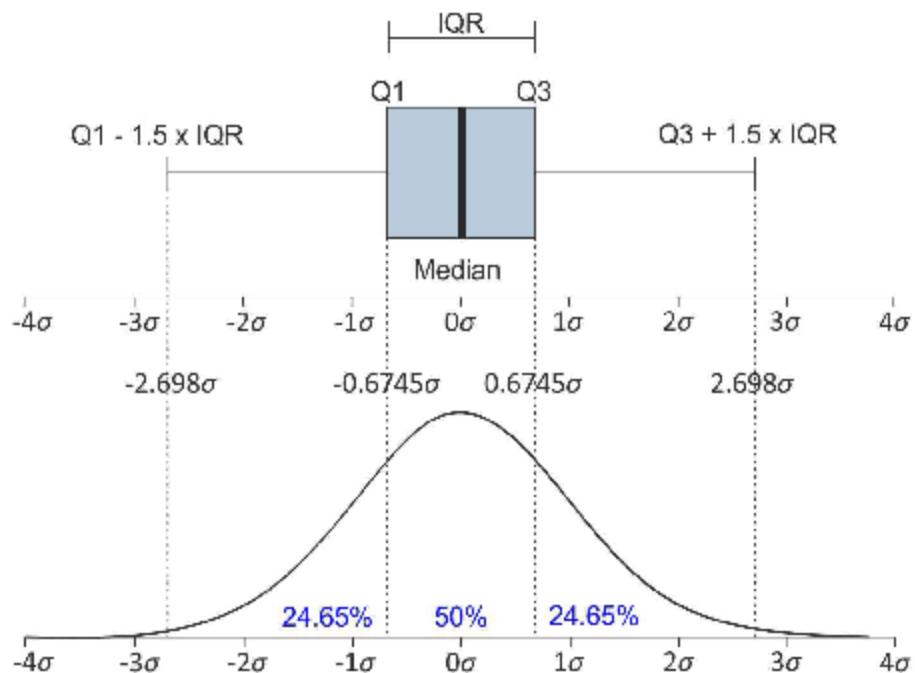


Рис.5.1. График boxplot и функции плотности вероятности нормально распределённых данных

Рассмотрим подробнее расчёт межквартильного размаха и границ выброса на следующем примере. Определение границ выбросов выполняется в диапазоне  $[q1 - (iqr*1.5), q3 + (iqr*1.5)]$ , где  $q1$  и  $q3$  - значения первого и третьего квартиля. Всё, что выходит за границы диапазона можно считать выбросами. Выбросы, которые находятся вне диапазона можно удалить, но, как правило, их заменяют другими значениями. Например, можно выполнить замену средним значением по всему столбцу.

---

```

q1 = df_new.feature_0.quantile(0.25) # 25-й перцентиль
q3 = df_new.feature_0.quantile(0.75) # 75-й перцентиль
iqr = q3 - q1                      # межквартильный размах
lower_bound = q1 - (iqr*1.5)         # нижняя граница выбросов
upper_bound = q3 + (iqr*1.5)         # верхняя граница выбросов

print('25-й перцентиль: {},'.format(q1),
      '75-й перцентиль: {},'.format(q3),
      "IQR: {}, ".format(iqr),
      "Границы выбросов: [{lb}, {ub}].".format(lb=lower_bound,
      ub=upper_bound))
  
```

---

Рассмотрим пример обработки выбросов, используя функцию `del_outliers()` и среднее значение. Можно использовать и другие величины для замены: мода, медиана или нулевое значение.

---

```

mean_value = df_new['feature_0'].mean()

def del_outliers(row):
  
```

```
if row > upper_bound or row < lower_bound:  
    return mean_value  
else:  
    return row  
  
df_new['feature_3'] = df_new['feature_0'].apply(del_outliers)
```

---

Для графического анализа используем boxplot. На графике представлено распределение признака feature до (feature\_0) и после (feature\_3) замены выбросов на среднее.

---

```
import seaborn as sns  
from scipy import stats  
  
norm_rv = stats.norm(loc=30, scale=5)  
samples = np.trunc(norm_rv.rvs(365))  
  
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(15, 4))  
fig.suptitle('Распределение признака feature_0')  
histplot = sns.histplot(x=samples, ax=axes[0])  
boxplot = sns.boxplot(x=samples, ax=axes[1])  
  
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(15, 4))  
fig.suptitle('Распределение признака feature_3')  
histplot = sns.histplot(x=samples, ax=axes[0])  
boxplot = sns.boxplot(x=samples, ax=axes[1])
```

---

Существует множество других способов нормализации данных при наличии выбросов. Среди наиболее распространённых можно отметить создание признака на основе логического условия и создание признака на основе логарифмирования.

---

```
# Создание признака на основе булева условия  
df_new['feature_4'] = np.where(df_new['feature_1'] < 20, 0, 1)  
  
# Логарифмирование признака  
df_new['feature_5'] = [np.log(x) for x in df_new['feature_2']]
```

---

Попробуйте самостоятельно проанализировать полученные результаты преобразования признаков, используя графическое представление данных. Отметим, что любой процесс стандартизации данных при наличии большого количества выбросов является бесполезным, т.к. среднее значение и дисперсия будут отражать реальное распределение данных.

## 5.7. Дискретизация признаков

Далее мы рассмотрим процесс дискретизации данных. Дискретизация данных будет полезна, если изначально понятно, что числовой признак может

быть представлен в качестве категориального признака. Например, имеем данные о возрасте некоторой группы людей. Возрастной диапазон находится в пределах от 17 до 85 лет. Мы можем группировать данные согласно категориям. Подгруппа от 17 до 35 лет можно отнести к молодому возрасту, от 36 до 65 лет к среднему возрасту, старше 66 лет к пожилому возрасту. В итоге, получим деление на 3 большие группы. Для бинаризации данных можно использовать класс Binarizer().

---

```
import numpy as np
from sklearn.preprocessing import Binarizer

# Создать признак
X_data = np.array([[1], [10], [100], [1000]])

# Создание экземпляра класса Binarizer()
binarizer = Binarizer(50) # порог бинаризации = 50

# Бинаризация признака
X_binar = binarizer.fit_transform(X_data)
print(X_binar.T)
```

---

Для дискретизации данных для трёх и более корзин можно использовать метод digitize() библиотеки numpy.

---

```
# Разбиение на диапазоны
X_binar_digit = np.digitize(X_data, bins=[1, 10, 100, 1000])
print(X_binar_digit.T)
```

---

Предварительный анализ данных всегда позволяет определить необходимость дискретизации данных на нужное количество классов, исходя из условий задачи.

## 5.8. Работа с пропущенными данными

В зарубежной литературе по анализу данных на языке Python очень часто можно встретить термин импутация (imputation) данных. Под импутацией в предобработке данных понимают процесс замены пропущенных или некорректных значений. Вариантов решения данной проблемы существует достаточно много. Наиболее часто используют замену данных на основе статистической оценки данных или использовании машинных алгоритмов. Первое основано на расчёте статистической величины, например, среднего значения и замены этим значением пропусков. Второе основано на возможности предсказания неизвестной величины по существующим данным. Примером является алгоритм k ближайших соседей (KNN). Рассмотрим

некоторые способы заполнения пропущенных значений на основе тестового датафрейма.

---

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Создание списка с пропущенными данными
numbers = [[1, 8, np.nan],
            [2, np.nan, 16],
            [np.nan, 10, 17],
            [7, np.nan, 18],
            [5, 12, np.nan]]

header = ['feature_1', 'feature_2', 'feature_3']

# Создание датафрейма
df = pd.DataFrame(data = numbers, columns = header)

# Подсчёт пропущенных значений в датафрейме
df.isnull().sum()
```

Для подсчёта количества пропущенных значений использовать метод `isnull()` (или `isna()`) и метод `sum()`. Если применить методы к всему датафрейму, то можно узнать кол-во пропусков по каждому признаку. Анализ пропусков можно проводить с помощью графического представления данных. Одним из вариантов графического представления является построение тепловой карты `heatmap` из библиотеки `seaborn`.

---

```
# Визуализация пропусков
plt.figure(figsize=(6,4))
sns.heatmap(df.isna().transpose(),
            cmap="YlGnBu",
            cbar_kws={'label': 'Пропущенные данные'})
plt.show()
```

После предварительного анализа пропусков необходимо выполнить их замену. Можно воспользоваться методами для расчёта статистических показателей: метод `mean()`, `median()` и `mode()`. Для замены пропусков использовать метод `fillna()`.

---

```
# Замена пропусков

feat_1_mean = df['feature_1'].mean()                                # расчёт среднего
feat_2_median = df['feature_2'].median()                             # расчёт медианы
feat_3_mode = df['feature_3'].mode()                                 # расчёт моды
```

```
df['feature_1'].fillna(feat_1_mean, inplace=True)      # замена пропусков на
 среднее
df['feature_2'].fillna(feat_2_median, inplace=True)    # замена пропусков на
 медиану
df['feature_3'].fillna(feat_3_mode[0], inplace=True)   # замена пропусков на
 моду
```

---

Для замены пропусков можно использовать специальную библиотеку, например, `sklearn.impute` и класс `SimpleImputer()`. При создании экземпляра класса обратите внимание на атрибут `strategy`, который позволяет определить данные для замены (по умолчанию среднее значение).

---

```
# Импутация признаков
from sklearn.impute import SimpleImputer

X_data = np.array(numbers)

# Создание экземпляра класса SimpleImputer()
mean_imputer = SimpleImputer(missing_values=np.nan, strategy='most_frequent')

# Импутация значений
mean_imputed = mean_imputer.fit_transform(X_data)

print("Импутируемое значение:", mean_imputed[0,2])
```

---

По точности предсказанного значения алгоритм KNN лучше, чем замена средним значением. Проблема заключается в том, что на больших данных использование алгоритма ближайших соседей представляется проблематичным, из-за ошибки нахождения истинного. Замена средним, в случае масштабирования на миллионы данных, в итоге, будет предпочтительнее.

## 5.9. Кодирование номинальных категориальных признаков

Кодирование номинальных признаков можно назвать одним из важных этапов при предобработке данных. Существуют ситуации, когда числовых признаков в наборе данных практически нет. Использование подобных данных при создании и обучении алгоритмов машинного обучения представляется невозможным или трудно реализуемым. Преобразование признаков, в этом случае, позволяет решить эту задачу. Теперь рассмотрим более подробные примеры. Для начала рассмотрим так называемую однократную бинаризацию признаков. В зарубежной литературе очень часто встречается определение кодирование с одним активным состоянием (*one-hot encoding*). Терминология была позаимствована из цифровой схемотехники, где информация может быть закодирована с использованием только "1" или "0".

На основе этого принципа были созданы специальные библиотеки, например, `sklearn.preprocessing`.

Далее мы подробно рассмотрим примеры использования класса `LabelBinarizer()` и `MultiLabelBinarizer()`. Первый класс используется в том случае, если строка признака содержит только одно значение (список с единственным значением). При создании экземпляра класса и использовании метода `fit_transform()` будет создан новый признак. Только новая структура будет содержать исключительно массивы с числами. Если значение присутствовало в исходном признаке, то оно будет закодировано единицей, все остальные значения будут представлены нулями. Аналогично для всех строк признака. Рассмотрим пример использования.

---

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelBinarizer, MultiLabelBinarizer

# Создание признака
X_data = np.array([['pressure'],
                   ['gas'],
                   ['pressure'],
                   ['temp'],
                   ['pressure']])

# Создание экземпляра класса LabelBinarizer()
one_hot = LabelBinarizer()

# Кодирование признака
one_hot.fit_transform(X_data)
```

---

Второй класс используется в том случае, если строка признака содержит несколько значений (список с несколькими значениями). Если значения присутствовали в исходном признаке, то они будут закодированы единицами, все остальные значения будут представлены нулями. Аналогично для всех строк признака. Далее пример использования.

---

```
# Создание мультипризнака
X_data_mult = np.array([['pressure', 'humidity'],
                        ['gas', 'pressure', 'temp'],
                        ['pressure', 'humidity'],
                        ['temp', 'pressure'],
                        ['pressure', 'humidity']])

# Создание экземпляра класса MultiLabelBinarizer()
one_hot = MultiLabelBinarizer()

# Кодирование признака
one_hot.fit_transform(X_data_mult)
```

---

При использовании библиотеки pandas тоже можно реализовать однократную бинаризацию признаков one-hot encoding (ОНЕ). На рис.5.2 представлена схема однократного бинарного кодирования с использованием данного метода.

<b>id</b>	<b>feature</b>	<b>id</b>	<b>feature_pressure</b>	<b>feature_gas</b>	<b>feature_temp</b>
1	pressure	1	1	0	1
2	gas	2	0	1	0
3	pressure	3	1	0	0
4	temp	4	0	0	1
5	pressure	5	1	0	0

Рис.5.2. Схема бинарного кодирования с использованием one-hot encoding

В pandas можно использовать специальный метод `get_dummies()` для реализации ОНЕ. В данном случае мы получаем результат, который можно было получить при использовании `LabelBinarizer()`, а вот реализовать `MultiLabelBinarizer()` указанным методом нельзя.

---

```
# Кодирование признака и создание датафрейма
pd.get_dummies(X_data[:,0])
```

---

Далее будут рассмотрены примеры совместного использования библиотек `sklearn` и `pandas`. Кодирование признаков можно выполнить на основе следующих классов `sklearn.preprocessing`: `OrdinalEncoder()`, `LabelEncoder()`, `OneHotEncoder()`. Эти классы имеют особенности их применения. Если требуется выполнить преобразование признака, в котором количество различных меток более 10-15, то лучше использовать ое- или ле-кодирование. В этом случае каждая метка будет закодирована числом. Преобразование может быть выполнено для исходного признака (его пересохранение), новых признаков, при этом, создано не будет. Иная ситуация получается при оне-кодировании. Кол-во признаков будет определяться наличием уникальных меток признака. Представьте, если количество уникальных меток 100, 1000 или 1 млн. Понятно, что использование оне в данной ситуации является нецелесообразным, т.к. потребуются большие вычислительные ресурсы компьютера.

---

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OrdinalEncoder, LabelEncoder, OneHotEncoder
```

```

# Создание словаря
data = {'temp': ['high', 'low', 'high', 'low'],
         'size': ['small', 'big', 'big', 'medium'],
         'pressure': ['isobar','isochor','isother','isochor']}

# Создание датафрейма
df = pd.DataFrame(data = data)

# Создание экземпляра класса OrdinalEncoder()
ord_enc = OrdinalEncoder()

# Кодирование признака
df['pressure_ordinal'] = ord_enc.fit_transform(df[['pressure']])

# Создание экземпляра класса LabelEncoder()
lab_enc = LabelEncoder()

# Кодирование признака
df['pressure_label'] = lab_enc.fit_transform(df['pressure'])

# Создание экземпляра класса OneHotEncoder()
ohe_enc = OneHotEncoder()
# Кодирование признака
ohe_res = ohe_enc.fit_transform(df[['pressure']])

ohe_df = pd.DataFrame(ohe_res.toarray(), columns=list(*ohe_enc.categories_))
df = pd.concat([df, ohe_df], axis=1)

# ОНЕ с использованием библиотеки pandas
pd.get_dummies(df, columns=['size','temp'], prefix=['size','temp'])

# Проверка типа данных
type(df['pressure_ordinal'].iloc[2])
type(df['pressure_label'].iloc[2])

```

---

Отметим, что oe- или le-кодирование позволяют получить очень близкие результаты. Принципиальное отличие заключается в том, что OrdinalEncoder() предназначен для двумерных данных (2D) с формой (n\_samples, n\_features), а LabelEncoder предназначен для одномерных данных (1D) с формой (n\_samples,). Обратите внимание, что отличается тип преобразованных данных. В случае oe тип данных float, при le тип данных int. Если говорить про применимость рассмотренных методов, то их чаще всего используют для обработки порядковых категориальных признаков.

Альтернативой библиотеки sklearn.preprocessing является библиотека category\_encoders. В ней можно аналогичным образом выполнить ОНЕ обработку признаков.

---

```

import category_encoders as ce

# Создание экземпляра класса OrdinalEncoder()

```

```

ord_enc = ce.OrdinalEncoder()
data_ord = ord_enc.fit_transform(df['size'])
df = pd.concat([df, data_ord], axis=1)

# Создание экземпляра класса OneHotEncoder()
ohe_enc = ce.OneHotEncoder(cols=['pressure'])
data_ohe = ohe_enc.fit_transform(df['pressure'])
df = pd.concat([df, data_ohe], axis=1)

# Создание экземпляра класса BinaryEncoder()
bin_enc = ce.BinaryEncoder(cols=['pressure'])
data_bin = bin_enc.fit_transform(data['pressure'])
data = pd.concat([data, data_bin], axis=1)

```

Подведём некоторый итог. Кодирование номинальных категориальных признаков позволяет преобразовать строковые данные в числовые значения. Если количество признаков не превышает 10-15 значений, то можно смело использовать охе-кодирование. Если признаков больше, то можно использовать ое- или ле-кодирование. Второе можно назвать частным случаем, т.к. кодирование меток в случае номинальных категориальных признаков может, в итоге, дать отрицательный результат при обучении модели. При кодировании большого количества признаков настоятельно рекомендуется использовать бинарное кодирование. На рис.5.3 представлена схема бинарного кодирования.

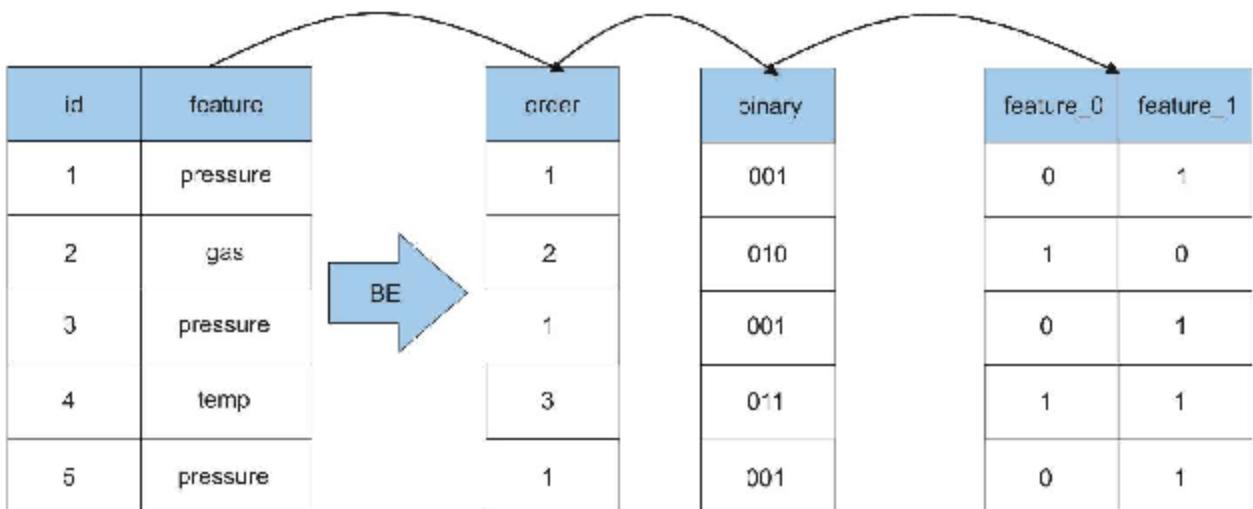


Рис.5.3. Схема бинарного кодирования с использованием BinaryEncoder()

Бинарное кодирование основано на представлении данных в двоичном формате. Схема метода следующая. Сначала выполняется преобразование категориального признака в число, что-то наподобие кодирования меток. На следующем шаге полученные числовые значения будут преобразованы в двоичный код. Количество разрядов будет определять число новых столбцов. Данный метод целесообразно использовать в том случае, если вы имеете дело с большим количеством признаков в столбце (более 10-15).

Далее будут рассмотрены способы кодировки порядковых категориальных признаков.

## 5.10. Кодирование порядковых категориальных признаков

Ранее мы уже рассмотрели использование методы кодирования признаков с использование OrdinalEncoder() и LabelEncoder(). Было отмечено, что использование этих методов при обработке номинальных категориальных признаков, т.к. может негативно влиять на результаты предсказания модели. Вес меток будет неправильно интерпретирован алгоритмом. В случае с порядковыми признаками ситуация иная. Эти методы можно и нужно использовать. Подробнее с методами можно ознакомиться в предыдущем разделе. В данном разделе мы обсудим дополнительные способы преобразования порядковых признаков.

Для преобразования порядковых категориальных признаков не всегда удобно использовать специально написанную функцию. Более простым подходом является использование специальных методов. Например, на практике очень часто реализуют замену признаков на основе словарей и методов replace() и map(). Рассмотрим примеры использования данных методов.

---

```
# Замена порядковых категориальных признаков методом replace()

import pandas as pd

# Создание словаря
data = {'temp': ['high', 'low', 'high', 'middle'],
        'size': ['small', 'big', 'big', 'medium'],
        'pressure': ['isobar','isochor','isother','isochor']}

# Создание датафрейма
df = pd.DataFrame(data = data)

scale_rep = {'small': 1, 'medium': 2, 'big': 3}
df['size'] = df['size'].replace(scale_rep)

# Замена признаков с использованием вложенных словарей
scale_rdic = {'temp': {'low': 1, 'middle': 2, 'high': 3}}
df = df.replace(scale_rdic)

# Замена порядковых категориальных признаков методом map()
scale_map = {'isochor': 1, 'isother': 2, 'isobar': 3}
df['pressure'] = df['pressure'].map(scale_map)
```

---

Необходимо всегда представлять возможность реализации данного подхода. Если, например, в столбце указано 5-10 категорий, то создание словаря не будет вызывать трудностей. В случае большего количества

категорий желательно использовать иные подходы, например, уже хорошо нам известный кодировщик LabelEncoder.

## 5.11. Кодирование словарей признаков

Ранее было показано, что для преобразования категориальных признаков были использованы словари. Словарь является уникальной структурой, которая позволяет реализовать более сложную структуру с использованием списка или кортежа. В частности, далее мы рассмотрим пример создания массива данных на основе списка и вложенных словарей. Обратите внимание, что каждый словарь содержит пару ключ-значение, где в качестве ключа используется признак, а в качестве значения числовые данные. На выходе программы получается массив числовых данных.

---

```
# Создание массива категориальных признаков на основе словарей

from sklearn.feature_extraction import DictVectorizer

# Создание словаря с данными
data_dict = [{"high": 2, "low": 5},
              {"high": 4, "low": 3},
              {"high": 3, "middle": 2},
              {"high": 2, "middle": 4}]

# Создание экземпляра класса DictVectorizer()
dictvec = DictVectorizer(sparse=False)

# Кодировка признаков
features = dictvec.fit_transform(data_dict)

# Матрица признаков и названия признаков
feature_names = dictvec.get_feature_names()

# если sparse=True, то features.toarray()
features, feature_names
```

---

При создании массива очень важным параметром DictVectorizer() является атрибут sparse. Если атрибут принимает значение False, то будет сгенерирована плотная матрица, значение True позволит сгенерировать разреженную матрицу, в которой будут сохранены только те данные, где значения отличны от нуля. Подобный подход позволит, в итоге, минимизировать потребности в оперативной памяти, что особенно важно, например, при обработке естественного языка (текстовая информация).

В качестве ещё одного важного этапа преобразования категориальных признаков можно отметить этап обработки пропущенных данных (импутация данных). Для этого можно воспользоваться методами, которые были

использованы при обработке числовых данных. Правда, перед этим потребуется предварительное преобразование признаков.

## 5.12. Категориальные переменные

Существуют ситуации, когда данные необходимо представить в виде ограниченного набора категорий. Процесс категоризации предназначен для преобразования порядковых данных в категории. Категориальные переменные - это тип данных pandas (dtype: category), соответствующий категориальным переменным в статистике. Переменная может принимать фиксированное число возможных значений (уровней). Примерами категорий может быть возраст, пол, социальный класс, группа крови и многое другое. Рассмотрим пример создания категорий на основе числовых данных. В качестве тестовых значений будем использовать последовательность случайных целых чисел.

---

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Создание датафрейма на основе генерации случайных чисел
df = pd.DataFrame({'value': np.random.randint(0, 100, 1000)})

# Создание меток категорий
labels = ['{} - {}'.format(i, i + 9) for i in range(0, 100, 10)]
# Создание признака категорий
df['value_group'] = pd.cut(df.value, range(0, 105, 10), right=False,
                           labels=labels)

sns.countplot(x='value_group', data=df)
plt.show()
```

---

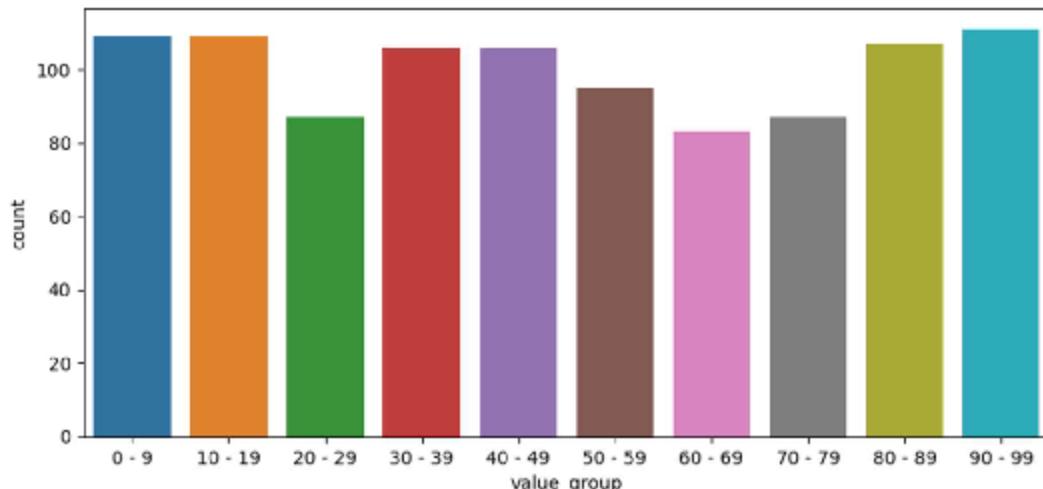


Рис.5.4. Распределение данных после категоризации

С помощью функции range() были созданы различные диапазоны (bins), в которых распределены исходные числовые данные. Можно было указать диапазоны в явном виде, например, используя список [0, 10, 20, 30 … 70, 80, 90, 105]. На рис.5.4 представлено распределение данных после категоризации. Для генерации чисел используется метод randint(). По этой причине каждое новое построение распределения может немного отличаться.

## 6. Исследовательский анализ данных и машинное обучение



Данный раздел будет посвящен рассмотрению различных алгоритмов исследовательского анализа, основанных на методах машинного обучения. **Машинное обучение** является частью науки о данных и представляет собой научное исследование вычислительных алгоритмов и статистических моделей для решения различных задач с помощью определённых шаблонов. Машинное обучение можно представить набором инструментов для построения моделей на основе данных. Специалисты по обработке данных исследуют данные, выбирают и строят модели (машины), настраивают параметры таким образом, чтобы модель соответствовала наблюдениям (обучению), а затем используют модель для прогнозирования и понимания особенностей новых неизвестных данных. Можно выделить четыре этапа машинного обучения: загрузка данных (*import*), создание модели (*instantiate*), обучение (*fit*), предсказание (*predict*). Важным моментом использования модели является её проверка. Плохо обученная или неправильно обученная модель при предсказании может давать ошибочные результаты.

В машинном обучении все алгоритмы можно разделить на две большие группы: контролируемое и неконтролируемое обучении. **Контролируемое обучение** (обучение с учителем, *supervised learning*) подразумевает, что входные данные обладают различными векторами признаков и метками (связанные метки). Связанные метки определяет учитель и по своей сути они являются правильным ответом на поставленный вопрос. Алгоритм обучения сравнивает векторы и метки для поиска структуры взаимосвязи между ними. Примером контролируемого обучения является предсказание погоды по набору входных данных: влажность, давление и температура. **Неконтролируемое обучение** (обучение без учителя, *unsupervised learning*) предполагает, что обучение происходит с использованием наборов данных без определенной структуры и меток. В этом случае модель сама выполняет классификацию данных. Примером неконтролируемого обучения является анализ покупательской активности людей в магазине.

Для реализации алгоритмов машинного обучения на языке Python можно воспользоваться библиотекой sklearn или scipy.

## 6.1. Проверка мультиколлинеарности и отбор признаков

Теперь рассмотрим ещё один важный этап в исследовании данных: анализа мультиколлинеарности и проведение статистических тестов. Основной принцип, которого необходимо придерживаться, говорит о том, что признаки не должны иметь высокую степень связи между собой. Степень корреляции между признаками можно оценить, например, используя критерий корреляции Пирсона или Спирмена. Если между признаками имеется сильная степень связи ( $+/->0.7$ ) или очень сильная связь ( $+/->0.9$ ), один из этих признаков нужно удалить. В противном случае это может оказаться отрицательное влияние на качество модели. Главное помнить, что корреляция необязательно должна быть причинно-следственной связь.

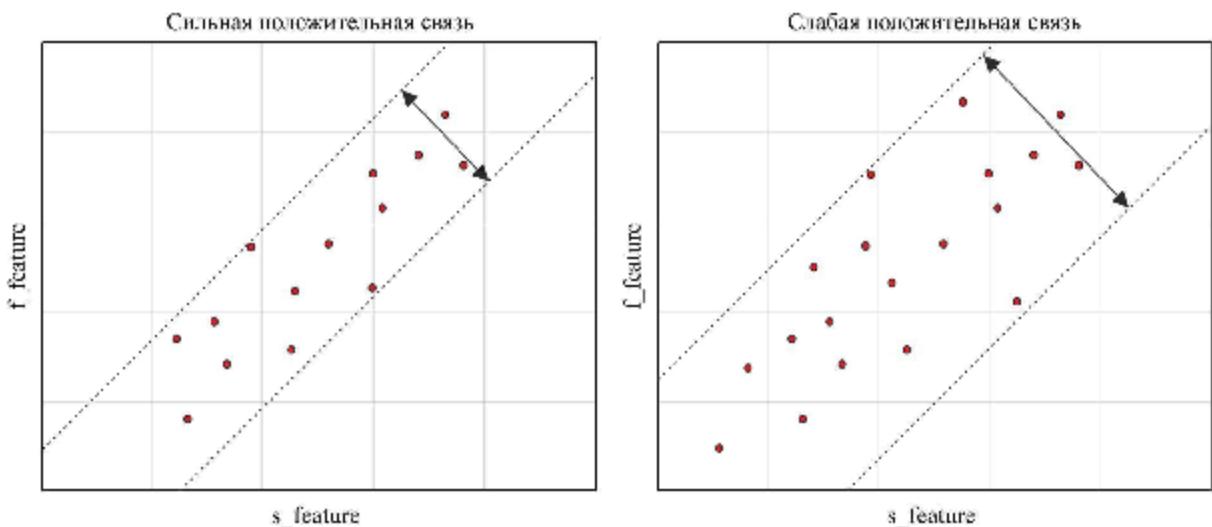


Рис.6.1. Демонстрация положительной связи между признаками

Существует множество способов анализа степени связи между признаками. Для качественной оценки можно воспользоваться графической интерпретацией (в случае пары признаков). На рис.6.1 представлена положительная связь двух признаков. Обратите внимание, что при сильной связи точки на графике имеют меньший разброс. Можно определить и отрицательную корреляцию. На рис.6.2 представлена сильная отрицательная связь двух признаков. Дополнительно приведена зависимость в случае отсутствия связи. Для построения зависимости признаков можно использовать, например, scatter plot. Существуют и иные способы графической визуализации, которые используются для этих целей. Ранее мы уже рассматривали один из них. Более подробное представление данных можно выполнить с использованием scatter\_matrix (библиотека matplotlib) и pairplot (библиотека seaborn).

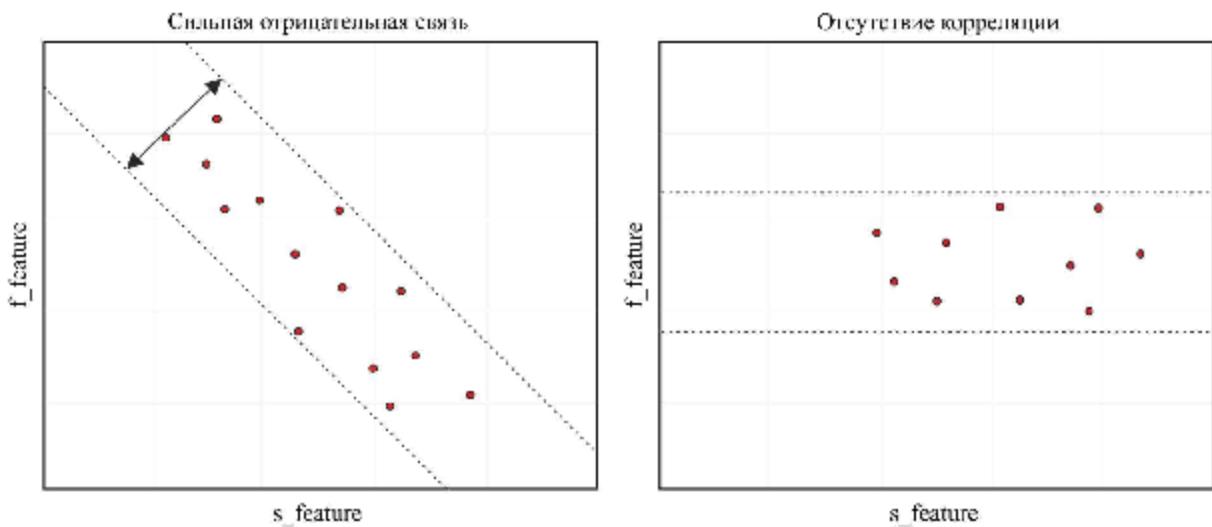


Рис.6.2. Демонстрация отрицательной связи и отсутствие связи между признаками

Рассмотрим пример корреляции признаков на датасете, в котором содержится информация о размерах о цветках ириса.

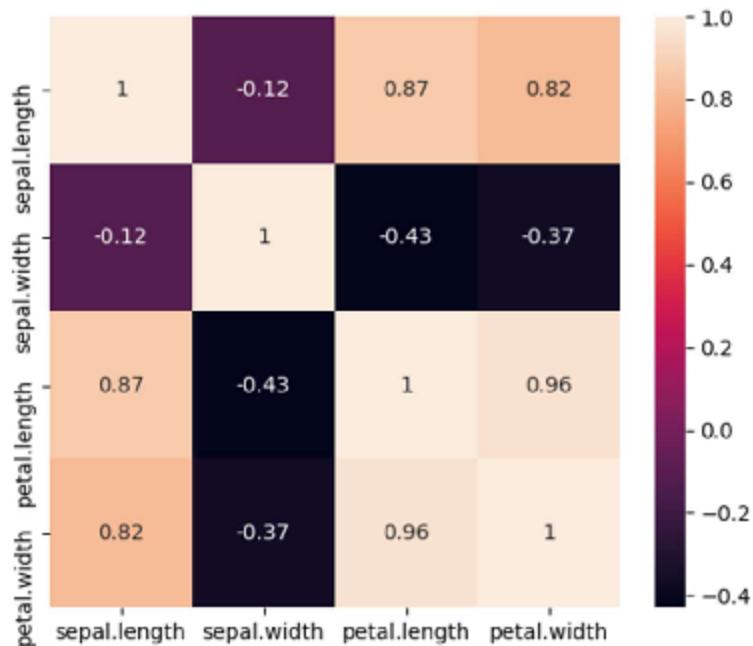
---

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

iris = pd.read_csv('iris.csv')

plt.rcParams['figure.figsize'] = (10,10)
sns.heatmap(iris.drop(['species'], axis=1).corr(), annot=True)
plt.show()
```

---



На рис.6.3 представлены коэффициенты корреляции на тепловой карте heatmap(). На рис.6.4 представлена зависимость всех данных с использованием pairplot. Обратите внимание, что по диагонали приведены гистограммы распределения данных. В этом случае можно предварительно сделать вывод о том, распределены ли данные нормально или нет.

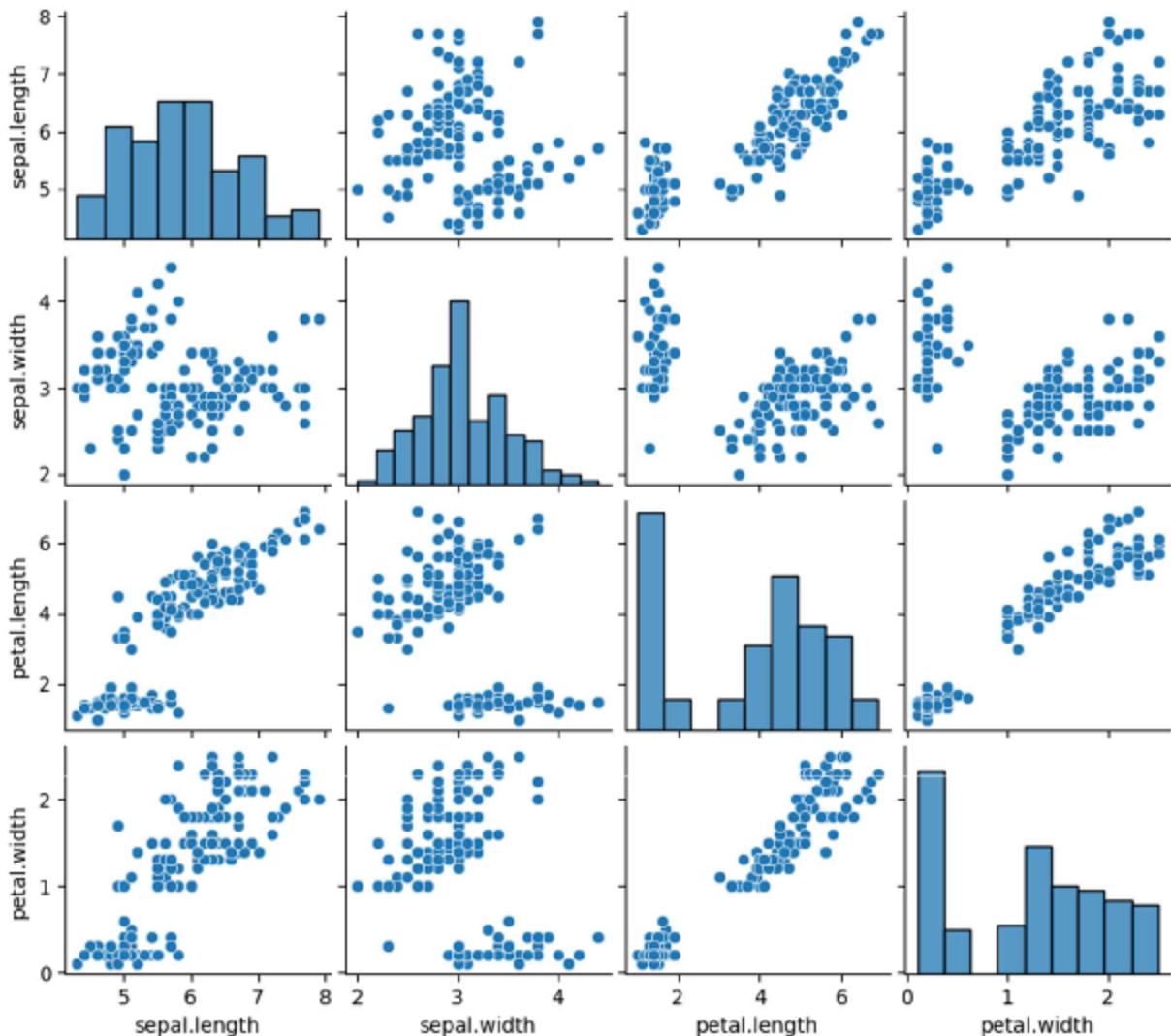


Рис.6.4. Коэффициенты корреляции на тепловой карте heatmap()

---

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

iris = pd.read_csv('iris.csv')

sns.pairplot(iris)
plt.show()
```

---

Второй важный момент, на который необходимо обратить внимание, связан с влиянием признаков на целевую переменную. Основная задача

оставить в наборе только те признаки, которые будут влиять на целевую переменную. Для этого можно воспользоваться оценкой хи-квадрат (в случае классификации) и дисперсионным анализом (второе название ANOVA).

---

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from itertools import compress
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.model_selection import train_test_split

iris = pd.read_csv('iris.csv')

X = iris.drop(['species'], axis=1)
y = iris['species']

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,
random_state = 42, test_size = 0.33)
skb = SelectKBest(f_classif, k=2).fit(X_train, y_train)
mask = skb.get_support()
name_col = X_train.columns.values
result = list(compress(name_col, mask))
score_skb = skb.scores_

plt.bar(result, score_skb)
plt.show()
```

---

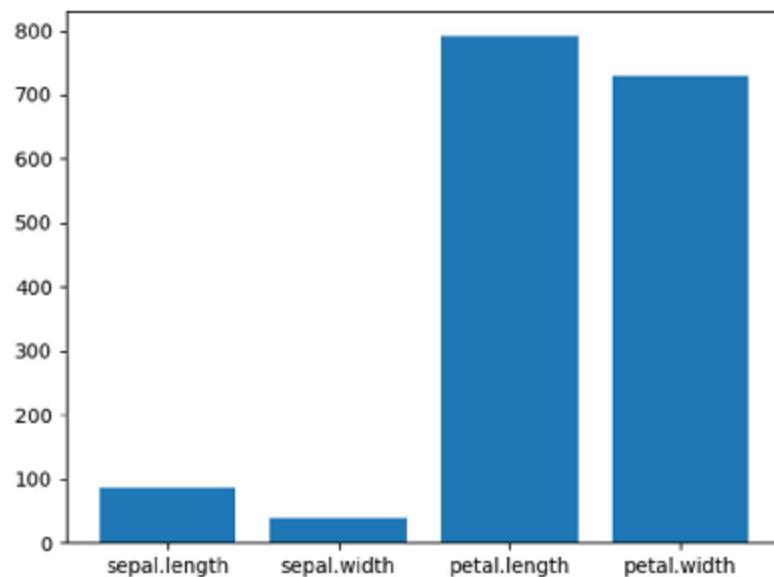


Рис.6.5. Оценка важности признаков с помощью `f_classif`

На рис.6.5 представлен результат анализа важности признаков с помощью `f_classif`. Результат анализа свидетельствует о важности каждого. Более важные признаки могут быть использованы при построении алгоритма машинного обучения. Отбор признаков, в итоге, позволит повысить эффективность полученной модели.

## 6.2. Регрессионный анализ

Изучение машинных алгоритмов контролируемого обучения мы начнём с регрессионного анализа. Частным случаем регрессионного анализа является линейная регрессия. Линейная регрессия (англ. Linear regression) направлена на поиск неизвестных параметров линейной функции, которая может быть описана математически как функция  $y = kx + b$ . В нашем случае  $x$  и  $y$  набор выходных и выходных данных,  $k$  определяет наклон прямой,  $b$  – точку пересечения графика с осью ординат  $(0, b)$ . Линейная регрессия сводится к поиску именно неизвестных параметров  $k$  и  $b$  по известному набору данных. Условием эффективности поиска параметров является сведение ошибки подбора к минимуму.

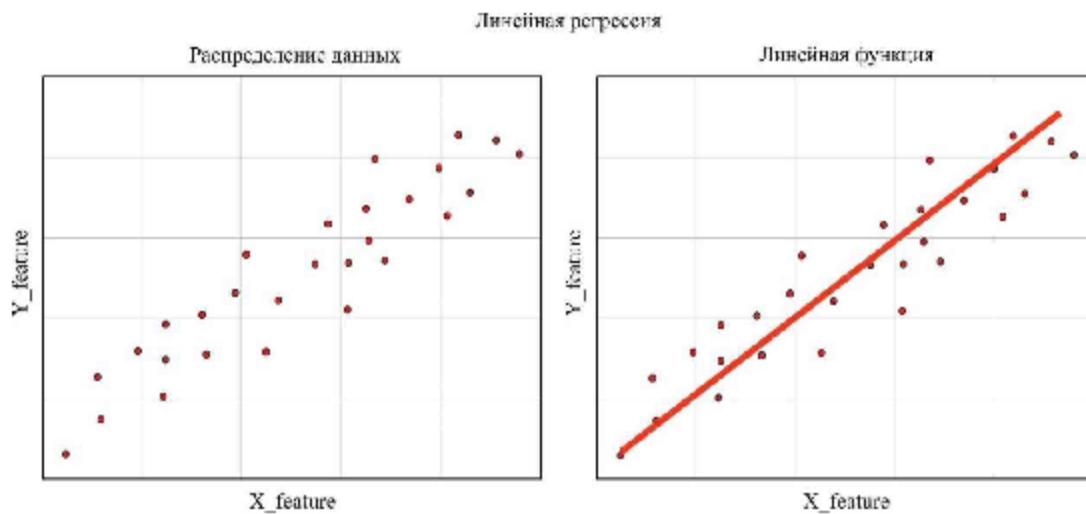


Рис.6.3. Пример реализации линейной регрессии

На этом месте возникает закономерный вопрос. Каким образом происходит подбор неизвестных параметров? Эффективность подбора функции зависит от разности (расстояния) между теоретической точкой и реальным значением.

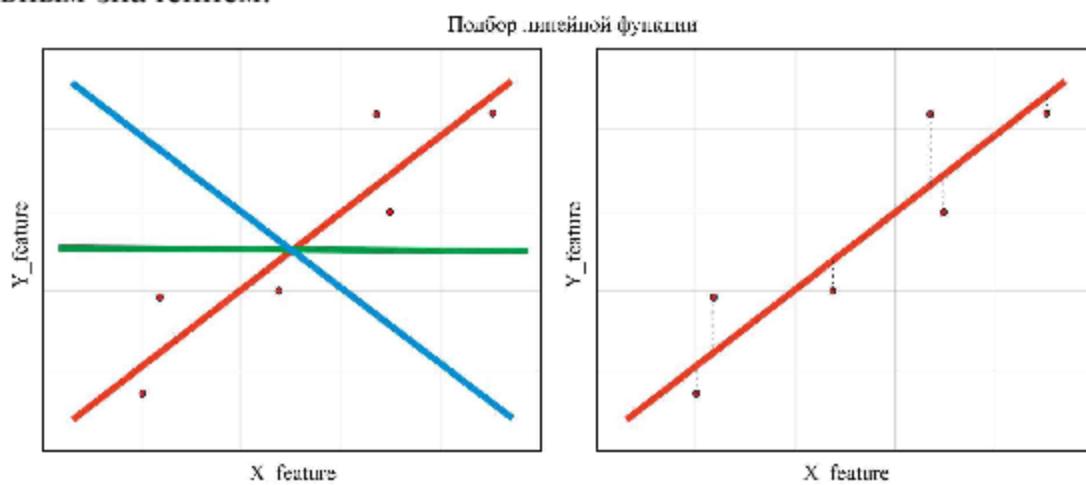


Рис.6.4. Поиск оптимальных параметров  $k$  и  $b$

Если говорить точнее, то будет подсчитана сумма квадратов всех разностей между теоретическим и практическим значением. Стремление данной суммы к нулю свидетельствует об эффективности подбора теоретической функции. Использование линейной регрессии обусловлено простотой модели возможностью быстро проводить обучение и предсказание.

А теперь давайте рассмотрим применение данного алгоритма на конкретных экспериментальных данных. В нашем распоряжение имеются данные по температурному изменению земли и воздуха. Задача заключается в создании модели на основе линейной регрессии, обучении модели, проверки предсказаний модели и анализ эффективности самой модели. Для простоты восприятия программы мы разобьём её на несколько небольших блоков. Каждый блок будет выполнять определённую операцию. Подключим все необходимые нам библиотеки и загрузим таблицу для анализа:

---

```
# Подключение класса LinearRegression из модуля scikit-learn
from sklearn.linear_model import LinearRegression

# Подключение train_test_split из модуля scikit-learn
from sklearn.model_selection import train_test_split

# Подключение библиотеки matplotlib
import matplotlib.pyplot as plt

# Подключение библиотек numpy и pandas
import numpy as np
import pandas as pd

df = pd.read_excel('temp_around.xlsx')

X = df[['temp_ground']] # строго двойные кв. скобки
Y = df['temp_air']
```

Пока в этом блоке нам не известны только модули из библиотеки `sklearn`: `LinearRegression` и `train_test_split`. Первый модуль позволяет реализовать алгоритм линейной регрессии, а второй разбить все наши данные на два набора. Один набор будет использован при обучении системы, а второй при тестировании. Разделение данных также можно выполнить без использования дополнительных методов. Создадим модель, разделим данные, проведём обучение и найдём неизвестные параметры линейной функции.

---

```
# Разбиение данных на два подмножества: для обучения и тестирования (Train-Test Split)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.3,
random_state=1)

# STEP1 - создание экземпляра модели (Instantiating the Model)
model = LinearRegression()
```

```
# STEP2 - обучение модели (Fitting the Model)
model.fit(X_train, Y_train)

# STEP3 - расчёт коэффициентов линейной модели  $Y = \text{intercept}_\text{ } + \text{coef}_\text{ } * X$ 
print(model.intercept_.round(2))
print(model.coef_.round(2))
```

---

Таким образом мы создали новую модель для анализа данных. Теперь давайте попробуем проверить нашу модель на предсказание. Модель будет тестироваться на 1 значении и на наборе тестовых значений.

```
# STEP4 - проверка модели на одном тестовом значении (Prediction)
new_temp_ground = np.array([23.5]).reshape(-1,1) # необходим 2d или DataFrame
print(model.predict(new_temp_ground))

# STEP5 - проверка модели на множество тестовых значений (оценка производительности)
y_test_predicted = model.predict(X_test)
print(y_test_predicted)
```

---

Мы убедились, что наша модель работает и достаточно неплохо предсказывает новые данные. Но так ли она хорошо работает? Для оценки эффективности нашей модели мы будем использовать две характеристики: среднеквадратическая ошибка и R-квадрат.

```
# STEP6 - среднеквадратическая ошибка (mean squared error (MSE) -> 0)
residuals = Y_test - y_test_predicted
print((residuals**2).mean())

# STEP7 - нахождение R-квадрата (R-squared, 0-100%)
# 1 - (residuals**2).sum() / ((Y_test-Y_test.mean())**2).sum()
print(model.score(X_test, Y_test))
```

---

R-квадрат показывает нам, какая доля вариаций может быть описана нашей моделью. Значение меняется в пределах от 0 до 100%. 100% соответствует полной функциональной зависимости. Принято не доверять моделям с R-квадрат менее 50%. Для нашей модели значение составляет R-квадрат = 99%, MSE = 0.00022. Ну и если говорить о полном анализе полученной модели, то было бы неплохо представить графически модель и исходные данные на одном графике.

```
# STEP8 - графический анализ результатов (Residuals)
plt.scatter(X_test, Y_test, label='Экспериментальные данные');
plt.plot(X_test, y_test_predicted, label='Модель', linewidth=3)
plt.xlabel('temp_ground'); plt.ylabel('temp_air')
```

---

```
plt.legend(loc='upper left')
plt.show()
```

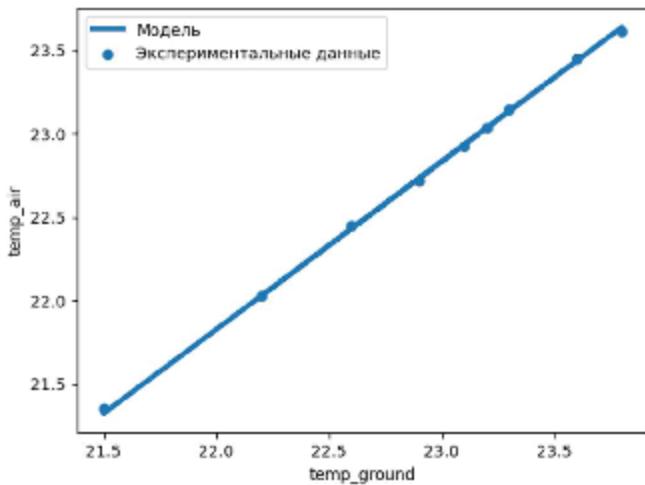


Рис.6.5. Модель линейной регрессии и экспериментальные данные

На этом исследование полученной модели окончено. Дополнительно можно изучить ещё два регрессионных метода анализа: множественная линейная регрессия и полиномиальная регрессия. Подробнее с ними можно ознакомиться на сайте разработчика библиотеки sklearn.

### 6.3. Классификация

По типу входных данных все виды классификации можно разделить на двухклассовые и многоклассовые. Если представлены только два класса для прогнозирования, то это проблема двухклассовой классификации. Например, когда необходимо однозначно ответить на вопрос да или нет после анализа данных. Если у нас больше двух классов, то задача является многоклассовой. Например, классификация видов цветка ириса, которые могут иметь сорт versicolor, virginica или setosa. Разделение видов основана на анализе размеров чашелистика (sepal) и лепестка (petal) цветка.

Наиболее распространенные алгоритмы классификации: логистическая регрессия, метод k-ближайших соседей, деревья решений, наивные байесовские модели, машины опорных векторов, нейронные сети и т. д. В данной части мы будем использовать метод k-ближайших соседей для классификации видов цветка ириса.

Метод машинного обучения на основе классификации имеет своей целью построение функции отображения из входных данных. Аналогичный подход используется в регрессионном анализе. Разница в том, что регрессия использует непрерывные данные, а классификация может использовать дискретные величины или даже категориальные переменные.



Рис.6.6. Цветок ириса

Мы будем использовать базу данных цветка ириса, которая впервые была использована Р.А. Фишером (англ. Sir Ronald Aylmer Fisher). Таблица содержит 150 растений ириса, каждое из которых имеет 4 числовых атрибута: длина чашелистика в см, ширина чашелистика в см, длина лепестка в см и ширина лепестка в см. Задача классификации состоит в том, чтобы предсказать один из трёх видов цветка (*iris-setosa*, *iris-versicolor* или *iris-virginica*) на основе этих атрибутов. Перед применением алгоритмов машинного обучения рекомендуется загрузить таблицу средствами pandas, ознакомиться со структурой и получить предварительную суммарную статистику методом `describe()`. Для просмотра уникальных значений воспользуйтесь методом `value_counts()`.

---

```
import pandas as pd
iris = pd.read_csv('iris.csv')

print(iris['species'].value_counts())
```

---

Мы установили, что в нашей таблице всего 150 цветков. Всего 3 сорта и каждый сорт имеет выборку в 50 единиц. Можно сделать вывод, что набор данных является сбалансированным и классификация допустима. В случае несбалансированного набора необходимо использовать иной алгоритм машинного обучения.

На начальном этапе аналистику важно понять, какой признак объекта использовать при обучении? Здесь можно воспользоваться графическим представлением данных. Построим две зависимости: `sepal-length/sepal-width` и `petal-length/petal-width`.

---

```
import matplotlib.pyplot as plt
import pandas as pd
iris = pd.read_csv('iris.csv')

# создадим цифровой код для каждого сорта
inv_name_dict = {'iris-setosa': 0,
```

```

'iris-versicolor': 1,
'iris-virginica': 2}

# создадим цифровой код для каждого цвета
colors = [inv_name_dict[item] for item in iris['species']]

scatter = plt.scatter(iris['sepal_len'], iris['sepal_wd'], c = colors)
plt.xlabel('sepal length (cm)')
plt.ylabel('sepal width (cm)')
plt.legend(handles=scatter.legend_elements()[0],
labels = inv_name_dict.keys())
plt.show()

```

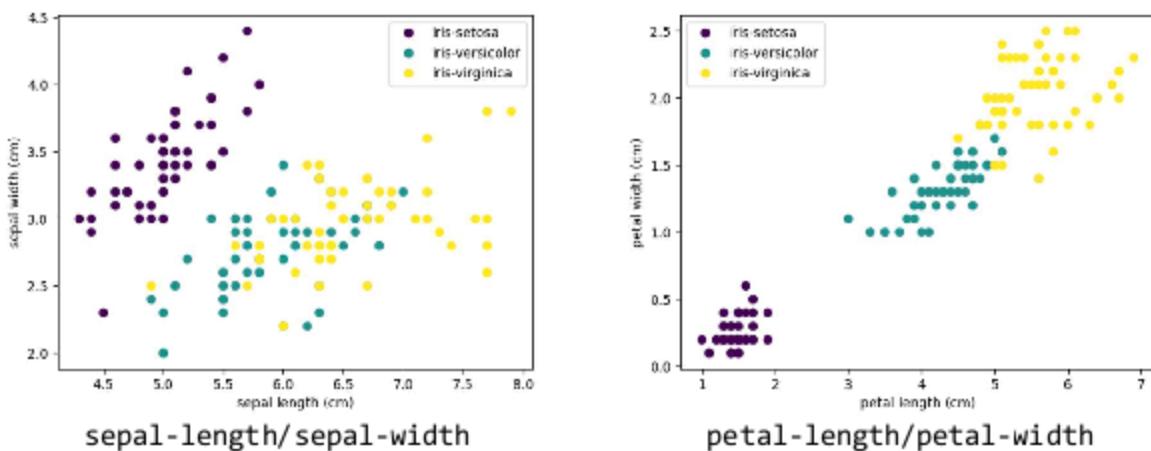


Рис.6.7. Цветок ириса

Графический анализ показал, что длина и ширина лепестков очень сильно коррелирует. Результат свидетельствует о том, что эти атрибуты будут полезны для классификации идентификации разных видов цветка.

Мы будем использовать алгоритм машинного обучения **k-ближайших соседей**. К-ближайших соседей (knn) – это контролируемая модель машинного обучения, которая берет точку данных, анализирует ее k-ближайших соседей, имеющих свою метку, и назначает метку большинства соседей. Понятно, что мы можем ограничить количество соседей, по которым будет проведен анализ. Параметр, отвечающий за количество соседей анализируемых соседей, называется гиперпараметром.

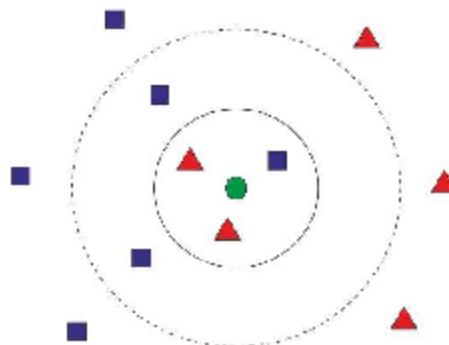


Рис.6.8. Метод k-ближайших соседей

Рассмотрим особенности алгоритма k-ближайших соседей. На рис.6.8 представлен набор значений. Для всех величин определены свои атрибуты, кроме значения в центре (обозначен зелёным). Согласно алгоритму, если мы предлагаем программе определить принадлежность неизвестного по 3-ём соседям, то неизвестный будет красным треугольником, т.к красных треугольников по соседству больше всего. А если мы выбираем алгоритм по 5-ти соседям, то неизвестный будет синим квадратом, поскольку синих квадратов большинство. Теперь вернёмся к нашим цветкам. Понятно, что результаты на рис.6.7 (petal-length/petal-width) позволяют нам использовать этот алгоритм.

Для простоты восприятия программы мы разобьём её на несколько небольших блоков. Подключим все необходимые нам библиотеки, загрузим таблицу для анализа и разделим данные на две части.

---

```
# Подключение библиотеки Pandas
import pandas as pd
iris = pd.read_csv('iris.csv')

# Подключение train_test_split из модуля scikit-learn
from sklearn.model_selection import train_test_split

# Подключение neighbors из модуля scikit-learn
from sklearn.neighbors import KNeighborsClassifier

X = iris[['petal_len', 'petal_wd']] # строго двойные кв. скобки
y = iris['species']

# Разбиение данных на два подмножества: для обучения и тестирования (Train-Test Split)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,
random_state=1, stratify=y)
```

Далее программа должна выполнить следующие шаги: создание модели, обучение модели с n\_neighbors = 5 и проверка модели тестовыми данными.

---

```
# STEP1 - создание экземпляра модели (Instantiating the Model)
# Кол-во соседних значений n = 5
knn = KNeighborsClassifier(n_neighbors=5)

# STEP2 - подгонка модели (Fitting the Model)
knn.fit(X_train, y_train)

# STEP3 - проверка модели на одном тестовом значении (Prediction)
import numpy as np
new_data = np.array([3.76, 1.20])
new_data = new_data.reshape(1, -1)
print(knn.predict(new_data))

# STEP4 - проверка модели на множестве тестовых значений (Prediction)
pred_X_test = knn.predict(X_test)
```

```

print(pred_X_test)

# STEP5 - вероятностное предсказание (Probability Prediction)
y_pred_prob = knn.predict_proba(X_test)
print(y_pred_prob[10:12])

# STEP6 - проверка точности модели
print(knn.score(X_test, y_test))

# STEP7 - построение матрицы неточностей и метрик precision, recall, F-мера
y_pred = knn.predict(X_test)

from sklearn.metrics import classification_report
report = classification_report(y_test, y_pred)
print(report)

from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_pred))

```

---

Из приведённого выше кода для нас является незнакомым только `predict_proba()` и `confusion_matrix()`. Метод рассчитывает вероятностное предсказание величины, т.е. показывает вероятность предсказания в %-ах. Матрица ошибок – это сводная информация в виде матрицы о количестве правильных и неправильных предсказаний, разбитых по каждому классу. Дополнительно методом `classification_report()` подготовлен отчёт по метрикам: `recall` (полнота), `precision` (точность), `f1`-оценка. Для понимания данных метрик рассмотрим пример матрицы ошибок на примере бинарной классификации (1 и 0).

	0 (actual)	1 (actual)
0 (predict)	TP	TN
1 (predict)	FP	FN

TP – true positive (модель ответила “Да” и угадала)

TN – true negative (модель ответила “Нет” и угадала)

FP – false positive (модель ответила “Да” и не угадала)

FN – false negative (модель ответила “Нет” и не угадала)

P = TP + FN количество ответов “Да”

N = TN + FP количество ответов “Нет”

Теперь рассмотрим математическую запись матрик.

$$precision = \frac{TP}{TP + FP} \quad (1)$$

$$recall = \frac{TP}{TP + FN} \quad (2)$$

$$F1 = \frac{2 * recall * precision}{recall + precision} \quad (3)$$

Первая метрика (1) даёт нам информацию о точности всех предсказаний модели среди всех положительных ответов модели (процент корректных предсказаний). Вторая метрика (2) сообщает долю правильных ответов модели среди всех возможных правильных ответах. Метрика (3) объединение (1) и (2). Метрики (1) и (2) можно рассчитать, если у вас построена матрица ошибок. Precision можно определить разделив диагональный элемент матрицы на сумму всех элементов столбца, recall можно определить разделив диагональный элемент матрицы на сумму всех элементов строки.

Далее проведём перекрёстную проверку (K-fold Cross Validation) и перекрёстную проверку с анализом сетки (GridSearchCV).

---

```
# STEP8 - перекрёстная проверка (K-fold Cross Validation)
from sklearn.model_selection import cross_val_score

knn_cv = KNeighborsClassifier(n_neighbors=3)
# 5-fold cv
cv_scores = cross_val_score(knn_cv, X, y, cv=5)
# print each cv score (accuracy)
print(cv_scores)
print(cv_scores.mean())

# STEP9 - перекрёстная проверка (K-fold Cross Validation с GridSearchCV)
from sklearn.model_selection import GridSearchCV
# Создание нового экземпляра модели (Instantiating the Model)
knn2 = KNeighborsClassifier()
# Создание словаря с разными параметрами соседних значений
param_grid = {'n_neighbors': np.arange(2, 10)}
# Анализ всех n_neighbors при 5-кратной перекрёстной проверке
# Обычно используют 5-ти или 10-кратную перекрёстную проверку
knn_gscv = GridSearchCV(knn2, param_grid, cv=5)
# Подгонка модели и вывод лучшего n_neighbors
knn_gscv.fit(X, y)
print(knn_gscv.best_params_)
```

---

Последний блок реализует алгоритм перекрёстной валидации модели. Перекрёстная валидация выполняется с разбиением данных на несколько подмножеств. Каждое подмножество будет использовано при построении модели. Далее проводится проверка точности каждого прохода методом cv\_scores(). Но мы должны взять среднее значение по точности и принять его как единственное верное для данной перекрёстной валидации cv\_scores.mean(). В нашем случае этот показатель равен 95.3%. GridSearchCV производит поиск оптимального значения k-ближайших соседей для данной модели. В данной программе оптимальное значение n\_neighbors = 4.

## 6.4. Деревья решений

**Дерево принятий решений** (дерево классификации или регрессионное дерево) – один из алгоритмов машинного обучения, который позволяет

принимать решения на основе анализа данных и статистики. Основные задачи алгоритма: классификация и регрессия. Условно алгоритм можно представить в виде корня, веток и листьев. В ветках записаны атрибуты целевой функции, в листьях находятся значения целевых функций. Места размещения атрибутов и значений также принято называть узлами. Если узел не имеет дочерних узлов, то его принято называть листовым узлом. Цель метода заключается в том, чтобы создать модель, позволяющую предсказывать значение целевой переменной по входным данным. Проверка условий выполняется при движении от корней к листьям. Каждый внутренний узел соответствует одной входной переменной.

Важным критерием проверки условия является так называемый показатель Gini ( $Gini = 1 - (x/n)^2 - (y/n)^2$ ,  $x$  – положительный ответ,  $y$  – отрицательный ответ,  $n$  – количество образцов), который определяет распределение образцов или загрязнённость (impurity) данных.

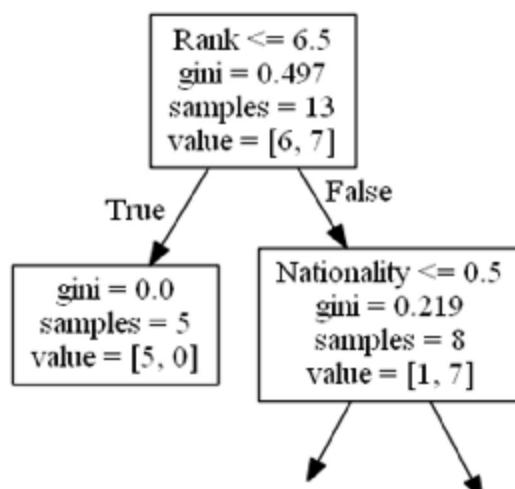


Рис.6.9. Пример дерева принятий решений

Если  $Gini = 0$ , то это значит, что модель смогла однозначно разделить все образцы на классы, если  $Gini = 0,5$ , то все образцы разделены точно посередине. Показатель  $Gini$  не может превышать значения 0,5.

Рассмотрим конкретный пример реализации алгоритма. В качестве датасета будем использовать набор данных, который содержит информацию о погодных наблюдениях за определённый период времени. Наша задача построить модель, позволяющую предсказать возможность выпадения осадков при заданных условиях: температура воздуха, атмосферное давление, скорость ветра, влажность и температура воды.

---

```

# Подключение библиотеки Pandas
import pandas as pd
# Подключение библиотеки sklearn
from sklearn import tree
# Использование модели классификации в деревьях решений
from sklearn.tree import DecisionTreeClassifier
  
```

```

df = pd.read_csv('weather.csv')

# rename и map() позволяют преобразовать категориальные признаки в числа
rename = {'rain': 1, 'sunny': 0}
# целевая переменная
df['rainfall'] = df['rainfall'].map(rename)

# переменные для построения модели
features = ['temp', 'press', 'wind', 'humidity', 'temp_water']

X = df[features]
y = df['rainfall']

# создание экземпляра класса (Instantiating the Model)
dtree = DecisionTreeClassifier()

# подгонка модели (Fitting the Model)
dtree = dtree.fit(X, y)

# предсказание модели (Prediction)
X_new = [15, 750, 4.3, 63, 14]
print(dtree.predict([X_new]))

# если [1] – будет дождь, если [0] – будет солнечная погода
print("[1] rain")
print("[0] sunny")

```

Ранее мы неоднократно подробно рассматривали алгоритмы машинного обучения. Деревья решений аналогичные этапы, что и рассмотренные модели: импорт данных, создание модели, обучение модели и предсказание. Дерево решений для данной модели выглядит следующим образом.

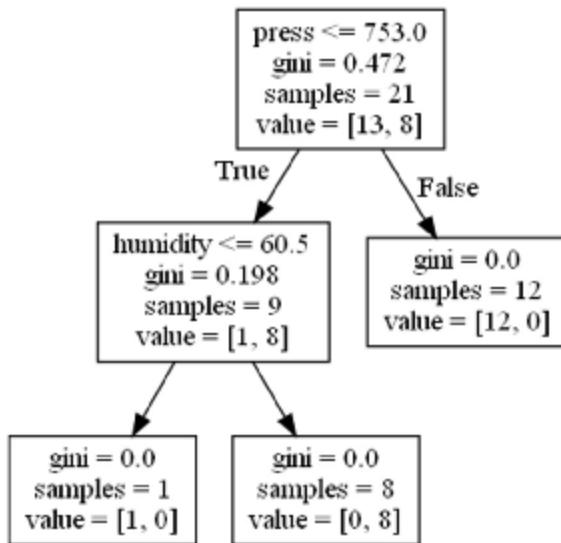


Рис.6.10. Дерево решений для предсказания погоды

Обратите внимание, что в качестве главных признаков модели программа использует данные о давлении и влажности. Это говорит о том, что

данные признаки оказывают наибольшее влияние на целевой признак. Далее представлен листинг программы для построения деревьев решений.

---

```
# Подключение библиотеки Pandas
import pandas as pd
# Подключение библиотеки sklearn
from sklearn import tree
# Использование модели классификации в деревьях решений
from sklearn.tree import DecisionTreeClassifier

# Библиотеки для построения графика
import pydotplus
import matplotlib.pyplot as plt
import matplotlib.image as pltimg

df = pd.read_csv('weather.csv')

# rename и map() позволяют преобразовать категориальные признаки в числа
rename = {'rain': 1, 'sunny': 0}
# целевая переменная
df['rainfall'] = df['rainfall'].map(rename)

# переменные для построения модели
features = ['temp', 'press', 'wind', 'humidity', 'temp_water']

X = df[features]
y = df['rainfall']

# создание экземпляра класса (Instantiating the Model)
dtree = DecisionTreeClassifier()

# подгонка модели (Fitting the Model)
dtree = dtree.fit(X, y)

# построение графика и создание изображения
data = tree.export_graphviz(dtree, out_file=None, feature_names=features)
graph = pydotplus.graph_from_dot_data(data)
graph.write_png('decisiontree.png')

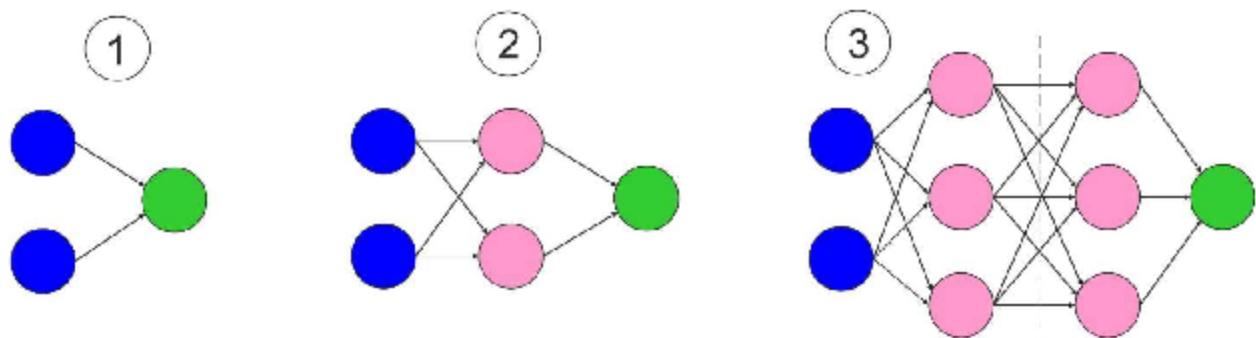
# вывод графика на экран
img=pltimg.imread('decisiontree.png')
imgplot = plt.imshow(img)
plt.show()
```

Дополнительно для построения графика и вывода его на экран мы использовали библиотеку matplotlib и pydotplus.

## 6.5. Нейронные сети

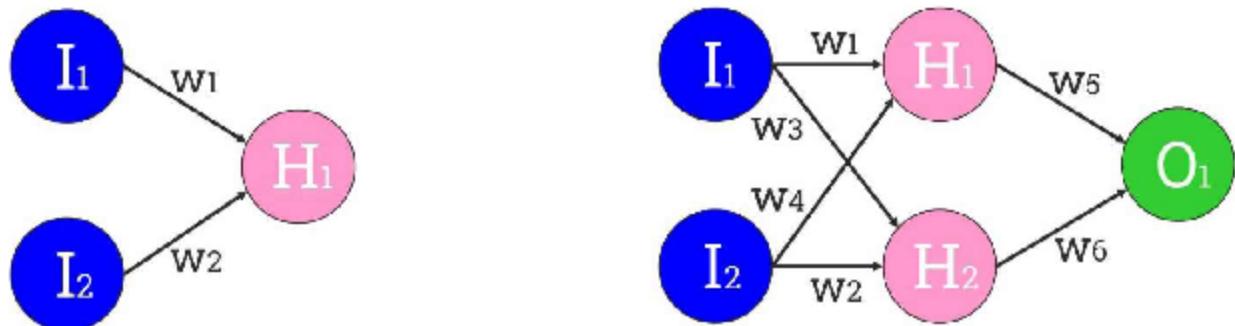
**Нейронная сеть** (искусственная нейронная сеть) в машинном обучении – математическая модель, построенная по принципу биологических нейронных сетей. Аналогично биологической модели, в искусственной

нейронной сети минимальной вычислительной единицей является нейрон. **Нейрон** выполняет приёма данных, выполнения простых операций и передачи информации дальше по цепочке. Простейшей нейронной сетью является персептрон (Perceptron). **Персептрон** – это система, которая состоит из элементов трех разных типов: сенсоров (S-элементы), ассоциативных элементов (A-элементы) и реагирующих элементов (R-элементы). Разделяют следующие типы персептронов: однослоинный (1), персептрон с одним скрытым слоем (2), многослойный персептрон (3). Выбор многослойной структуры позволяет улучшить алгоритм вычислений, но и нагружает вычислительную машину.



Типы нейронных сетей

Теперь рассмотрим принцип выполнения вычислений нейронной сетью. Обозначим слой входных нейронов как I, слой скрытых нейронов как H, слой выходных нейронов буквой O.



Входной слой определяет количество известных параметров, которые описывают выборку данных. Связь между входным нейроном и нейроном в скрытом слое называется весом. В случае двух входных нейронов и одного выходного мы получим следующие соотношения:

$$H_{1\text{input}} = (I_1 * w_1) + (I_2 * w_2), \\ H_{1\text{output}} = f_{\text{activation}}(H_{1\text{input}}).$$

Если рассматривать биологический нейрон, то аналогом веса является синапс. Искусственные нейроны оперируют с данными в диапазоне  $[0, 1]$  или  $[-1, 1]$ . Для приведения данных к такому виду используется функция активации, которая выполняет нормализацию. Если принять то, что

совокупность входных данных определяется как вектор  $X = \{x_1, x_2, \dots x_n\}$ , веса нейрона  $W = \{w_1, w_2, \dots w_n\}$ , то вектор выходных данных  $Y$  может быть найден как  $Y = WX$  или  $Y = f(X) = \phi(WX)$ . Основные виды функций активации: линейная, сигмоид (логистическая) и гиперболический тангенс. Обучение новой сети происходит следующим образом:

1. на первом этапе обучения на вход подаётся последовательность данных, на выходе рассчитывается значение  $f(X)$  при произвольных значениях матрицы  $W$ ;
2. на втором этапе рассчитывается разность между правильными ответами и теми, которые получились при произвольной матрице  $W$ , происходит нахождение вектора ошибок;
3. на третьем этапе происходит корректировка весовых коэффициентов (множественные итерации) до тех пор, пока не будет минимизирован вектор ошибок.

Для нейронной сети можно выделить несколько фундаментальных понятий: эпоха (epoch), размер партии (batch size) и итерации (iterations). **Эпоха** – это процесс, при котором происходит проход всего набора тренировочных данных через нейронную сеть. **Размер партии** – пакет данных определённого размера. **Итерации** – количество пакетов данных для одной эпохи. Например, если у нас в наборе тренировочных данных имеется 1000 объектов, то их можно разбить на пакеты размером по 100 объектов. Для завершения эпохи, в этом случае, потребуется 10 итераций.

В качестве примера создания и обучения нейронной сети мы будем использовать хорошо нам известный набор данных для цветков ириса. В качестве алгоритма машинного обучения будет использован контролируемый алгоритм обучения многослойного персептрона (**MLP**).

---

```
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np

iris = pd.read_csv('iris.csv')

X = iris[['petal_len', 'petal_wd', 'sepal_len', 'sepal_wd']] # строго двойные
# КВ. скобки
y = iris['species']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,
random_state=1, stratify=y)

clf = MLPClassifier(solver = 'lbfgs', alpha=1e-5, max_iter = 1000, random_state
= 0)

scale = StandardScaler()
clf = clf.fit(X_train, y_train)
```

```
X_test_scaled = scale.fit_transform(X_test)

print('prediction: ', clf.predict(X_test_scaled))
print(clf.score(X_test_scaled, y_test).round(2))
```

---

Для полученной модели на тестовом наборе данных точность предсказаний соответствует 91%. Это значение можно повысить, если подобрать количество скрытых единиц (нейронов) и скрытых слоёв (нейронных слоёв) в атрибуте `hidden_layer_sizes` функции `MLPClassifier()`. Попробуйте самостоятельно проанализировать результат с `hidden_layer_sizes = (9,)`. В этом случае модель по умолчанию имеет один входной слой, один скрытый слой с девятью нейронами и один выходной слой.

## 6.6. Кластеризация

Кластеризация – это тип неконтролируемого обучения, который позволяет группировать данные на основе общих признаков. Алгоритм очень удобен, когда априори нет никакой исходной информации о метках данных. В качестве примера использования кластеризации можно указать группировку документов, музыки или фильмов на основе их признаков. Ещё одним примером кластеризации является группировка покупателей магазина на основе их покупательской способности. Группа или кластеры образуются на основе общих признаков, отличающихся от признаков в других кластерах. Известно более 100 алгоритмов кластеризации, порядком 10 из них реализованы в библиотеке `sklearn`. В документации к библиотеке можно ознакомиться с ними.

В данной части мы познакомимся с вами с моделью на основе центроидов (**Centroid based models**), когда каждый кластер представлен одним средним вектором (например, `k-means`). Предполагая, что существует  $n$  точек данных, алгоритм работает следующим образом:

1. выбор  $k$  случайных точек в качестве центров кластеров (центроидов);
2. формирование кластера (группировка кластера) на основе анализа расстояния от точек до центроида;
3. обновление центроида на основе анализа кластера и анализа среднего значения данных в группе;
4. повторение (2) и (3) до тех пор, пока изменения в кластерах не будут минимальны.

Каким образом выполняется (2) в алгоритме `k`-средних? Один из способов связан с использованием Евклидовой метрики, прямая линия между двумя точками данных определяется как корень квадратный из суммы квадратов разности значений координат.

---

```
import numpy as np
```

```
x1 = np.array([0, 2])
x2 = np.array([4, 0])
result = np.sqrt(((x1-x2)**2).sum())
print(np.sqrt(result).round(2))
```

Для примера кластеризации мы будем использовать датасет готовых данных `load_wine`, который содержит набор признаков (химический состав) для набора неизвестных вин. Задача заключается в возможности разбиения всех данных на определённые группы (кластеры). Разобьём задачу на несколько подзадач:

## 1. Загрузка данных. Предварительный анализ.

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_wine

data = load_wine()
wine = pd.DataFrame(data.data, columns=data.feature_names)
print(wine.shape)
print(wine.columns)
```

Предварительный анализ показал, мы в таблице имеется 13 признаков (alcohol, malic acid, ash, alcalinity of ash, magnesium, total phenols, flavanoids, nonflavanoid phenols, proanthocyanins, color intensity, hue, od280/od315 of diluted wines, and proline.) и 178 образцов неизвестного вина.

## 2. Графическое представление данных.

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.datasets import load_wine
from pandas.plotting import scatter_matrix

data = load_wine()
wine = pd.DataFrame(data.data, columns=data.feature_names)
scatter_matrix(wine.iloc[:,[0,5]])
plt.show()
```

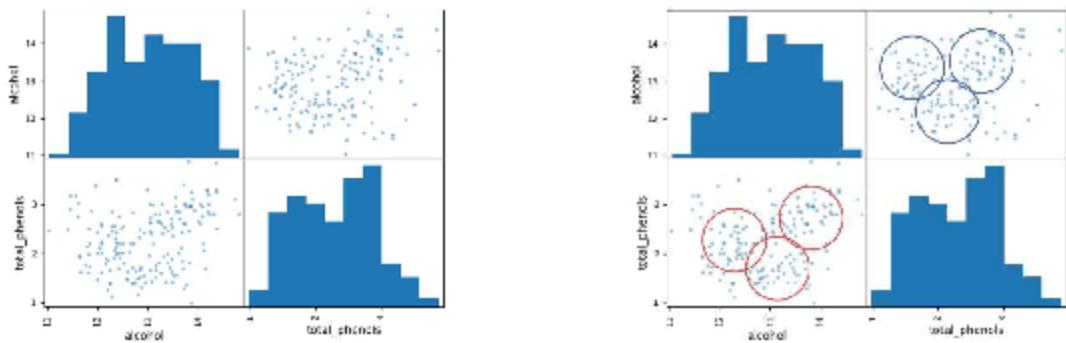


Рис.6.11. Построение scatter\_matrix выбранной части данных

Качественно график показывает, что в нашем наборе есть три выраженные группы данных, для которых можно будет определить центроид и образовать кластер.

### 3. Стандартизация. Масштабирование.

---

```

import numpy as np
import pandas as pd
from sklearn.datasets import load_wine

data = load_wine()
wine = pd.DataFrame(data.data, columns=data.feature_names)

X = wine[['alcohol', 'total_phenols']]

from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
scale.fit(X)

print(scale.mean_)
print(scale.scale_)

X_scaled = scale.transform(X)
print(X_scaled.mean(axis=0))
print(X_scaled.std(axis=0))

```

---

Стандартизация данных будет сведена к операции масштабирования данных. Поскольку в основе кластеризации в данном случае лежит алгоритм k-средних работает лучше, если каждый атрибут имеет сходные масштабы. В результате масштабирования мы получили, что среднее значение стремится к нулю, стандартное отклонение к единице.

---

```

import pandas as pd
from sklearn.datasets import load_wine
from sklearn.preprocessing import StandardScaler

data = load_wine()

```

```
wine = pd.DataFrame(data.data, columns=data.feature_names)

X = wine[['alcohol', 'total_phenols']]

scale = StandardScaler()
scale.fit(X)
X_scaled = scale.transform(X)

from sklearn.cluster import KMeans
# Создание нового экземпляра модели (Instantiating the Model)
kmeans = KMeans(n_clusters=3)
# Обучение модели (Fitting the Model)
kmeans.fit(X_scaled)
# Проверка модели (Prediction)
y_pred = kmeans.predict(X_scaled)
print(y_pred)

# Предсказание на одном образце
X_new = np.array([[13, 2.5]])
X_new_scaled = scale.transform(X_new)

print(kmeans.predict(X_new_scaled))
```

---

В результате мы получим массив данных, в котором представлены образцы вина, разбитые на три кластера. Также мы можем проверить предсказание нашей модели на одном конкретном образце.

#### 4. Оптимизация количества кластеров.

Возникает вопрос, какое значение  $k$  (количество кластеров) является оптимальным при построении модели? В этом случае используют следующий алгоритм оценки. Программа должна найти параметр `inertia`, который содержит суммарную информацию о близости точек в кластере к её центроиду. С увеличением количества кластеров будет уменьшаться численное значение данного параметра. Аналитик получает график зависимости количества кластеров от инерции и по графику определяет оптимальное значение параметра в той точке, где останавливается её резкое уменьшение.

---

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_wine
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

data = load_wine()
wine = pd.DataFrame(data.data, columns=data.feature_names)

X = wine[['alcohol', 'total_phenols']]

scale = StandardScaler()
```

```

scale.fit(X)
X_scaled = scale.transform(X)

kmeans = KMeans(n_clusters=3)
kmeans.fit(X_scaled)
print(kmeans.inertia_)

import numpy as np
# Вычисление инерции при разном кол-ве кластеров
inertia = []
for i in np.arange(1, 11):
    km = KMeans(
        n_clusters=i
    )
    km.fit(X_scaled)
    inertia.append(km.inertia_)

plt.plot(np.arange(1, 11), inertia, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()

```

---

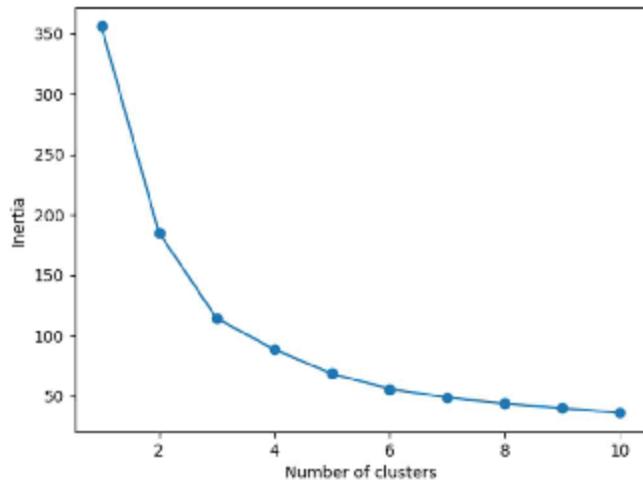


Рис.6.12. График зависимости кол-ва кластеров от инерции

На графике видно, что резкий спад заканчивается на значении  $n = 3$ . Именно это значение необходимо использовать при построении модели кластеризации. Мы показали алгоритм нахождения оптимального значения  $n$  на 2-ух признаках, аналогично можно определить значение для всех признаков в таблице.



## Задания для самостоятельной работы

### Исследовательский анализ данных и машинное обучение

Для выполнения заданий по машинному обучению необходимо скачать датасет, который содержит информацию о медицинских анализах группы людей, страдающих сердечными заболеваниями. Дополнительная информация о наборе данных:

---

```
# Heart Disease UCI
'https://www.kaggle.com/ronitf/heart-disease-uci'

# Подготовленный датасет для выполнения задания находится здесь
'https://github.com/mculab64/data_science/tree/main/ds_datasets/heart.csv'
```

---

Используемый датасет ограничен данными Cleveland Clinic Foundation (cleveland.data). Исходная база имеет 76 признаков. Подготовленный датасет имеет 14 наиболее важных признака. Информация об основных атрибутах представлена в таблице.

1. age	Возраст.
2. sex	Пол (1 = м, 0 = ж).
3. cp	Тип боли в груди: 1: типичная стенокардия; 2: атипичная стенокардия; 3: боль, не связанная со стенокардией; 4: бессимптомный.
4. trestbps	Артериальное давление, мм рт. ст.
5. chol	Содержание холестерина в крови, мг/дл.
6. fbs	Уровень сахара в крови натощак > 120 мг/дл

	(1 = да, 0 = нет).
7. restecg	Результаты электрокардиографии в покое: 0: норма; 1: наличие аномалии сегмента ST-T; 2: гипертрофия левого желудочка.
8. thalach	Максимальная частота сердечных сокращений.
9. exang	Стенокардия, вызванная физической нагрузкой (1 = да; 0 = нет).
10. oldpeak	Изменение сегмента ST, вызванное нагрузкой.
11. slope	Наклон сегмента ST при пиковой нагрузке: 1: восходящий; 2: плоский; 3: нисходящий.
12. ca	Количество крупных сосудов (0-3), окрашенных при рентгеноскопии.
13. thal	3 = норма, 6 = исправленный дефект; 7 = обратимый дефект.
14. num (target) (прогнозируемый атрибут)	Диагностика сердечных заболеваний (ангиографический статус болезни): 0: сужение диаметра < 50% (mild disease); 1: сужение диаметра > 50% (significant disease).

Используя библиотеку matplotlib.pyplot, numpy, pandas и sklearn выполните следующие задания исследовательский анализ различными машинными алгоритмами. В качестве целевого атрибута необходимо использовать атрибут num (target). Алгоритмы должны содержать 4 обязательных этапа машинного обучения: загрузка данных (import), создание модели (instantiate), обучение (fit), предсказание (predict). Для выполнения задания используйте алгоритмы контролируемого обучения:

1. классификация;
2. деревья решений;
3. нейронные сети.

Для оценки эффективности построенных моделей используйте стандартные метрики. Сравните полученные модели для разных алгоритмов между собой.



## 7. Сбор и хранение данных

Аналитик данных выполняет детальный анализ уже собранных. Форматов хранения данных существует достаточно много. Существует ли универсальный способ хранения данных? К сожалению, нет. Например, для хранения небольшого количества данных, можно использовать табличные значения в формате csv или excel. Для хранения большого количества данных лучше использовать специальные базы данных SQL. Мы выяснили, что существует много различных форматов, но есть ли универсальный инструмент для работы с данными на Python? Такую возможность представляет библиотека pandas, в которой используется специальный API (англ. application programming interface) I/O – способ взаимодействия со сторонними приложениями и программами. В таблице ниже представлены некоторые методы библиотеки для чтения и записи данных в файл.

Функции чтения	Функции записи	Назначение
<code>read_csv</code>	<code>to_csv</code>	Чтение/Запись csv или txt файла
<code>read_excel</code>	<code>to_excel</code>	Чтение/Запись excel файла
<code>read_sql</code>	<code>to_sql</code>	Чтение/Запись данных базы sql
<code>read_json</code>	<code>to_json</code>	Чтение/Запись json данных
<code>read_html</code>	<code>to_html</code>	Чтение/Запись html файла

Далее мы рассмотрим особенности работы с данными методами.

### 7.1. Работа с файлами формата csv и excel

Ранее мы большинстве примеров работы с файлами использовали файлы формата csv и excel. Давайте ещё раз рассмотрим примеры работы с форматом csv. Для предварительного анализа данных в таблице можно воспользоваться функцией `info()`. Данная функция покажет структуру таблицы, названия столбцов, количество элементов и тип данных.

```
# Подключение библиотеки pandas
import pandas as pd
# Применение метода read_csv
```

```
df = pd.read_csv('iris.csv')
# Сводная информация о таблице данных
print(df.info())
```

Получение среза данных можно несколькими способами. Можно выполнить выбор конкретных столбцов данных непосредственно при загрузке файла, а можно сделать срез, используя срез `loc[]`.

```
import pandas as pd
# Загрузка определённых столбцов из таблицы, id – индексный столбец

df = pd.read_csv('iris.csv', usecols = ['id', 'sepal_len', 'sepal_wd'],
index_col = ['id'], skiprows = 0, nrows = 50)

# Срез данных: первые 6 строк и 3 столбца
df_reject = df.loc[:5, :]
print(df_reject)

# Сохранение нового файла iris_rej.csv, разделитель ','
df_reject.to_csv('iris_rej.csv', sep = ',')
```

В результате выполнения программы был создана новая таблица, которая содержит 6 строк, 3 столбца и данные разделены запятой. Аналогично можно выполнить загрузку и обработку данных excel файлов.

```
import pandas as pd
# Подключение библиотеки для работы с xlsx
import openpyxl
# Загрузка определённых столбцов из таблицы
df = pd.read_csv('iris.csv', usecols = ['id', 'sepal_len', 'sepal_wd'])

# Срез данных: первые 6 строк и 3 столбца
df_reject = df.loc[:5, :]
print(df_reject)

df_reject.to_excel('iris.xlsx', startrow=0, startcol=0, sheet_name = 'Книга1')
```

В примере дополнительно использована библиотека для работы с excel файлами `openpyxl`. Параметры `startrow` и `startcol` позволяют установить начальную ячейку для вставки данных в таблицу.

## 7.2. Работа с файлами формата txt

Для работы с текстовыми файлами можно использовать метод `read_table()` или `to_csv()`. Для сохранения данных используем метод `to_csv()`.

```
import pandas as pd
```

```
df = pd.read_csv('iris.txt', sep = ',')
# df = pd.read_table('iris.txt', sep = ',')

# Срез данных: первые 6 строк и все столбцы
df_reject = df.loc[:5, :]
print(df_reject)

df.to_csv('iris_rej.txt', sep='\t', encoding='utf-8')
```

---

Обратите внимание на параметр `sep`, который позволяет определить разделитель данных. Метод `read_table()` может быть использован для парсинга данных текстового файла с использованием регулярных выражений.

### 7.3. Чтение и запись JSON-файлов

Теперь рассмотрим ещё один популярный формат для хранения данных. Он называется JSON (JavaScript Object Notation) – объектная запись javascript. Каждый элемент этого файла имеет сходство со словарём. Вся конструкция содержится в квадратных скобках, в которых определены объекты в фигурных скобках с данными `ключ:значение`. Удобство данного формата обусловлено возможностью передачи содержимого одной строкой (HTTP-запрос). После передачи строку можно снова восстановить до исходного объекта. Ключи в таком формате взяты в двойные кавычки. Содержимое может быть: строкой, числом, булевым значением, массивом или объектом. В Python для работы с файлами JSON используется специальный модуль `json`. Существует и обратный метод `json.dumps`, который выполняет преобразование из Python в JSON. Формат может быть использован на разных платформах и в разных языках программирования.

---

```
import pandas as pd
import json

df = pd.read_csv('iris.txt', sep = ',')

# Срез данных: первые 6 строк и все столбцы
df_reject = df.loc[:5, :]
print(df_reject)

df.to_json('iris.json')
```

---

В примере показана обработка и преобразование текстового файла в файл формата json. В результате получится файл, данные в котором хранятся в одну строку.

## 7.4. Чтение html-файлов из интернета, парсинг данных, API

В pandas присутствуют модули для чтения html-файлов (локальных или с URL-адресов). Можно использовать дополнительные пакеты **LXML**, **Html5Lib** и **BeautifulSoup4**. Все они позволяют выполнять чтение и запись html-файлов. Дополнительно для чтения файлов в интернет можно использовать метод `get()` из модуля **requests**.

---

```
import requests
# Адрес запрашиваемой страницы
URL = 'https://elteha.ru/index.php'

# Сохранение запроса в переменную result
result = requests.get(URL)

# Проверка ответа сервера
print(result.status_code)
# Вывод результатов в консоль
print(result.text)
```

Понятно, что сервер передаёт всё содержимое html страницы. Для человека, который не знает языка гипертекстовой разметки, трудно будет разобрать с содержимым ответа. В этом случае можно использовать определенный фильтр, который можно создать на основе регулярного выражения. В Python для этой задачи используется библиотека **re**.

---

```
import requests
import re
URL = 'https://elteha.ru/index.php'

result = requests.get(URL).text

# Поиск всех вхождений, удовлетворяющих условиям
found_parts = re.findall('Проектирование[А-яА-з- ]+', result)
print(len(found_parts))
print(found_parts)
```

Вся страница сохраняется в переменную, а потом, используя регулярное выражение, в тексте находятся все вхождения, которые начинаются со слова “Проектирование”. Результат поиска является списком. Поиск данных на странице можно упростить, если заранее известна структура документа DOM (Document Object Model). Дело в том, что любой html-документ имеет свою объектную модель. Если заранее известен элемент, в котором находятся данные, известны атрибуты элемента, то можно сформировать фильтр для поиска данных. В этом случае можно использовать библиотеку **BeautifulSoup**.

```
import pandas as pd
import requests
from bs4 import BeautifulSoup

URL = 'https://elteha.ru/index.php'
result = requests.get(URL).text
soup = BeautifulSoup(result, 'lxml')

head_title = []
for row in soup.find_all('p', attrs = {'class':'top_text_1'}):
    head_title.append(row.text)

foot_title = []
for row in soup.find_all('p', attrs = {'class':'bottom_text_1'}):
    foot_title.append(row.text)

data = pd.DataFrame({'head-title': head_title, 'foot-title': foot_title,})
print(data.head(5))
```

---

Страница сохраняется в переменную. Далее содержимое анализируется и выделяются те части документа, которые соответствуют условиям: текст размещён между тегами <p></p>, в качестве атрибута используются 2 класса (top\_text\_1, bottom\_text\_1). Данные после анализа добавляются в 2 независимых списка. Последний шаг создание DataFrame и вывод 5 первых строк в консоль. Подобный подход поиска данных на странице html называется **парсингом HTML**.

И последний пример, который мы рассмотрим, это использование специального интерфейса передачи данных – API. Если коротко, то это правило взаимосвязи двух программ. Рассмотрим пример использования API погоды на сайте openweathermap.org.

---

```
import requests
import json

BASE_URL = 'http://api.openweathermap.org/data/2.5/weather'

s_city = 'Saratov, RU'
appid = 'e9e0185f6cdf9bcb75dd543cb70738d7'

params = {'q': s_city, 'units': 'metric', 'APPID': appid}

response = requests.get(BASE_URL, params = params)
response_parsed = json.loads(response.text)

temp = response_parsed['main']['temp']
humidity = response_parsed['main']['humidity']

print('Погода в Саратове: температура t = {} градусов, влажность H = {}%'.
format(temp, humidity))
```

---

В ответ на наш запрос сервер формирует строку формата json. В этой строке содержится вся доступная информация об интересуемом нас городе. Программа выводит текущую температуру и влажность в городе.

## 7.5. Чтение из базы данных SQL, запись в базу данных SQL

Перед работой с базами данных определим несколько фундаментальных понятий. **База данных** – это структура для хранения различной информации. Для работы с данными необходимы специальные правила, которые позволяют структурировать данные, добавлять данные, извлекать данные и создавать новые локальные области для хранения данных – **таблицы**. Именно из множества таблиц состоит база данных. Если среди таблиц наблюдается связь (таблицы имеют общие столбцы с данными), то такие базы называются **реляционными**. Таблиц предstawлены совокупностью строк и столбцов. Управление базами данных осуществляется специальной системой, которая называется системой управления базой данных **СУБД**. Например, система PostgreSQL. С подробной документацией к системе можно ознакомиться на сайте разработчика: <https://postgrespro.ru/docs/postgresql/9.6/index>. В основе системы управления используется **язык структурированных запросов SQL** (англ. structured query language). Именно этот язык позволяет формировать сложные запросы к базе данных и выполнять предварительную обработку данных.

Библиотека pandas позволяет проводить чтение данных из любой базы данных SQL, которая поддерживает специальные адаптеры данных Python интерфейса DB-API. Чтение выполняется с помощью метода `io.sql.read_sql()`, запись можно выполнить с помощью метода `to_sql()` таблицы DataFrame. Рассмотрим пример создания базы данных, создания таблицы и чтение/запись данных.

### 1. Создание базы данных в DB Browser for SQLite

Для начала нам нужно скачать программу DB Browser for SQLite, которая позволяет создавать и управлять базами данных. Скачать программу можно на сайте разработчика: <https://sqlitebrowser.org/dl/>. Установите программу, запустите её и через меню создайте новую базу данных. В моём случае `iris_db.sqlite3`

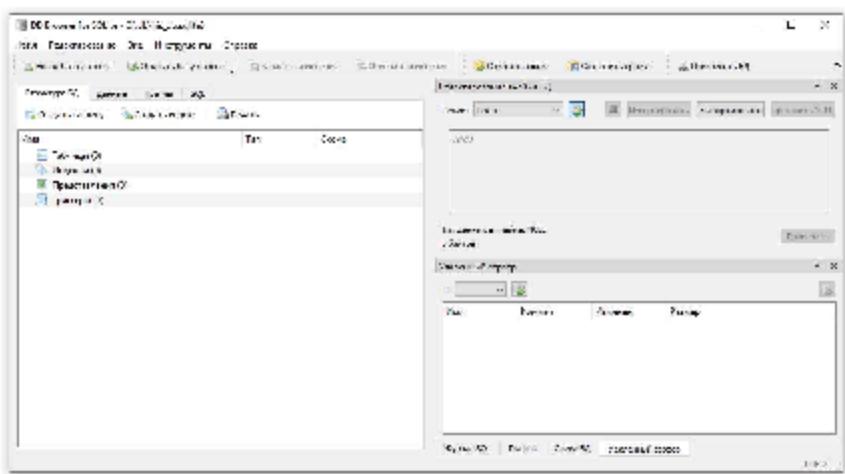


Рис.6.1. Окно программы DB Browser for SQLite

## 2. Создание программы для подключения к базе данных

Подключение к базе данных выполняется с помощью библиотеки sqlite3 в Python. Нам необходимо выполнить следующее. В программе на Python выполнить загрузку (создать DataFrame) данных таблицы хорошо нам известных цветков ириса в формате csv и отправить полученные данные с помощью метода to\_sql() в ранее созданную базу данных iris\_db.sqlite3.

---

```
import pandas as pd
import sqlite3

df = pd.read_csv('iris.csv')

# Подключение к базе данных
connection = sqlite3.connect('iris_db.sqlite3')

# Отправка данных DataFrame в базу (таблица создаётся автоматически)
df.to_sql('iris_table', connection, if_exists="append")

# Подтверждение отправки данных и закрытие соединения.
connection.commit()
connection.close()
```

---

Отметим, что все файлы (iris.csv, db.py и iris\_db.sqlite3) в данном случае находились в одной локальной директории. Для проверки загруженных данных вернитесь в программу DB Browser for SQLite и посмотрите изменения в базе данных iris\_db.sqlite3 во вкладке данные.

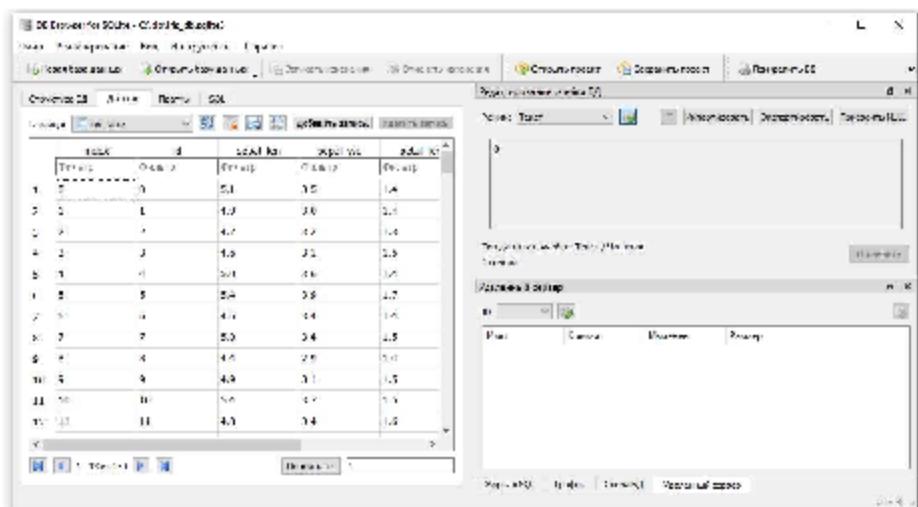


Рис.6.2. Окно программы DB Browser for SQLite с таблицей iris\_table

### 3. Чтение данных из базы

Ранее мы сказали, что чтение данных выполняется с помощью метода io.sql.read\_sql(). Загрузим таблицу iris\_table из базы данных iris\_db.sqlite3 в нашу программу.

```
import pandas as pd
import sqlite3

# Подключение к базе данных
connection = sqlite3.connect('iris_db.sqlite3')

# Чтение всего * содержимого из таблицы iris_table
iris_table = pd.read_sql('SELECT * FROM iris_table',
                         connection, index_col = 'index')
connection.close()

print(iris_table[:5])
```

Первый атрибут в методе io.sql.read\_sql() определяет условие запроса к базе данных. Здесь кроется вся мощь языка SQL. Запросы позволяют существенно упростить процесс чтения данных из базы. Рассмотрим ещё один пример.

```
import pandas as pd
import sqlite3

connection = sqlite3.connect('iris_db.sqlite3')

# Чтение данных по условию sepal_len > 5 AND species='iris-setosa'
query = "SELECT * FROM iris_table WHERE sepal_len > 5 AND species='iris-setosa'"

iris_table = pd.read_sql(query, connection, index_col = 'index')
connection.close()
```

```
print(iris_table[:10])
```

В программе был создан отдельный запрос, который позволил с помощью команды WHERE создать условие выбора определённых данных. В нашем случае мы выбрали все цветки вида iris-setosa с длиной чашелистика больше 5 см. Количество условий не ограничено. Для составления нужных запросов существует специалист, который называется инженер данных. В отличие от аналитика данных, исследующего данные, инженер данных помогает упростить процесс выгрузки данных из базы.

## 7.6. Операторы для работы с базой данных в библиотеке sqlite3

В предыдущей части мы научились работать с базой данных через библиотеку pandas. Теперь давайте рассмотрим возможности работы с базами данных на примере библиотеки sqlite3. Любой начинающий специалист должен понять и освоить принцип работы нескольких операторов языка SQL. Среди операторов можно выделить следующие (см. таблицу).

Оператор	Назначение
<b>CREATE TABLE</b>	Создание новой таблицы в базе данных
<b>SELECT (cols)</b>	Выбор определённых столбцов в таблице
<b>FROM (table)</b>	Выбор таблицы из открытой базы данных
<b>WHERE</b>	Условие выбора данных
<b>AND</b>	Логический оператор (*)
<b>INSERT</b>	Вставка в таблицу с данными нового значения
<b>DELETE</b>	Удаление из таблицы указанного значения
<b>UPDATE</b>	Обновление уже существующих данных в таблице

Рассмотрим пример создания новой таблицы fruit в базе данных db.sqlite3, используя только библиотеку sqlite3.

```
import sqlite3 as sq

# Соединение с базой данных db
con = sq.connect("db.sqlite3")

# Создание курсора (объекта для работы с БД)
cur = con.cursor()

# Создание новой таблицы
cur.executescript("""
    CREATE TABLE IF NOT EXISTS fruit (
        fruit_id INTEGER PRIMARY KEY AUTOINCREMENT,
        fruit TEXT,
        price INTEGER)
""")

# Закрытие соединения
con.commit()
```

```
con.close()
```

---

Теперь добавим несколько фруктов в нашу таблицу.

---

```
import sqlite3 as sq

# Данные о фруктах
fruit = [
    ('Apple', 1.5),
    ('Orange', 2.0),
    ('Banana', 1.7),
    ('Peach', 1.0),
    ('Grape', 2.5)
]

con = sq.connect("db.sqlite3")
cur = con.cursor()

# Вставка данных в таблицу
cur.executemany("INSERT INTO fruit VALUES(NULL,?,?)", fruit)

con.commit()
con.close()
```

---

Извлечём информацию из базы данных и выведем её на экран.

---

```
import sqlite3 as sq

con = sq.connect("db.sqlite3")
cur = con.cursor()

# Чтение всех данных из таблицы fruit
cur.execute("SELECT fruit, price FROM fruit")

# rows = cur.fetchall() - вывод всех данных
# rows = cur.fetchone() - вывод одного значения
# rows = cur.fetchmany(4) - вывод 4 значений

print(rows)

con.commit()
con.close()
```

---

Удалим некоторое значение из таблицы, выбрав его по id.

---

```
import sqlite3 as sq

con = sq.connect("db.sqlite3")
cur = con.cursor()

# Удаление данных из таблицы с id = 3
```

```
cur.execute("DELETE FROM fruit WHERE fruit_id = 3")  
  
# Обновим данных в таблице с id = 1  
cur.execute("UPDATE fruit SET price = 2.2 WHERE fruit_id = 1")  
  
con.commit()  
con.close()
```

При работе с оператором SELECT существует множество различных условий, которые помогают группировать, сортировать и обрабатывать данные запроса. В рамках данного материала мы не будем останавливаться на них. Рекомендуется обратиться к литературе на данную тему и самостоятельно их изучить.

## 7.7. Представление результатов в Jupyter Notebook – IPython

Ещё одним мощным инструментом для обработки экспериментальных данных и отображения результатов анализа (Data Science) является веб-приложение с открытым исходным кодом **Jupyter Notebook**. Ознакомиться с документацией можно на сайте <https://jupyter.org/>. В данном разделе мы ознакомимся с возможностями работы Jupyter Notebook – IPython, позволяющего выполнять скрипты, написанные на языке Python. Для установки веб-сервера приложения можно воспользоваться установщиком pip.

```
pip3 install jupyter # Для Python 3
```

С помощью команды cd в консоли можно определить папку для сохранения проектов. Запуск приложения выполняется следующей командой.

```
jupyter notebook # Запуск приложения
```

Приложение запустится автоматически и в браузере будет открыта страница по адресу <http://localhost:8888/tree>. В начале работы можно загрузить уже существующий проект (Upload) или создать новый (New). После создания нового проекта откроется основное окно программы.

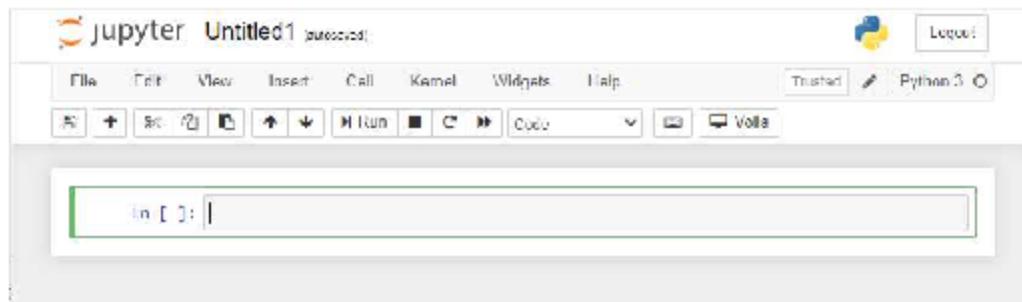


Рис.6.3. Окно программы Jupyter Notebook

В первой строке указано выражение In [ ], которое свидетельствует о возможности ввода информации. Основные блоки для вставки **code** (кода) и **markdown** (специальная разметка текста). Рассмотрим небольшой пример вывода текста методом print(). Для запуска скрипта нажмите «Run».

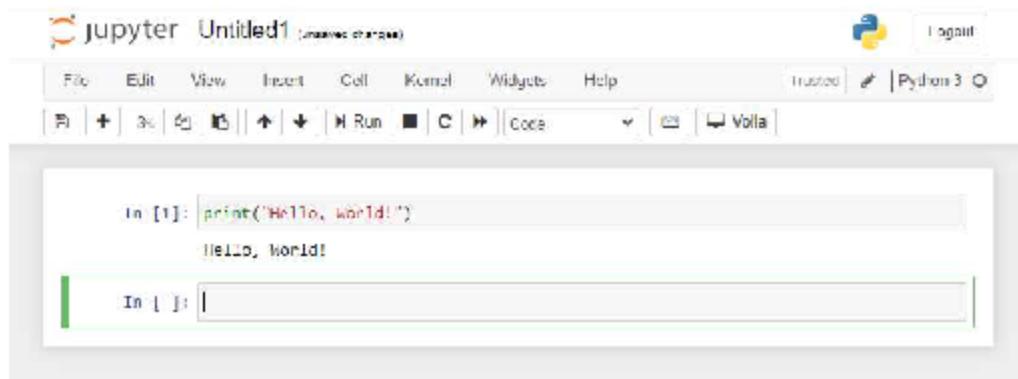


Рис.6.4. Запуск скрипта Python в Jupyter Notebook

Значение в In [ ] изменилось на In [1], цифра определяет порядок выполнения скриптов. С помощью разметки **markdown** можно разместить текст. Заголовок начинается с символа #, для выделения текста курсивом используется символ \*. Рекомендуется ознакомиться с документацией синтаксиса markdown для использования всех доступных возможностей форматирования текста.

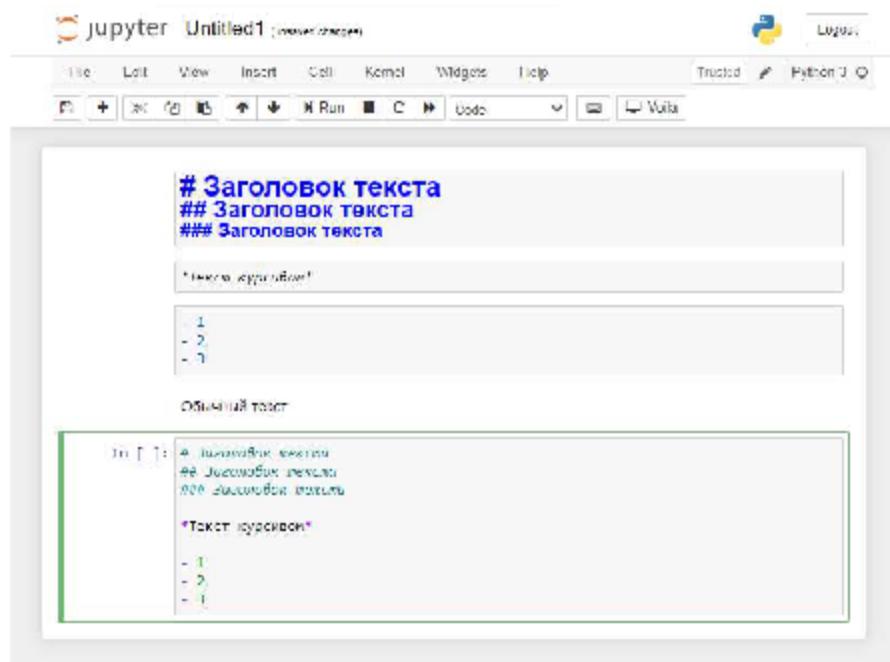


Рис.6.4. Использование markdown для разметки текста в Jupyter Notebook

Далее рассмотрим пример, который мы уже подробно рассматривали ранее. В программу будет загружен набор данных цветков iris.csv.

**Загрузка и анализ данных (iris dataset)**

В программу загружается массив данных, который содержит информацию (атрибуты) о наборе параметров различных видов цветка ирис.

```
In [1]: import matplotlib.pyplot as plt
import pandas as pd
iris = pd.read_csv('https://yiteha.ru/data_science/iris.csv')

#Помощь информации о типах данных методом info()
```

```
In [2]: print(iris.info())


RangeIndex: 150 entries, 0 to 149
Data columns (total: 5 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   id          150 non-null   int64  
 1   sepal_len   150 non-null   float64 
 2   sepal_wd   150 non-null   float64 
 3   petal_len   150 non-null   float64 
 4   petal_wd   150 non-null   float64 
 5   species     150 non-null   object  
dtypes: int64(1), float64(4), object(1)
memory usage: 6.5+ KB
None
```

Вывод 5 первых строк методом head()

```
In [3]: print(iris.head(5))

   id  sepal_len  sepal_wd  petal_len  petal_wd  species
0   0       5.1       3.5      1.4       0.2  iris-setosa
1   1       4.9       3.0      1.4       0.2  iris-setosa
2   2       4.7       3.2      1.3       0.2  iris-setosa
3   3       4.6       3.1      1.5       0.2  iris-setosa
4   4       5.0       3.6      1.4       0.2  iris-setosa
```

Рис.6.4. Пример анализа данных в Jupyter Notebook

В отличии от интерпретатора, мы получаем страницу, на которой последовательно выводится информация. Подобный подход очень удобен при составлении и подготовки исследовательского анализа. Нет необходимости копировать код, текст, графики и др.

На сервере Jupyter существует меню, которое позволяет управлять документом. Можно создать новый проект или экспортить существующий в один из предлагаемых форматов. В таблице представлены основные меню пункта.

Меню	Назначение
File	Создание, копирование, переименование и сохранение notebook в файл
Download	Возможность скачать notebook в разных форматах, включая pdf, html и slides для презентаций
Edit	Вырезать, копировать и вставка кода
View	Способ отображения номеров строк и панель инструментов
Insert	Добавление новых ячеек (до/после)
Cell	Работа с ячейками (изменение порядка и типа)
Help	Документация, в том числе к библиотекам Python: numpy, scipy, matplotlib и pandas

Следует отметить, что Jupyter Notebook – это мощный инструмент для отображения результатов исследовательского анализа. Аналитик после анализа данных должен предоставить отчёт, который может быть прочитан

разными специалистами. Данный подход позволяет реализовать некий стандарт представления результатов анализа.

## 7.8. Редактор кода Visual Studio Code

Далее рассмотрим ещё один популярный инструмент для редактирования исходного кода, который очень часто используется при создании скриптов на языке Python специалистами в области Data Science. В данном разделе мы ознакомимся с возможностями программы **Visual Studio Code**, позволяющей выполнять различные скрипты, редактировать и запускать блокноты jupyter, взаимодействовать с базами данных, управлять системами контроля версий git и многое другое. Программа выполнена в виде редактора, вся мощь которого проявляется после возможности подключения дополнительного модуля (расширения). Ознакомиться с документацией, скачать приложение можно на сайте <https://code.visualstudio.com>. Установите скачанное приложение себе на компьютер и установите его. Если редактор установлен правильно, то при первом запуске вы увидите что-то подобное рис.7.1.

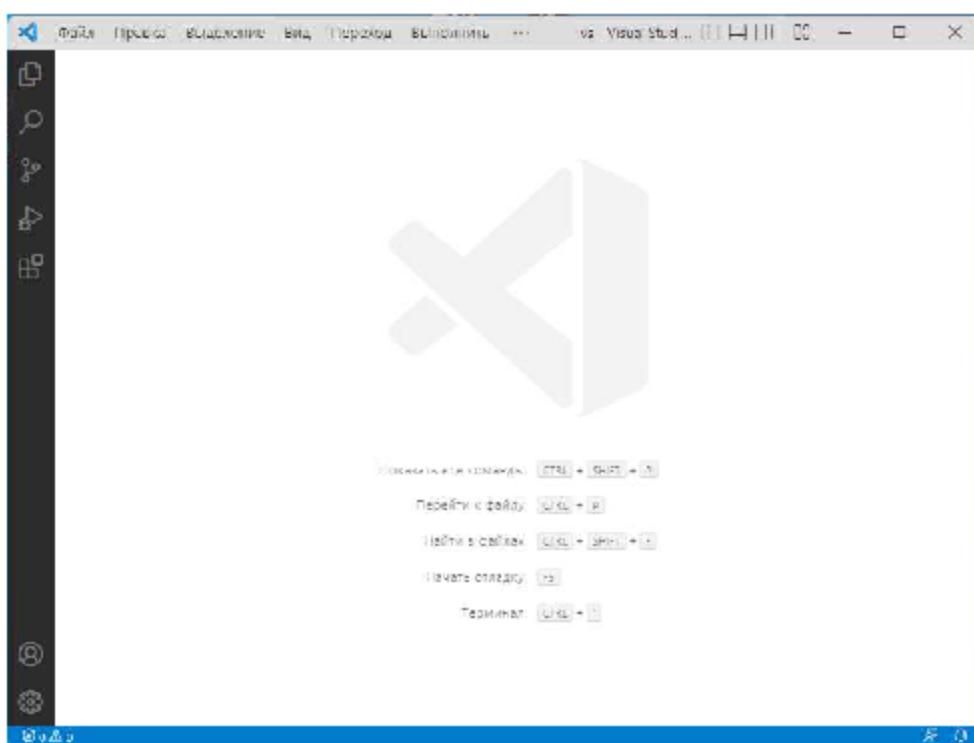


Рис.7.1. Окно программы Visual Studio Code

Обратите внимание на набор быстрых команд, которые предлагает программа. Далее их можно будет использовать для быстрого взаимодействия с программой. На рисунке представлен вид программы со светлой темой. Вы можете выбрать цветовое оформление на своё усмотрение. Для переключения установленных тем используются горячие клавиши Ctrl+K+T (нажать Ctrl, потом не отпуская, «K» и «T»). Дальше курсором выбираем любой вариант.

Программа имеет понятный и доступный интерфейс. Навигация в программе осуществляется через верхнее меню и левую боковую панель, в нижнем левом углу есть кнопка для дополнительных настроек программы.

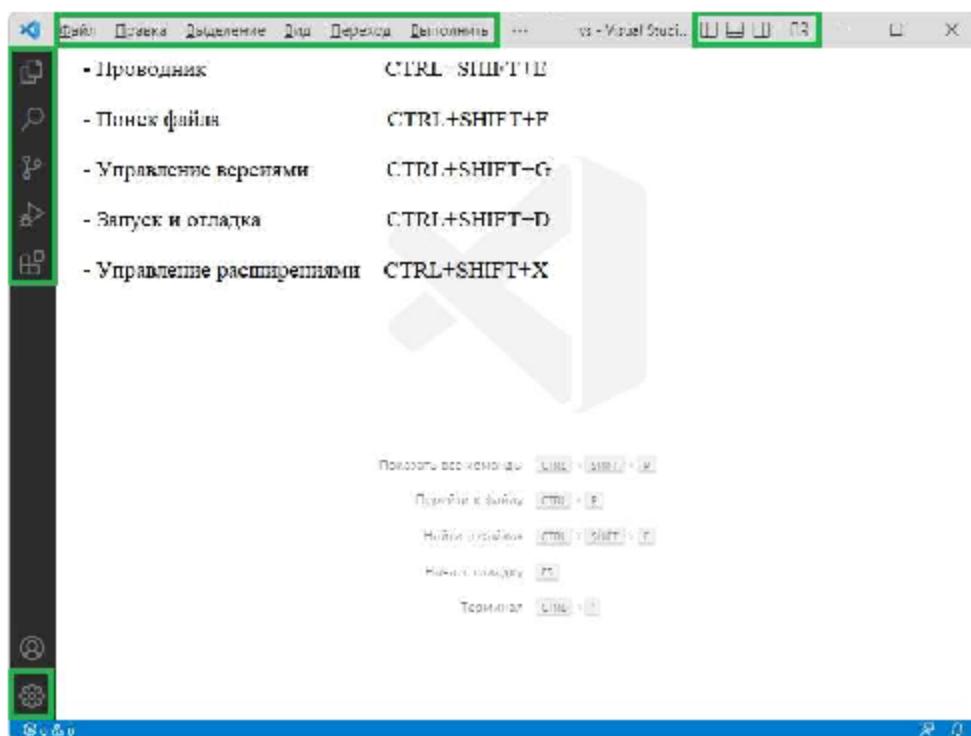


Рис.7.2. Навигация в программе Visual Studio Code

Особое внимание необходимо уделить боковой панели. На рис.7.2 показано описание кнопок боковой панели с сочетанием горячих клавиш. Проводник используется для просмотра содержимого каталога, создания файлов и папок. Управление версиями предназначено для быстрого доступа к системе контроля версий git (подробнее об этом мы поговорим в следующем разделе). В режиме запуска и отладки программы можно проверить работоспособность программы. В разделе управления расширениями можно выбрать и установить любой интересующий пакет. Для обновления последнего требуется активное подключение к сети интернет, т.к. база расширений постоянно обновляется и дополняется новыми модулями.

Для написания скриптов на языке Python и для работы с блокнотом юпитера нам потребуются специально установленные расширения. Перейдите в раздел управления расширениями и установить следующие пакеты: «Python extension for Visual Studio Code» и «Jupyter Extension for Visual Studio Code». Обязательно следите, чтобы дополнения были от Microsoft, т.к. возможны другие варианты подобных модулей. Для удобства дополнительно можно установить языковой пакет для русского языка. В будущем вы сможете доустановить другие полезные расширения.

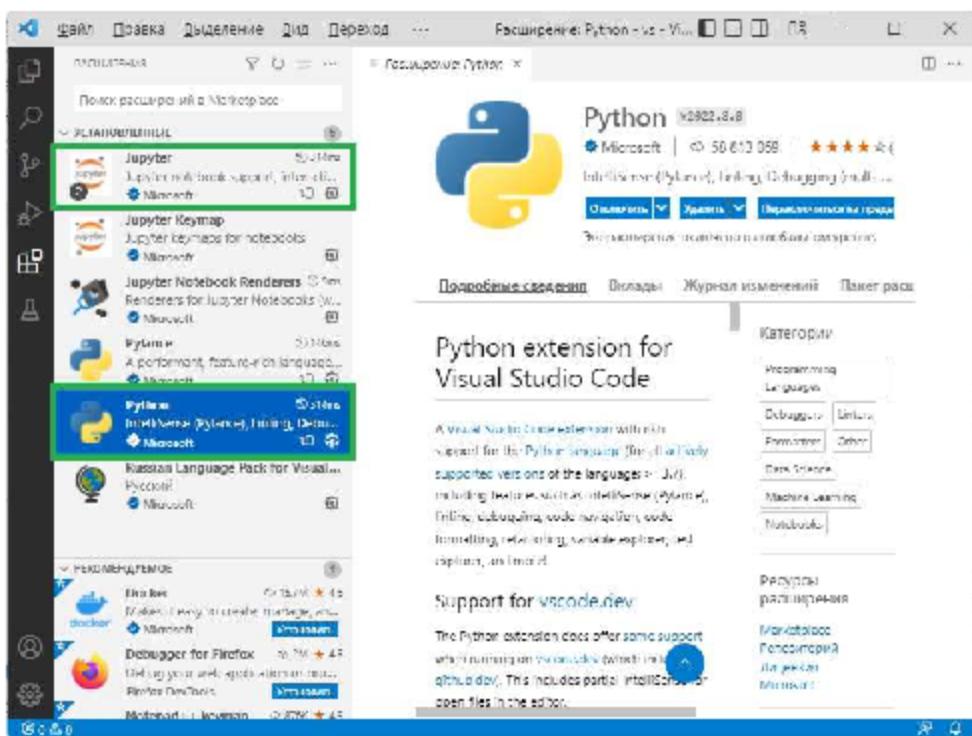


Рис.7.3. Установка расширений в программе Visual Studio Code

Далее рассмотрим процесс создания новой директории и её последующее открытие в программе. Постарайтесь при подготовке проекта всегда создавать новую директорию и работать именно в ней. Если же использовать только одну директорию и сохранять в неё все файлы, то рано или поздно может возникнуть конфликт файлов. Теперь о создании рабочей области проекта.

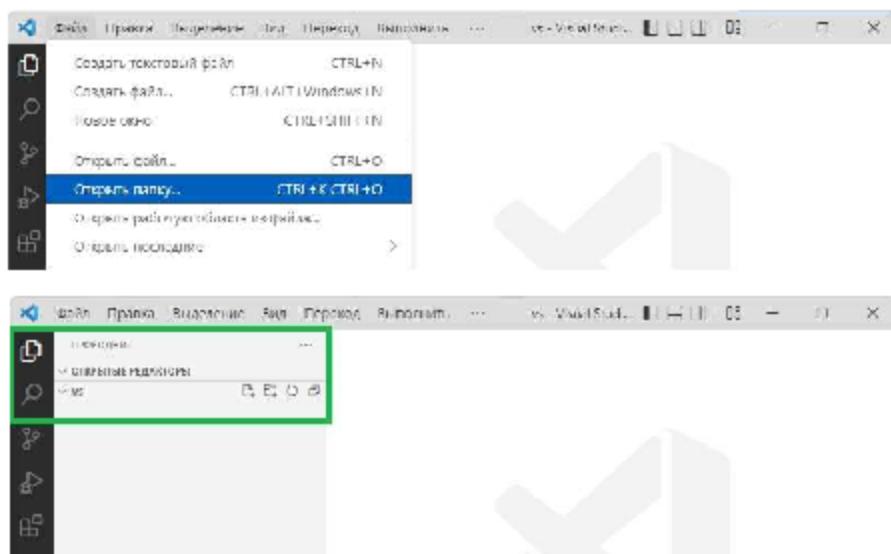


Рис.7.4. Открытие новой папки в программе Visual Studio Code

Под проектом мы будем понимать любую совокупность файлов, подготовленную в рамках выполнения конкретной работы. Давайте создадим, например, на локальном диске «С» в корне новую папку и назовём её vs.

Выберите в главном меню «Файл -> Открыть папку» и укажите ранее сохранённую папку vs (см. рис.7.4).

Если всё выполнено правильно, то в проводнике должна отобразится ранее созданная папка vs. Наша папка пуста. Добавим в неё новый файл с расширением \*.py. Для этого в окне проводника напротив названия папки vs нажмите «Создать файл» (при необходимости создайте папку «Новая папка»). Например, создайте и сохраните файл example.py (указание расширения файла обязательно). Добавим в программу небольшой код (на ваше усмотрение). В моём случае программа выводит текст 'Hello World!' в терминале программы.

```
print('Hello World!')
```

Сохраните программу нажав сочетание клавиш «Ctrl+S». Двойным нажатием на файл example.py откройте его редакторе. Перед запуском программы проверьте версию Python (при необходимости можно сменить интерпретатор). В нижнем правом углу нажмите на версию текущего интерпретатора (на рис. версия 3.9.2 64-bit). В программе будет показано версия используемого интерпретатора и версии, которые можно выбрать в качестве альтернативы.

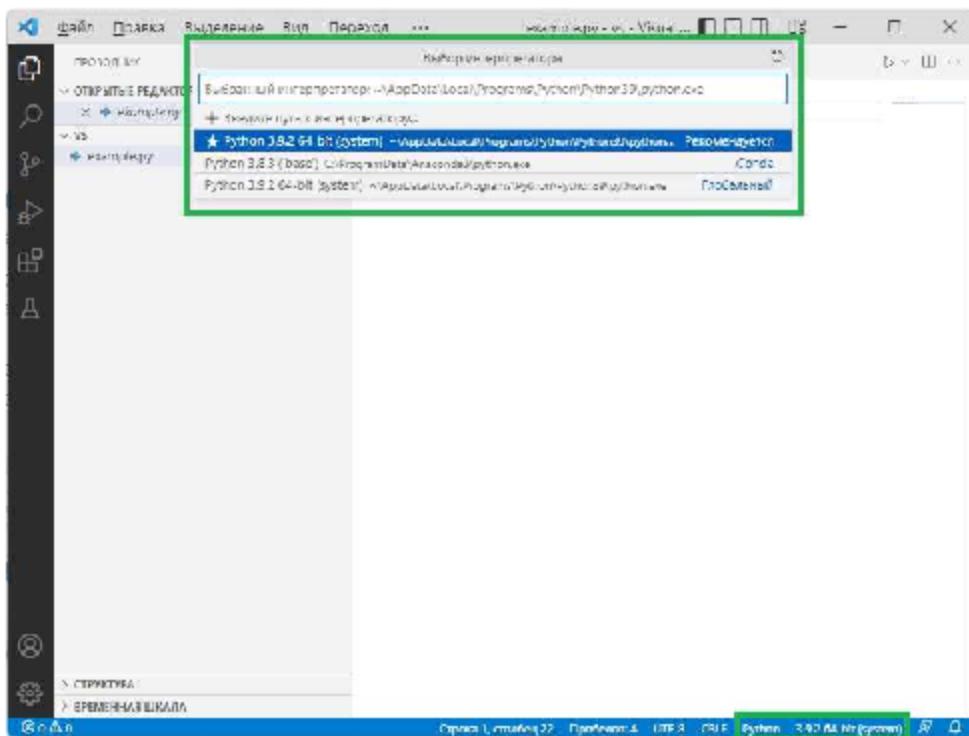


Рис.7.5. Выбор версии интерпретатора в программе VS Code

Если в системе до этого не было установлено никаких версий Python, а специальное расширение не подключено, то выполнить скрипт не получится. Для стабильной работы программы необходимо регулярно обновлять версию

интерпретатора. На рис. 7.6 показан результат выполнения написанной ранее программы.

The screenshot shows the VS Code interface with a Python file named 'ex1.py' open. The code contains a single line: `print('Hello World!')`. To the right of the editor is a terminal window titled 'Terminal' which displays the output of the script: 'Hello World!' followed by the path to the Python executable and its version information. The terminal also shows the command 'python' being typed and the file 'ex1.py' being specified.

Рис.7.6. Результат выполнения скрипта python в программе VS Code

В программе VS Code при запуске скрипта («Ctrl+F5» – без отладки, «F5» – с отладкой) будет автоматически открыто окно терминала, в котором будет показан результат. При необходимости терминал можно вызвать через главное меню «Терминал -> Создать терминал». Через терминал можно выполнить много разных процедур, используя специальные команды. В таблице ниже представлены некоторые из этих команд на примере установщика пакетов pip.

Меню	Назначение
<code>python3 --version</code>	Версия python 3
<code>pip install [package-name]</code>	Устанавливает последнюю версию пакета
<code>pip install [package-name] – upgrade</code>	Обновляет версию пакета
<code>pip freeze &gt; requirements.txt</code>	Подготовка файла зависимостей
<code>pip install [package-name]==4.8.2</code>	Обновляет версию пакета конкретной версии
<code>pip download [package-name]</code>	Скачивает пакеты
<code>pip uninstall [package-name]</code>	Удаляет пакеты
<code>pip freeze</code>	Выводит список установленных пакетов в необходимом формате (обычно используется для записи в requirements.txt)
<code>pip list</code>	Выводит список установленных пакетов
<code>pip help</code>	Помощь по командам

На практике очень часто вы будете пользоваться именно командой для установки пакетов и библиотек «`pip install`». Например, при попытке использовать в скрипте библиотеку `numpy`, которая не установлена в системе, в терминале получим следующее сообщение:

---

```
# ModuleNotFoundError: No module named 'numpy'
```

---

Для использования модуля необходимо его установить через терминал, используя следующую команду:

---

```
pip3 install numpy
```

---

Проверка всех установленных пакетов в системе может быть выполнена командой «`pip list`». Подробнее о всех доступных командах терминала можно узнать из справочника для языка `python`. Для фиксации версий пакетов можно использовать команду «`pip freeze`». Иногда будут ситуации, при которых потребуется запуск скриптов с определённым версиями библиотек. Для этого необходимо знать версии модулей. Сохранение версий пакетов обычно выполняют в файле `requirements.txt`. Значение нужных версий позволит вам без особых проблем и ошибок выполнить запуск приложения на другом компьютере.

Напоследок поговорим о возможности создания виртуальной среды через терминал `VS Code`. Если кратко, то виртуальная среда необходима для локализации вашего будущего проекта и приложения. Допустим, что мы разработали проект, который использует определённые версии пакетов, которые уже немного устарели. Мы не хотим обновлять версии библиотек, т.к. это может полностью нарушить работоспособность нашего приложения. В данной ситуации нам поможет создание отдельной виртуальной среды. В созданной среде будут установлены только те версии пакетов, которые нам интересны. Глобальное обновление библиотек не затронет зависимостей нашего нового виртуального окружения. Кроме того, мы можем сохранить виртуальную среду, при желании перенести на другую машину и развернуть там необходимое окружение (активировать виртуальную среду).

Рассмотрим команды для создания и активации/деактивации виртуальной среды:

---

```
# Создание виртуального окружения, venv - название вк  
python -m venv venv
```

```
# Активация рабочей среды  
venv\scripts\activate # для Windows  
source venv/bin/activate # для Linux
```

```
# Деактивация рабочей среды
venv\scripts\deactivate

# https://medium.com/@ph1174/python-venv-на-windows-10-2118ad685b1
```

Теперь практика создания виртуальной среды. Создадим на локальном диске «С» в папке vs новую папку, которую назовём её venv. Выберите в главном меню «Файл -> Открыть папку» и укажите ранее сохранённую папку venv. Далее необходимо вызвать терминал и через команду создать новую виртуальную среду new\_venv. После этого активируйте созданное окружение. Если всё выполнено верно, то результат должен быть аналогичный результату на рис.7.7.

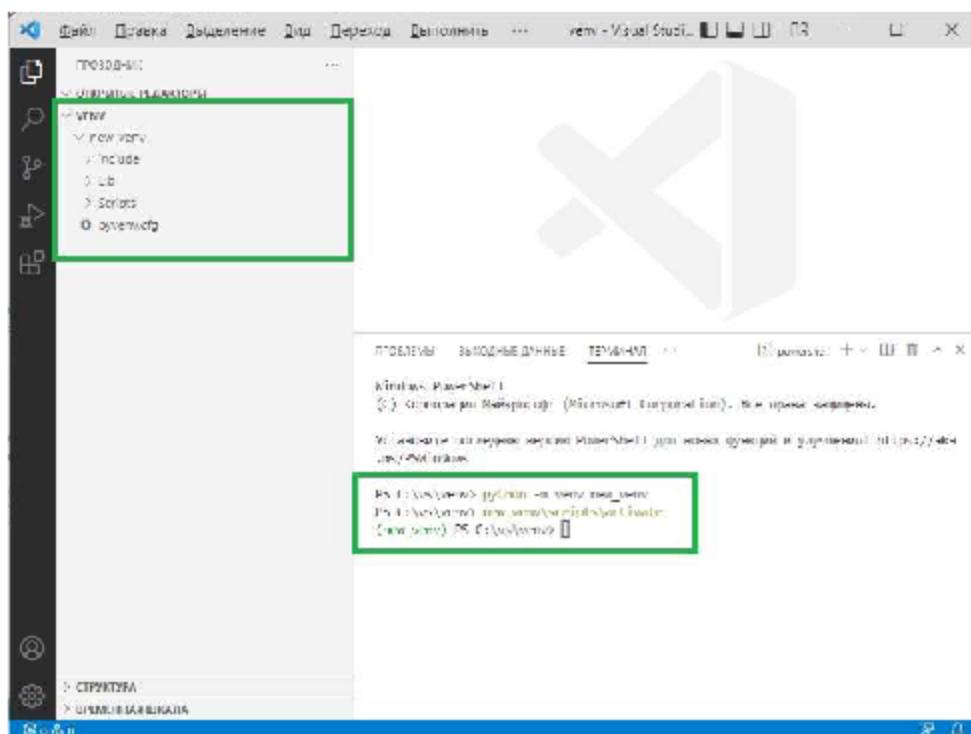


Рис.7.7. Результат выполнения скрипта python в программе VS Code

В проводнике мы увидим ранее созданную директорию и набор дополнительных файлов, которые были созданы после добавления новой виртуальной среды. В терминале можно наблюдать этапы создания и активации среды. После того, как среда была активирована, слева появилась дополнительная запись (`new_venv`), что свидетельствует о готовности виртуальной среды к работе. Теперь немного экспериментов. Если в терминале выполнить команду `pip list`, то мы увидим, что у нас практически нет установленных библиотек. Попытка запустить скрипты, в которых подключаются разные пакеты, мы получим ошибку. Т.е. мы создали локальную директорию, в которой существует отдельная версия интерпретатора со своим набором библиотек. В следующем разделе мы

подробно обсудим принцип работы с системами контроля версий git, используя программу VS Code.

## 7.9. Система контроля версий git

В этом разделе мы подробно поговорим о том, как начать работу с системой контроля версий git. Для начала рассмотрим основные концепции системы git, установим приложение на персональный компьютер, настроим систему и выполним несколько практических заданий, используя программу VS Code и веб-сервис для хостинга it-проектов github.

Система контроля версий (СКВ) – это система, которая выполняет сохранение изменений в файлах в процессе работы с ними. Основным достоинством подобных систем является возможность отката на предыдущую версию файла, что удобно в случае ошибок или возникновения непредвиденных ситуаций. Мы можем отследить не только изменения в файлах, но и отследить автора, которые внёс изменения. По своей сути подобные системы предоставляют возможность командной работы над каким-либо проектом. Для современных СКВ существует возможность использования разветвлённых сетей с применением технологии интернет. Появляется возможность совместной проектной работы специалистов, которые могут находиться в любой стране мира. Существует несколько вариантов реализации СКВ: локальные системы контроля версий, централизованные системы контроля версий, распределённые системы контроля версий. Локальные системы предусматривают работа с файлами на локальном компьютере. Система обладает простотой, но имеет ряд недостатков, которые связаны с появлением ошибок. Централизованные системы более продвинутые, они позволяют использовать локальные ресурсы и специальный сервер данных. На компьютерах выполняется сохранение именно файлов, а на сервере их изменения. Главный недостаток подобных систем связан с наличием единственного сервера для хранения изменений. Если сервер выйдет из строя, то будет нарушена работа всей системы. Наиболее продвинутым вариантом можно назвать распределённые системы (Git, Mercurial, Bazaar или Darcs). Можно сказать, что эти системы сочетают все основные достоинства ранее рассмотренных. В распределённых системах используются локальные компьютеры и сервер, только в этом случае локальные системы будут сохранять не только файлы, но и изменения этих файлов. Происходит полное копирование репозитория (содержимое удалённого сервера). Если случится так, что на сервере произойдёт ошибка и данные будут потеряны, то их можно восстановить, используя локальные версии изменений. Особенностью системы Git является то, что изменения сохраняются не в виде списка, а виде неких снимков изменений.

Для понимания работы системы контроля версиями необходимо разобрать основные концепции хранения изменений. Система определяет для хранимой информации набор файлов, в которых хранятся изменения (см. рис. 7.8).

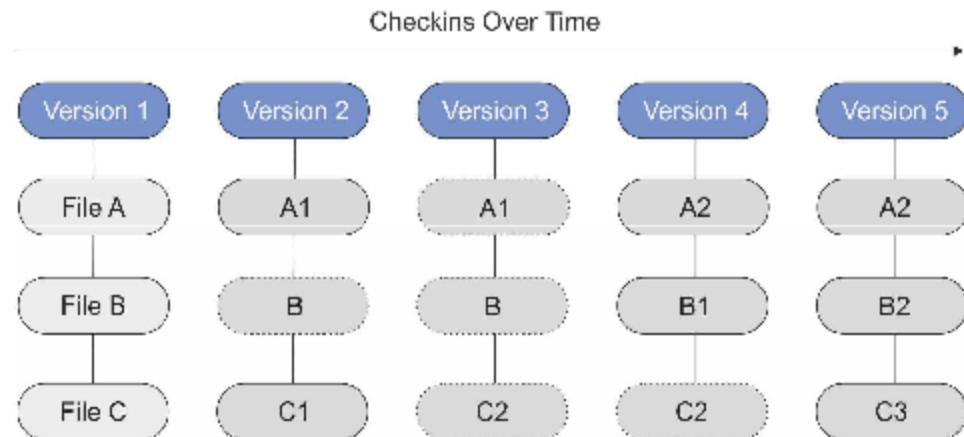


Рис.7.8. Структура хранения изменений файлов системой Git

Хранение изменений системой Git можно сравнить с набором снимков миниатюрной файловой системы. При выполнении коммита (сохранение изменений файла в системе Git), производится снимок состояния файла в данный момент времени, с последующим сохранением ссылки на данный снимок. Очень важно запомнить, что система не хранит изменённые файлы, а фиксирует поток снимков. Все операции выполняются локально. Даже если отсутствует связь с сервером, вы всё равно сможете работать в системе. Все обновления можно будет выполнить после активного подключения. Далее рассмотрим базовые понятия системы контроля версии.

В системе Git существует три основных состояния, в которых могут находиться файлы: изменённый (modified), индексированный (staged) и зафиксированный (committed).

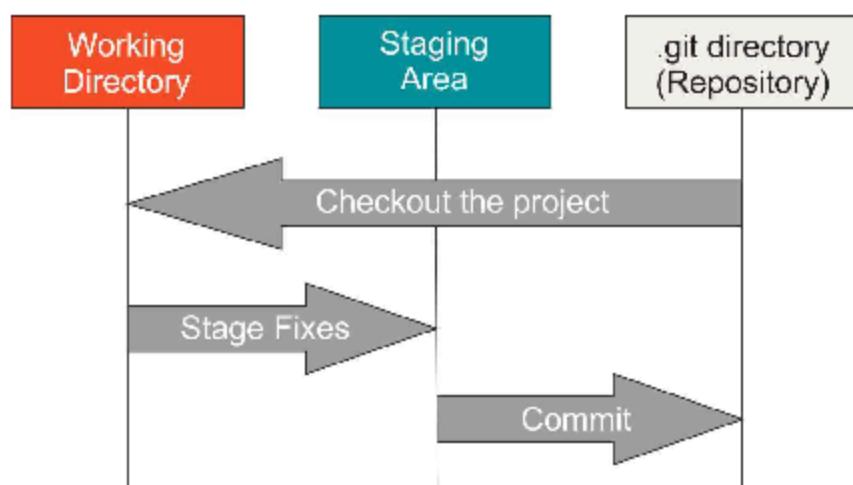


Рис.7.9. Структура проекта системы Git

Для работы системы требуется наличие трёх секций проекта Git (см. рис. 7.9): рабочая директория (working tree), область индексирования (staging area)

и директория git (Git directory). В рабочей области хранится информация о текущих изменениях файла. В области индексирования происходит определение тех изменений, которые будут отправлены в коммит, т.е. зафиксированы. Ещё эту область называют «область индексирования». Директория Git хранит базу объектов вашего проекта. Данная часть будет скопирована при клонировании репозитория, например, с другого компьютера или сервера.

Для удобства работы с системой Git можно использовать командную строку или специальный графический клиент. Если вы хотите понять все особенности функционирования системы, то рекомендуется использовать именно возможность работы через командную строку. Обязательно изучите вопрос запуска терминала в Mac OS, командной строкой или PowerShell в Windows, т.к. все приведённые ниже примеры будут разобраны с использованием предлагаемого подхода. Сразу отметим, что изучение команд системы Git требует дополнительного времени и изучения специальных ресурсов. Для углубленного изучения вопроса рекомендуется самостоятельно ознакомиться с материалом на данную тему. Например, можно использовать следующий материал:

<https://git-scm.com/book/ru/v2/>

В нашем случае будет рассмотрен пример работы с системой Git в операционной системе Windows. Для простоты мы разделим работу на несколько простых этапов. Так будет проще понять и запомнить особенности всей системы. Определим следующие этапы: установка системы контроля версий git на персональный компьютер, создание аккаунта на сайте it-проектов github, подключение к удалённому репозиторию с помощью программы VS Code.

## 1. Установка и настройка системы контроля версий git.

Программу Git можно скачать с официального сайта разработчика. Дополнительно можно скачать и установить GUI-клиент, позволяющий взаимодействовать с системой через графический интерфейс. Если вы используете операционную систему Windows, то можно воспользоваться следующей ссылкой для скачивания приложения <https://git-scm.com/download/win>. Скачайте и установите приложение на свой компьютер. Для запуска команд используйте командную строку.

Далее необходимо выполнить настройку и инициализации программы. Установленные настройки можно узнать с помощью следующей команды:

---

```
$ git config --list --show-origin
```

---

После установки программы необходимо определить пользователя программы – указать имя пользователя и адрес электронной почты. Данная информация будет использована при отправке коммита. При желании для работы с системой можно использовать текстовые редакторы: Vim, Emacs и Notepad++. В официальной документации можно подробнее об этом прочитать. Первое, что нам нужно будет сделать, так это указать имя пользователя и его адрес электронной почты.

---

```
# Имя пользователя  
$ git config --global user.name "John Doe"  
  
# Адрес электронной почты пользователя  
$ git config --global user.email johndoe@example.com
```

---

Ещё одной особенностью системы Git является возможность использования различных веток изменений. Дело в том, что система позволяет создавать дополнительные ветки, в которых можно сохранять изменения файлов. По сути, каждая ветка может быть закреплена за отдельным пользователем. Например, вы работаете над крупным проектом, в котором выполняете роль администратора. Параллельно с вами выполняют разработку несколько специалистов. В этом случае вы контролируете ветку master (по умолчанию), а остальными ветками управляют ваши коллеги. За администратором остаётся право выполнить откат версий или выполнить слияние веток, если изменения, подготовленные другими специалистами, могут быть применены. Для инициализации git на компьютере можно выполнить следующую команду:

```
# Для инициализации ветки master по умолчанию  
$ git init  
  
# Для инициализации ветки main по умолчанию  
$ git config --global init.defaultBranch main
```

---

Проверку выполненных настроек можно выполнить, используя команду:

```
# Проверка установок  
$ git config -l
```

---

Теперь остановимся на том, как можно создать локальный репозиторий. По репозиторием мы будем понимать место для хранения рабочих файлов проекта. Существуют два вида репозиториев: локальный и глобальный (репозиторий на github). Более подробно глобальный репозиторий будет

рассмотрен позже. Для инициализации локального репозитория необходимо перейти в созданный для этого каталог и выполнить команду «git init»:

---

```
# Для Linux:  
$ cd /home/user/my_project  
  
# Для macOS:  
$ cd /Users/user/my_project  
  
# Для Windows:  
$ cd C:/Users/user/my_project  
  
$ git init  
  
# Последующий коммит файлов  
$ git add *.c  
$ git add LICENSE  
$ git commit -m 'Initial project version'
```

---

В системе git предусмотрена возможность клонирования репозитория с удалённого сервера. Это полезно в том случае, если вы присоединяетесь к команде разработчиков и хотите иметь на своём компьютере последнюю версию файлов проекта. Для этого можно использовать команду git clone.

---

```
# Копирование удалённого репозитория  
# url в формате https://  
$ git clone <url>
```

---

Далее мы подробно поговорим о создании удалённого репозитория, настройке и его копировании на локальный компьютер.

## **2. Создание аккаунта и удалённого репозитория на сайте [github.com](https://github.com).**

Для создания удалённого репозитория будем использовать сайт <https://github.com>. Перейдите по ссылке на указанный сайт и зарегистрируйте нового пользователя. В дальнейшем при работе с Git нам потребуется имя пользователя и адрес электронной почты. Данные для подключения к системе на локальном компьютере и на сайте должны совпадать. Если получится так, что данные для подключения не совпадают, то у вас не получится синхронизировать репозитории (локальный и удалённый), а в этом заключается главное достоинство рассматриваемой системы.

Теперь создадим свой первый репозиторий. Адрес вашего аккаунта будет выглядеть следующим образом (nick\_name\_user – имя пользователя при регистрации):

[https://github.com/\[nick\\_name\\_user\]](https://github.com/[nick_name_user])

Для создания нового репозитория перейдите в раздел «Repositories» и создайте новый репозиторий нажав кнопку «New», вы сразу перейдёте по ссылке на страницу «Create a new repository».

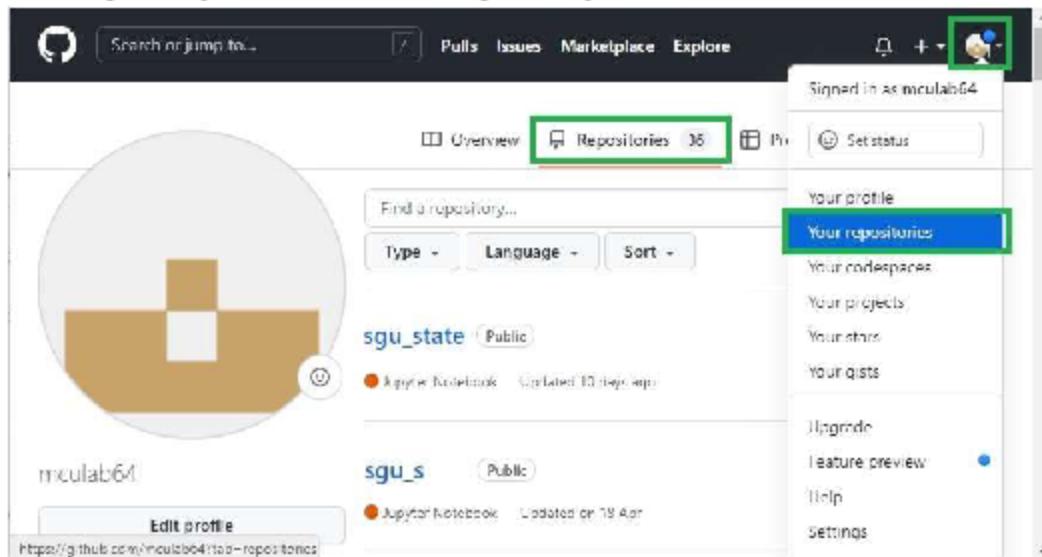


Рис.7.10. Создание нового репозитория в системе github

Далее необходимо указать название вашего будущего репозитория «Repository name», указать режим доступа к репозиторию как «Public» и отметить галочку для создания файла README. В режиме public ваш репозиторий будет доступен для просмотра и копирования всем пользователям сети интернет. Если в этом нет необходимости, то можете перевести режим доступа в «Private». Файл readme необходим для предварительного знакомства с репозиторием. В нём можно описать содержимое репозитория, его назначение и другую полезную информацию.

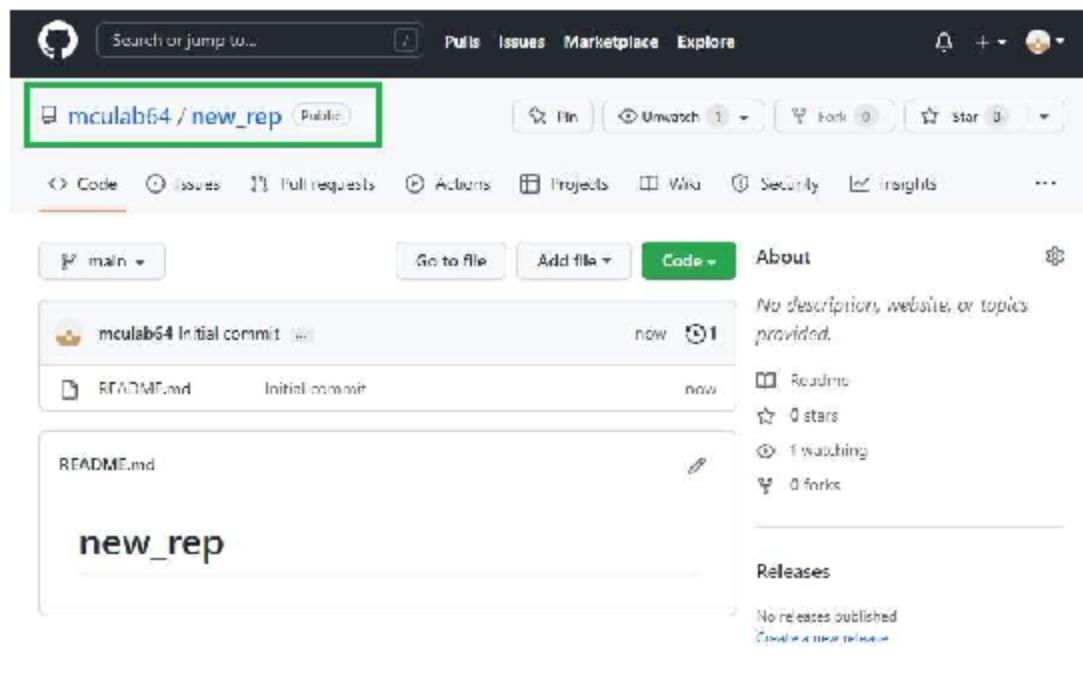
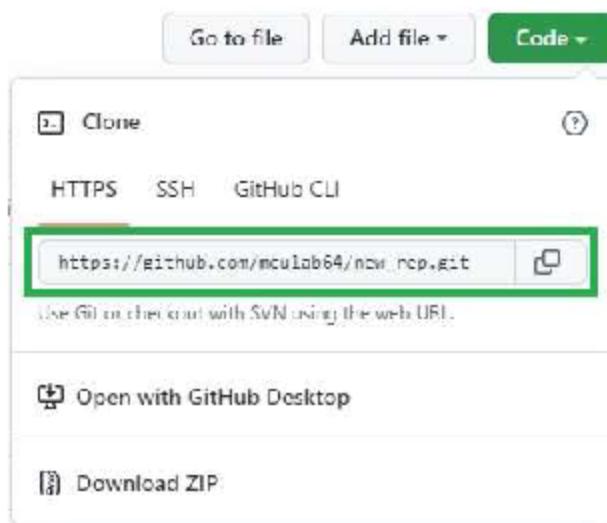


Рис.7.11. Страница репозитория на сайте [github.com](https://github.com)

Если всё выполнено правильно, то далее вы будете перенаправлены в созданный репозиторий (см. рис. 7.11). Название нового репозитория new\_gerp.

Для загрузки новых файлов можно использовать кнопку «Add file», но мы будем использовать другой подход наполнения репозитория. Сначала мы полностью клонируем созданный репозиторий на компьютер. В локальном репозитории мы будем сохранять все нужные нам файлы для проекта. В финале мы выполним отправку коммита с локального на удалённый репозиторий. В результате мы получим полную реализацию системы контроля версий git.



Нажмите на кнопку «Code», вы увидите информацию о ссылке на репозиторий. Ссылка дана в нескольких вариантах. Нас интересует ссылка с использованием протокола https. Выполните копирование адреса репозитория, он пригодится нам на следующем этапе.

### 3. Подключение к репозиторию с помощью программы VS Code.

Перед началом работы мы разберём ещё несколько ключевых моментов, которые важны для понимания функционирования системы git. На рис. 7.12. представлена упрощённая блок-схема передачи информации в системе.

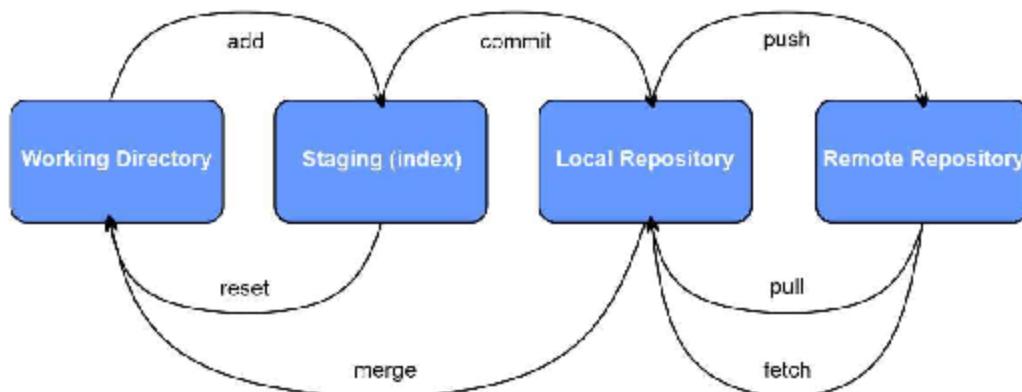


Рис. 7.12. Блок-схема обмена изменениями и файлами в системе git

Основные команды, которые нам необходимо изучить: add, commit и push/pull. Использование приведённых команд позволит реализовать минимум для работы в системе git. Последовательность команд выглядит следующим образом:

---

```
# Инициализация репозитория на компьютере
$ git init

# Проверка состояния системы (на любом этапе)
$ git status

# Добавление индексируемых файлов
$ git add *.c

# Подготовка коммита и сообщения для отправки
$ git commit -m 'Initial project version'

# Отправка изменений и содержимого НА удалённый репозиторий
$ git push

# Извлечение и загрузка содержимого ИЗ удалённого репозитория
$ git pull
```

---

Для начала необходимо выполнить инициализацию репозитория на локальном компьютере командой «git init». Далее необходимо добавить перечень файлов для индексирования командой «git add». Команда «git commit» позволяет записать индексированные изменения в репозиторий. Команда «git push» отправляет изменения в содержимом на удалённой репозиторий, обратная команда «git pull» извлекает содержимое из удалённого репозитория. Команда «git status» отображает состояние рабочего каталога и раздела проиндексированных файлов. Теперь приступим к практике в VS Code. Клонируем удалённый репозиторий, инициализируем его на локальном компьютере, создадим несколько файлов на компьютере, проиндексируем их в системе git, подготовим коммит и отправим содержимое на удалённой репозиторий.

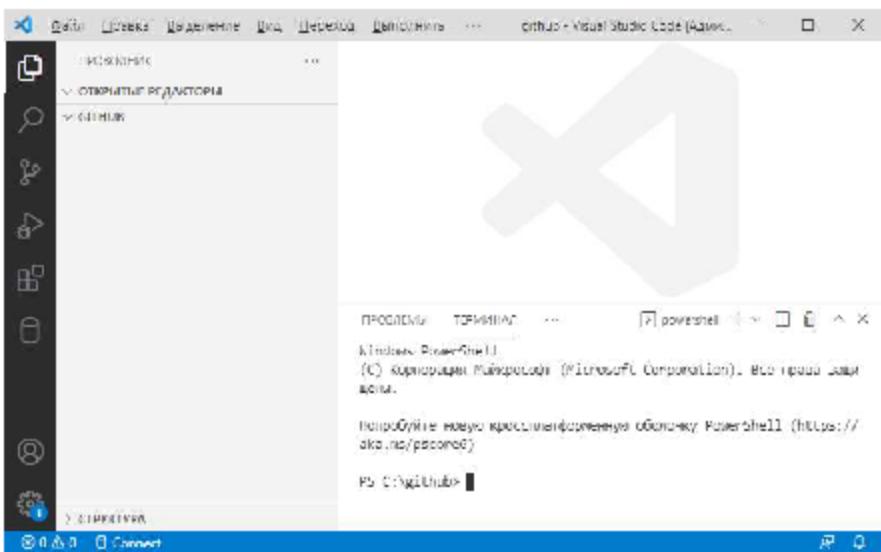


Рис.7.13. Окно VS Code при работе с системой Git

Создайте папку `github` на компьютере, запустите программу VS Code и откройте созданную папку в программе (см. рис.7.13). Клонируем удалённый репозиторий, для этого выполним следующие команды (клонирую ранее созданный мною репозиторий `new_rep`):

```
# Инициализируем репозиторий на локальном компьютере
$ git init

# Клонируем удалённый репозиторий
$ git clone https://github.com/mculab64/new_rep.git
```

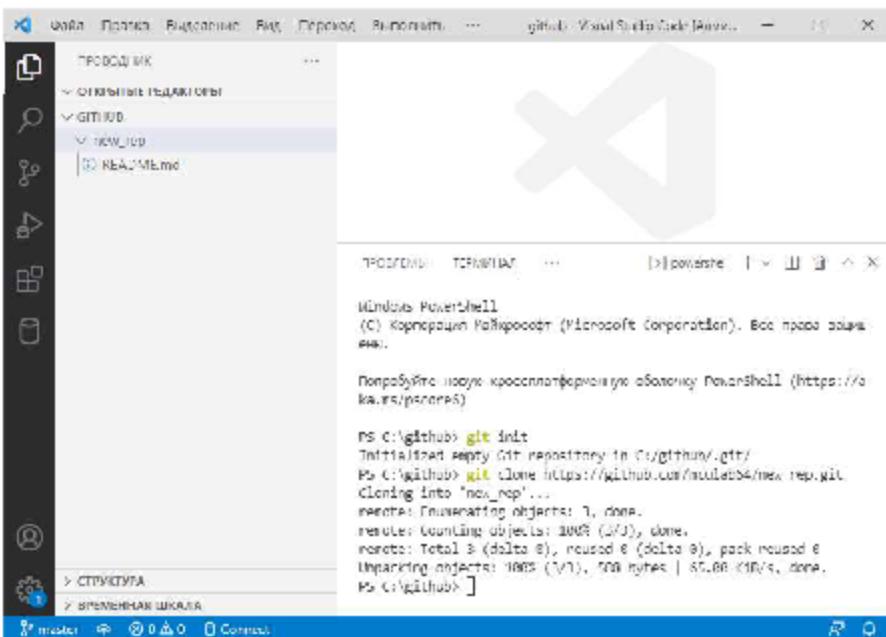


Рис.7.14. Клонирование репозитория в программе VS Code

После инициализации в терминале появилась информация о том, что был инициализирован пустой репозиторий. Следующая команда позволила

клонировать удалённый репозиторий new\_repos на компьютер (см. рис.7.14). В проводнике можно видеть файлы, которые были получены из удалённого репозитория. Сейчас наша папка new\_repos пуста. Добавим в неё новый файл с расширением. Для этого в окне проводника напротив названия папки github нажмите «Создать файл» (при необходимости создайте папку «Новая папка»). Например, создайте и сохраните файл example.py (указание расширения файла обязательно). Добавим в программу небольшой код. Сохраните программу нажав сочетание клавиш «Ctrl+S». Переместите созданный файл в папку new\_repos. Теперь можно сохранить изменения в системе и подготовить коммит для отправки изменений на удалённый репозиторий.

Перед отправкой изменений необходимо открыть заново папку new\_repos в программе VS Code. Далее проинициализируйте этот репозиторий командой «git init». Теперь строго по порядку. Во-первых, добавьте к индексированию файлы, используя команду «git add». Во-вторых, подготовьте коммит, используя команду «git commit». Обязательно укажите комментарий для коммита. Комментарий позволит отследить изменения, которые были выполнены вами. Для отправки изменений используем команду «git push».

---

```
# Индексирование файлов, атрибут -A позволит индексировать все файлы
$ git add -A

# Подготовка коммита и сообщения для отправки
$ git commit -m 'Initial project version'

# Отправка изменений и содержимого НА удалённый репозиторий
$ git push
```

---

На рис.7.15 представлен результат отправки изменений на удалённый репозиторий. Если в терминале появились сообщения об ошибках или предупреждения, то постарайтесь проверить всю последовательно действий заново. В случае ошибок отправка изменений не будет выполнена.

The screenshot shows the Visual Studio Code interface. On the left is the file explorer with a folder named 'new\_rep' containing files 'example.py', 'README.md', and '.gitignore'. The main area is a code editor with the file 'example.py' open, containing the following Python code:

```

example.py
1 import numpy as np
2
3 np_array = np.array([1,2,3,4,5])
4
5 print(np_array)

```

Below the code editor is a terminal window showing the command-line output of a git commit:

```

nothing added to commit but untracked files present (use "git add"
to track)
$ C:\github\new_rep> git add -A
$ C:\github\new_rep> git commit -m 'Initial project version'
[main f4e673c] Initial project version
 1 file changed, 5 insertions(+)
 create mode 100644 example.py
$ C:\github\new_rep> git push
Everything up-to-date
Counting objects: 4, done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
To https://github.com/mkulab64/new_rep.git
 ! [new branch] main -> main
$ C:\github\new_rep>

```

Рис.7.15. Клонирование репозитория в программе VS Code

Теперь нам только осталось проверить результат нашего коммита. Для этого необходимо вернуться на сайт github и проверим, есть ли изменения в удалённом репозитории.

На рис. 7.16 видно, что в репозитории появился новый файл `example.py`, который по своему содержанию является копией того файла, который был создан и сохранён в локальном репозитории. Аналогично можно копировать и другие файлы (другое расширение). Главное правило, которое необходимо запомнить, всё-таки github был задуман ресурсом для программистов. В основном в репозитории сохраняют программы различных форматов. Нет необходимости сохранять в репозитории всего подряд. Страйтесь придерживаться культуры использования данного ресурса.

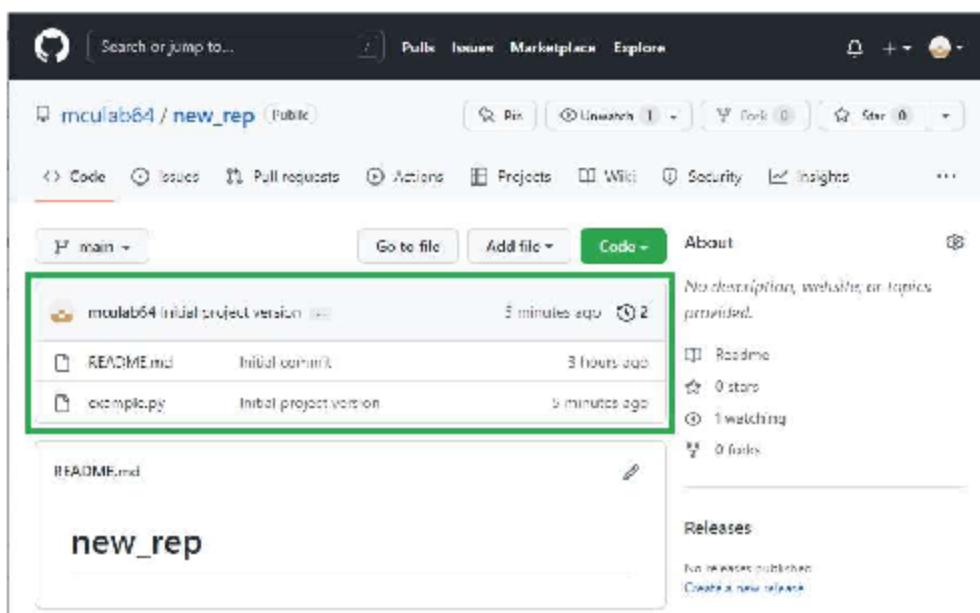


Рис.7.16. Результат коммита из локального в удалённый репозиторий

Возможен обратный процесс копирования изменений. Например, внесите изменения в ранее сохранённый файл на github изменения и сохраните их. Далее в программе VS Code выполните команду «git pull».

Отметим, что для полноценной двусторонней работы пользователь в системе git на компьютере и пользователь на сайте github должен совпадать. Если на компьютере выполняется первое подключение к удалённому репозиторию, то система может запросить авторизацию пользователя. Иногда для подтверждения прав на репозиторий система запрашивает специальный токен, который можно узнать на сайте [github.com](https://github.com) в профиле пользователя. К сожалению, в рамках данного материала не были рассмотрены многие важные функции системы Git. Например, не была рассмотрена система ветвлений. Рекомендуется самостоятельно изучить эти темы, используя специализированную литературу.



## Задания для самостоятельной работы

### Сбор и хранение данных

#### 1. Работа с файлами формата csv, excel и txt

Для выполнения задания необходимо использовать базу данных цветка ириса (база, созданная Р.А. Фишером). Напомним, что таблица содержит 150 растений ириса, каждое из которых имеет 4 числовых атрибута: длина чашелистика в см, ширина чашелистика в см, длина лепестка в см и ширина лепестка в см. Готовый датасет можно скачать по адресу:

---

'[https://github.com/mculab64/data\\_science/tree/main/ds\\_datasets/iris.csv](https://github.com/mculab64/data_science/tree/main/ds_datasets/iris.csv)'

---

Используя библиотеку matplotlib.pyplot и pandas, выполните следующие задания.

**1.1.** Загрузите датасет в программу и выведите на экран первые 10 строк. Проанализируйте данные

---

```
import pandas as pd  
df = pd.read_csv('iris.csv')
```

---

Используя известные методы работы с таблицами в pandas, ответьте на следующие вопросы:

- 1) какое количество элементов содержит таблица?
- 2) какой тип данных содержится в таблице?
- 3) какие уникальные данные содержит столбец df['species']?
- 4) какое количество образцов вида iris-setosa, iris-versicolor и iris-virginica содержится в таблице?

**1.2.** Выполните преобразование категориального признака [‘species’]. Для этого используйте словарь, который в качестве ключа использует название вида цветка, а в качестве значения целое число от 0 до 2.

```
var_flower = {'iris-setosa': 0, 'iris-versicolor': 1, 'iris-virginica': 2}  
df['species'] = df['species'].map(var_flower)
```

Получится следующее:

	id	sepal_len	sepal_wd	petal_len	petal_wd	species
0	0	5.1	3.5	1.4	0.2	0
1	1	4.9	3.0	1.4	0.2	0
2	2	4.7	3.2	1.3	0.2	0
3	3	4.6	3.1	1.5	0.2	0
4	4	5.0	3.6	1.4	0.2	0
..	...	...	...	...	...	...
145	145	6.7	3.0	5.2	2.3	2
146	146	6.3	2.5	5.0	1.9	2
147	147	6.5	3.0	5.2	2.0	2
148	148	6.2	3.4	5.4	2.3	2
149	149	5.9	3.0	5.1	1.8	2

**1.3.** Разделите полученный датасет на 3 таблицы. Первая таблица должна содержать данные о цветках с признаком [‘species’] = 0, вторая таблица данные о цветках с признаком [‘species’] = 1, третья таблица данные о цветках с признаком [‘species’] = 2. Сохраните полученные таблицы в формате \*.csv, \*.xls и \*.txt.

```
# Сохранение в формате *.csv  
df_1.to_csv('iris-setosa.csv')  
  
# Сохранение в формате *.xls  
df_2.to_excel('iris-versicolor.xls')  
  
# Сохранение в формате *.txt, разделитель ','  
df_3.to_csv('iris-virginica.txt', sep = ',')
```

**1.4.** Загрузите датасет iris-setosa.csv в программу. Используя библиотеку matplotlib.pyplot, постройте два графика: [sepal\_len/sepal\_wd] и [petal\_len/petal\_wd]. Используйте одно графическое окно (nrows=1, ncols=2).

**1.5.** Загрузите датасет iris-versicolor.xls в программу. Постройте два графика: [sepal\_len/sepal\_wd] и [petal\_len/petal\_wd]. Используйте одно графическое окно (nrows=1, ncols=2).

**1.6.** Загрузите датасет `iris-virginica.txt` в программу. Постройте два графика: `[sepal_len/sepal_wd]` и `[petal_len/petal_wd]`. Используйте одно графическое окно (`nrows=1, ncols=2`).

## 2. Чтение и запись JSON-файлов

Для выполнения заданий используйте датасет, который содержит информацию о группе людей: рост, масса, возраст и гендерная принадлежность.

---

```
'https://github.com/mculab64/data\_science/tree/main/ds\_datasets/peoples.json'
```

---

Для загрузки файла в программу используйте библиотеку `json`.

```
import pandas as pd
import json

with open('peoples.json') as f:
    peoples = json.load(f)

print(peoples['heights'].values())
```

**2.1.** Напишите программу, которая рассчитывает среднее значение, минимальное значение, максимальное значение и медиану в столбце `['heights']`, `['weights']` и `['ages']`.

**2.2.** Напишите программу, которая рассчитывает дисперсию и стандартное отклонение в столбце `['heights']`, `['weights']` и `['ages']`.

**2.3.** Напишите программу, которая рассчитывает межквартильное расстояние (разность данных между квантилями 25% и 75%) в столбце `['heights']`, `['weights']` и `['ages']`.

**2.4.** Для столбца `['gender']` определите уникальные значения. Ответ выведите на экран в виде списка.

## 3. Чтение html-файлов из интернета, парсинг данных, API

Для выполнения следующего задания потребуется подключение к сети интернет. Убедитесь, что в системе установлены следующие библиотеки: **lxml**, **requests**, **re** и **beautifulsoup4**. Загрузите в браузере главную страницу сайта:

---

```
'https://elteha.ru/index.php'
```

---

Проанализируйте содержимое страницы перед выполнением заданий. Для просмотра структуры страницы можно использовать сочетание клавиш на клавиатуре Ctrl+U.

**3.1.** Найдите на главной странице количество употреблений слова «Подробнее». Используйте библиотеки **requests** и **re**. На экран выводится только одно число.

---

```
import requests
import re
URL = 'https://elteha.ru/index.php'

result = requests.get(URL).text
```

---

**3.2.** Найдите все заголовки на странице, которые в качестве атрибута элемента имеют class="main\_title\_b". Результат выведите на экран в виде списка.

---

```
# Теоретические основы электроники
# Программирование микроконтроллеров
...
# Моделирование схем
```

---

**3.3.** Найдите все заголовки разделов, которые в качестве атрибута элемента имеют class="top\_text\_1". Результат выведите на экран в виде списка.

---

```
# Основы электроники
# Транзисторы
...
# Программирование в LabView
```

---

**3.4.** Найдите все заголовки разделов, которые в качестве атрибута элемента имеют class="bottom\_text\_1". Часть текста содержит информацию о количестве статей в разделе. Ваша задача найти сумму всех статей на главной странице сайта. На экран выводится только одно число.

---

```
# Кол-во статей: 7
# Кол-во статей: 4
...
# Кол-во статей: 16
```

---

**3.5.** Для выполнения следующего задания потребуется зарегистрироваться на сайте, который предоставляет возможность использования API, позволяющего предоставить данные о погоде в конкретном регионе. Зарегистрируйтесь на сайте и сгенерируйте личный api-ключ.

---

'[https://home.openweathermap.org/api\\_keys](https://home.openweathermap.org/api_keys)

---

Выберите один из городов России, например, Самара. Для данного работы с API напишите программу, которая выводит на экран информацию о температуре, давлении и влажности в выбранном населённом пункте.

---

```
import requests
import json

BASE_URL = 'http://api.openweathermap.org/data/2.5/weather'

s_city = 'Samara, RU'
appid = 'your_api_key'

...
print('Погода в Самаре: температура t = {} градусов,
      давление P = {} ГПа,
      влажность H = {}%'.format(temp, pressure, humidity))
```

---

Обратите внимание, что давление хранится в непривычном нам формате. Доработайте программу так, чтобы давление выводилось в мм рт. ст.

**3.6.** Измените предыдущую программу так, чтобы информация выводилась на экран каждые 10 секунд. Для реализации временной задержки используйте библиотеку time.

---

```
import time

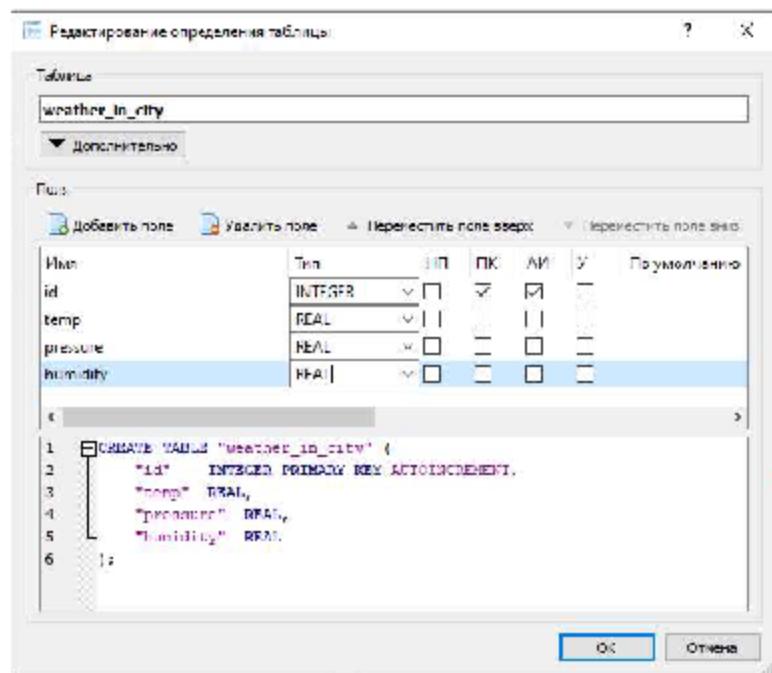
...
while True:
    ...
    ...
    time.sleep(10)
```

---

## 4. Работа с базой данных SQL в библиотеке sqlite3

Для выполнения следующих заданий потребуется установка на компьютере программы DB Browser for SQLite. Установите программу, запустите и создайте новую базу данных под названием weather.sqlite3. Работа с базами данных SQL на Python выполняется с использованием библиотеки sqlite3.

**4.1.** В программе DB Browser for SQLite создайте новую базу данных weather и таблицу weather\_in\_city. В созданную таблицу добавьте поля: id (integer), temp (real), pressure (real), humidity (real).



**4.2.** Напишите программу, которая, используя API, сохраняет информацию о температуре, давлении и влажности в выбранном населённом пункте в базу данных (база данных weather и таблица weather\_in\_city). Выберите один из городов России, например, Москва. Программа должна выполнять сохранение данных в базу каждые 30 секунд. Для реализации временной задержки используйте библиотеку time. Выполните сохранение не менее 10 измерений.

**4.3.** В программе на Python выполнить загрузку (создать DataFrame) данных из базы weather.sqlite3 (таблица weather\_in\_city). Методом head() выведите на экран 10 строк полученной таблицы.

**4.4.** Для полученных данных в таблице найдите следующие величины: минимальное, максимальное, среднее и стандартное отклонение величин в каждом столбце.

**4.5.** Используя библиотеку pandas, постройте три графика, которые показывают изменение температуры, давления и влажности во времени. Все графики должны иметь метки, заголовок, легенду и сетку. Тип графиков линейный. Шаг времени dt составляет 30 секунд.

## 5. Представление результатов в Jupyter Notebook – IPython

Для выполнения заданий потребуется установка **Jupyter Notebook** на компьютер. Условно, всё задание можно разбить на несколько логических блоков, которые позволяют упростить работу с данными. Мы будем обрабатывать набор данных, связанных со списком самых популярных американских комедий по версии сайта Кинопоиск за определённый период времени. Для каждого фильма известны следующие данные: название фильма, год выхода фильма, рейтинг по версии сайта Кинопоиск, длительность фильма

(мин.), бюджет фильма (млн. долларов США), кассовые сборы (млн. долларов США). Блокнот-шаблон с заданиями (`independent_work.ipynb`) можно скачать по адресу:

---

'[https://github.com/mculab64/data\\_science/tree/main/ds\\_datasets/](https://github.com/mculab64/data_science/tree/main/ds_datasets/)'

---

Цель практической работы: проанализировать полученный датасет в **Jupyter Notebook**, рассчитать некоторые статистические показатели, выявить закономерности и сделать выводы.

Примечание: выполнить задание можно в **Jupyter Notebook**, **Jupyter Lab** или `colab research.google.com`:

---

'<https://colab.research.google.com>'

---

Обратите внимание на дополнительный материал в конце блокнота (`breadcrumbs`).