

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«КРЫМСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ имени В.И.ВЕРНАДСКОГО»  
**ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ**  
Кафедра прикладной математики

**ДЮЛИЧЕВА ЮЛИЯ ЮРЬЕВНА,  
МАРШАЛОК МАРИЯ МИХАЙЛОВНА**

**ИНТЕЛЛЕКТУАЛЬНЫЙ АНАЛИЗ ДАННЫХ:  
NUMPY, PANDAS, MATPLOTLIB  
часть 1**

**Учебно-методическое пособие для практических занятий по дисциплине  
«Интеллектуальный анализ данных»**

для обучающихся направления подготовки:  
01.04.04 Прикладная математика  
очной формы обучения

**Симферополь 2024**

Дюличева Ю.Ю., Маршалок М.М. Интеллектуальный анализ данных: numpy, pandas, matplotlib, часть 1: учебно-методическое пособие / Ю.Ю. Дюличева. М.М. Маршалок – ФГАОУ ВО «КФУ им. В.И. Вернадского». – Симферополь, 2024. – 62 с.

Рекомендовано к печати заседанием кафедры прикладной математики  
протокол №6 от 17 января 2024

Рекомендовано к печати:

Учебно-методическим советом Физико-технического института (структурное подразделение) ФГАОУ ВО «КФУ им. В.И.Вернадского»

протокол №1 от 20 .сентября 2024

Учебно-методическим советом ФГАОУ ВО «КФУ им. В.И.Вернадского»

протокол №2 от 07 февраля 2025

Рецензенты:

Руденко Л. И., кандидат физико-математических наук, заведующая кафедрой информатики Физико-технического института ФГАОУ ВО «КФУ им. В. И. Вернадского»;

Алиев А.И., кандидат технических наук, доцент, декан инженерно-технологического факультета ГБОУВО РК КИПУ имени Февзи Якубова.

В учебно-методическом пособии рассматриваются основы анализа и визуализации данных с помощью базовых библиотек numpy, pandas, matplotlib. В частности, изучаются основные подходы для обработки массивов данных в numpy, извлечение данных по запросу из датафреймов в pandas, построение различных типов графиков и диаграмм в matplotlib.

Учебно-методическое пособие предназначено для направления подготовки 01.04.04 Прикладная математика, однако может быть рекомендовано к использованию в обучении других направлений подготовки и в рамках реализации программ дополнительного профессионального образования в ходе освоения тем, связанных с анализом и визуализацией данных.

## Оглавление

Список сокращений и условных обозначений.....	4
Введение .....	5
Раздел 1. Основные возможности библиотеки numpy .....	7
1.1. Способы создания объекта numpy.ndarray.....	7
1.2. Основные операции с объектом numpy.ndarray.....	9
Практическое занятие №1. Основные возможности пакета numpy.linalg .....	16
Задания для самостоятельной работы.....	18
Раздел 2. Основные возможности библиотеки pandas.....	20
2.1. Основные методы работы с объектом Series в pandas.....	21
Практическое занятие №2. Основные операции с объектом Series в pandas.....	25
Задания для самостоятельной работы.....	28
2.2. Основные методы работы с объектом DataFrame в pandas.....	29
Практическое занятие №3. Основные операции с объектом DataFrame в pandas. Отбор данных по критерию.....	34
Задания для самостоятельной работы.....	39
Практическое занятие №4. Основные операции с объектами DataFrame в pandas. Способы соединения датафреймов.....	41
Практическое занятие №5. Основные операции с объектами DataFrame в pandas. Метод построения запросов query().....	44
Раздел 3. Основные возможности библиотеки matplotlib.....	48
3.1. Построение графиков функций.....	48
Задания для самостоятельной работы.....	54
3.2. Построение диаграмм.....	55
Задания для самостоятельной работы.....	59
Список использованной и рекомендуемой литературы.....	62

## **Список сокращений и условных обозначений**

numpy — numeric Python

pandas — panel data

## **Введение**

Настоящее учебно-методическое пособие включает в себя описание возможностей базовых библиотек анализа и визуализации данных — numpy, pandas, matplotlib. В учебно-методическом пособии описываются высокоскоростные операции обработки массивов данных в numpy, включая решение задач линейной алгебры; демонстрируются более широкие возможности библиотеки pandas при обработке табличных данных, включая различные способы соединения таблиц и группировки данных, использования фильтрации по нескольким условиям с извлечением нужных строк и столбцов из табличных данных и т. п.; рассматриваются основные приемы построения различных графиков и диаграмм в matplotlib для информативной визуализации результатов анализа данных.

Материалы пособия предназначены для обучения студентов направления подготовки 01.04.04 Прикладная математика, однако могут быть рекомендованы к использованию в образовательном процессе других направлений подготовки бакалавриата и магистратуры, а также в рамках программ дополнительного профессионального образования при освоении темы «Анализ и визуализация данных».

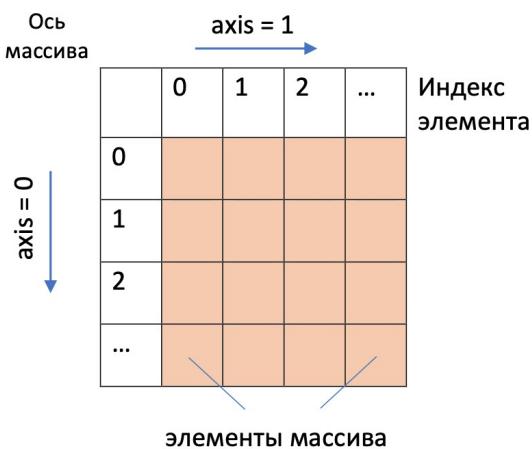
Для закрепления полученных знаний обучающимся предлагаются кейсы для извлечения и визуализации полученных закономерностей, эмпирической базой для которых служат датасеты, размещенные в открытом доступе на платформах UCI и Kaggle.

## Раздел 1. Основные возможности библиотеки numpy

### 1.1. Способы создания объекта numpy.ndarray

Numpy (numeric Python) – специальная библиотека Python для работы с многомерными массивами данных, включая математические функции линейной алгебры.

Основным объектом библиотеки является однородный N-мерный массив — объект `numpy.ndarray`. Измерения массива называются осями (`axis`). Например, в двумерном массиве соответственно две оси: нулевая ось – вертикальная (вычисление будет «по столбцам»), первая – горизонтальная (вычисление будет «по строкам»).



Для использования numpy необходимо импортировать библиотеку в проект.

```
#Импортируем библиотеку numpy
import numpy as np
```

Рассмотрим атрибуты объектов `numpy.ndarray`:

- 1) атрибут `shape` выводит на экран кортеж чисел, показывающих размерность массива, т.е. количество элементов по конкретной оси;
- 2) с помощью атрибута `ndim` можно посмотреть количество измерений массива;
- 3) атрибут `size` позволяет получить общее количество элементов массива;
- 4) атрибут `dtype` позволяет получить тип данных элементов массива.

```
import numpy as np
arr = np.array([[-15, 27, 38], [76, -13, 48]])
print(arr.shape) # форма массива, его размер
print(arr.ndim) # число осей массива (измерений)
print(arr.size) # общее количество элементов
print(arr.dtype) # тип данных массива
```

```
(2, 3)
2
6
int64
```

## Способы создания объекта `numpy.ndarray`

1 способ — применение метода `array()`, принимающего в качестве параметра список или кортеж. Ниже приведены примеры создания объектов `numpy.ndarray` на основе одномерного и двумерного списка.

```
import numpy as np
arr = np.array([-15, 27, 38])
print(arr)
print(type(arr))
```

```
[-15 27 38]
<class 'numpy.ndarray'>
```

```
import numpy as np
arr = np.array([[-15, 27, 38], [76, -13, 48]])
print(arr)
```

```
[[ -15  27  38]
 [ 76 -13  48]]
```

2 способ — Для создания массива из нулей или единиц используются методы `zeros()` и `ones()` соответственно. Оба метода принимают в качестве аргумента кортеж с размерами. По умолчанию элементы имеют тип данных `float`.

```
a = np.zeros([3,2])
print(a)
```

```
[[ 0.  0.]
 [ 0.  0.]
 [ 0.  0.]]
```

```
b = np.ones([2,2])
print(b)
```

```
[[ 1.  1.]
 [ 1.  1.]]
```

3 способ — С помощью метода `eye()` создаётся единичная матрица (квадратная матрица, у которой на главной диагонали расположены единицы, а остальные элементы равны нулю). В качестве параметра указывается размерность матрицы. Ниже создается единичная матрица размерности 3x3.

```
a = np.eye(3)
print(a)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

4 способ — Метод **empty()** создает массив из случайных значений (так называемого «мусора» из оперативной памяти), а метод **full()** позволяет заполнить все элементы массива определенным значением. Ниже создается матрица размерности 3x3 со случайными значениями и матрица 3x3, заполненная определенным значением, равным -5.

```
c = np.empty([3, 3])
print(c)
```

```
[2.5e-323 2.5e-323 2.5e-323]
[2.5e-323 2.5e-323 2.5e-323]
[2.5e-323 2.5e-323 2.5e-323]]
```

```
b = np.full([3, 3], -5)
print(b)
```

```
[-5 -5 -5]
[-5 -5 -5]
[-5 -5 -5]]
```

5 способ — заполнение элементов массива значениями из некоторого диапазона с определенным шагом выполняется с помощью метода **arange()**, имеющего три параметра — начало, конец последовательности и её шаг. Для заполнения элементов массива вещественными числами из некоторого диапазона используется метод **linspace()**. Ниже создается массив с элементами из диапазона от -5 до 5 включительно с шагом 2 и массив с элементами из диапазона [0,1] с шагом 0.25.

```
a = np.arange(-5, 6, 2)
print(a)
```

```
[-5 -3 -1 1 3 5]
```

```
r = np.linspace(0, 1, 5)
print(r)
```

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

6 способ — Метод **tile()** позволяет создать массив с повторяющимися значениями. На вход принимаются два аргумента: массив значений, которые будут повторяться, и количество повторений. Для создания многомерного массива вторым аргументом следует указать размерность необходимого массива по всем осям. Ниже создается одномерный массив, в котором элементы 1 и 3 повторяются 5 раз и двумерный массив с повторением элементов 1,3.

```
a = np.tile([1,3], 5)
print(a)
```

```
[1 3 1 3 1 3 1 3 1 3]
```

```
b = np.tile([1,3], [2,4])
print(b)
```

```
[[1 3 1 3 1 3 1 3]
 [1 3 1 3 1 3 1 3]]
```

## 1.2. Основные операции с объектом numpy.ndarray

Рассмотрим унарные операции, которые представлены в классе ndarray: сумма элементов (метод **sum()**), нахождение минимального (метод **min()**) или максимального (метод **max()**) значений массива, вычисление среднего значения (метод **mean()**).

```
arr = np.array([[[-15, 27, 38], [76, -13, 48]]])  
print(arr.sum()) # сумма всех элементов массива  
print(arr.min()) # минимальный элемент массива  
print(arr.max()) # максимальный элемент  
print(arr.mean())# среднее значение
```

```
161  
-15  
76  
26.83333333333332
```

По умолчанию данные методы применяются к любому массиву как к списку чисел. Но если передать как аргумент значение axis, операция будет производиться для указанной оси: 1) axis=0 — для столбца; 2) axis=1 — для строки.

```
arr = np.array([[[-15, 27, 38], [76, -13, 48]]])  
print(arr.sum (axis = 0)) # сумма всех элементов в каждом столбце  
print(arr.min (axis = 0)) # минимальный элемент в каждом столбце  
print(arr.max (axis = 1)) # максимальный элемент в каждой строке  
print(arr.mean(axis = 1)) # среднее значение элементов в каждой строке
```

```
[61 14 86]  
[-15 -13 38]  
[38 76]  
[16.66666667 37. ]
```

Для изменения размерностей массива в numpy существует несколько методов. Первым рассмотрим метод **reshape()**. Чтобы получить новый массив достаточно передать в качестве аргумента новый размерности данных. Ниже одномерный массив преобразуется сначала в матрицу размерности 2x3, а затем 3x2.

<pre>arr = np.array([-15, 27, 38, 76, -13, 48]) print(arr.reshape(2,3))</pre>	<pre>print(arr.reshape(3,2))</pre>
<pre>[[[-15 27 38]  [ 76 -13 48]]]</pre>	<pre>[[[-15 27]  [ 38 76]  [-13 48]]]</pre>
<pre>print(arr)</pre>	
<pre>[-15 27 38 76 -13 48]</pre>	

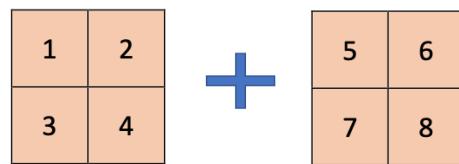
Как видно из примера выше, метод **reshape()** не изменяет исходный массив, поэтому рассмотрим второй метод — **resize()**. Данный метод напрямую изменяет форму исходного массива. В качестве аргумента передается новая размерность.

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

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

### **Объединение и разбиение массивов**

Для объединения массивов в Numpy используется два метода **hstack()** и **vstack()**. Рассмотрим работу этих методов на примере двух двумерных массивов:



1) Метод **hstack()** позволяет объединить массивы по оси `axis = 1`, т.е. по горизонтали:

1	2	5	6
3	4	7	8

2) Метод **vstack()** позволяет объединить по оси `axis = 0`, т.е. по вертикали:

1	2
3	4
5	6
7	8

Ниже приведены примеры объединения массивов по различным осям.

```
a = np.array([[-15, 27], [38, 47]])
b = np.array([[76, -13], [74, -18]])
c = np.hstack((a, b)) # объединить массивы по горизонтали
print(c)
```

```
[[ -15  27  76 -13]
 [ 38  47  74 -18]]
```

```
a = np.array([[-15, 27], [38, 47]])
b = np.array([[76, -13], [74, -18]])
c = np.vstack((a, b)) # объединить массивы по вертикали
print(c)
```

```
[[ -15  27]
 [ 38  47]
 [ 76 -13]
 [ 74 -18]]
```

Аналогичным образом работают методы **column\_stack()** и **row\_stack()** – объединяют массивы по столбцам и строкам соответственно.

```
a = np.array([-15, 27, 38, 47])
b = np.array([76, -13, 74, -18])
c = np.column_stack((a, b)) #объединение массивов по столбцам
print(c)
```

```
[[ -15  76]
 [ 27 -13]
 [ 38  74]
 [ 47 -18]]
```

```
a = np.array([-15, 27, 38, 47])
b = np.array([76, -13, 74, -18])
d = np.row_stack((a, b)) #объединение массивов по строкам
print(d)
```

```
[[ -15  27  38  47]
 [ 76 -13  74 -18]]
```

Для разбиения массива на подмассивы используются методы **hsplit()** и **vsplit()**. При помощи **hsplit()** массив равномерно разбивается горизонтально, достаточно передать как аргумент имя массива и количество необходимых подмассивов (количество элементов обязательно должно быть кратно числу возвращаемых массивов). Ниже приведен пример разбиения одномерного массива на два одномерных массива.

```
arr = np.array([-15, 28, 73, 45, 55, -76])
r = np.hsplit(arr, 2)
print(r)
```

```
[array([-15,  28,  73]), array([ 45,  55, -76])]
```

Метод `vsplit()` аналогично разбивает массив вертикально. Данный метод работает в том случае, если количество строк в массиве больше 1 и кратно числу подмассивов.

```
arr = np.array([[-15, 28, 73], [45, 55, -76]])
r = np.vsplit(arr, 2)
print(r)
```

[array([[ -15, 28, 73]]), array([[ 45, 55, -76]])]

## Сортировка массивов

В numpy метод `sort()` возвращает отсортированный по возрастанию массив. Обязательными параметрами метода являются массив и ось, по которой будет происходить сортировка.

```
arr = np.array([10, 27, 3, 15, 57, 9])
r = np.sort(arr, axis = 0)
print(r)
```

[ 3 9 10 15 27 57]

## Фильтрация по условию

В numpy можно задавать условия. Приведем пример отбора элементов массива со значениями не менее 17.

```
import numpy as np
a = np.array([-5, 12, 18, -6, 17, 38, 27])
condition = a >= 17
a1 = a[condition]
print(a1)
```

[18 17 38 27]

Ещё одним способом фильтрации данных является использование метода `np.where()`, который возвращает индексы элементов, удовлетворяющих условию. С помощью `np.where()` отберем отрицательные элементы массива.

```
import numpy as np
a = np.array([-5, 12, 18, -6, 17, 38, 27])
indexes = np.where(a<0)
print(a[indexes])
```

[-5 -6]

## Индексы, срезы, итерирование массивов

Обращение к элементам массива в Numpy аналогично обращению к элементам списка Python. Существует два типа индексов:

- положительные, при которых нумерация элементов идёт слева направо и начинается с нуля;
- отрицательные, которые отсчитывают элементы массива с конца (справа налево); индекс последнего равен -1, предпоследнего - -2 и так далее.

Положительные индексы	0	1	2	3	4	5	6	7	8
Элементы массива а	5	-8	43	100	16	-37	23	18	-7
Отрицательные индексы	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
arr = np.array([-5, 4, 47, -6, 8, -21, 13, -7, 9])
print(arr[0]) #обращение к 1-му элементу
#нумерация ведется с нуля
print(arr[-1]) # обращение к последнему элементу списка
```

```
-5
9
```

Для обращения к элементу многомерного массива следует указать индекс по каждой оси в виде кортежа чисел (последовательность чисел, разделенная запятыми). Например, чтобы обратиться к элементу со значением 8, необходимо указать вторую строку и второй столбец, т.е. arr[1, 1].

	0	1	2
0	-5	4	47
1	-6	8	-21
2	13	-7	9

Для того чтобы извлечь определенную строку, достаточно указать только первый индекс: например для извлечения второй строки - arr[1], что будет эквивалентно записи среза arr[1, :]. Для обращения к определенному столбцу необходимо использовать полный срез в качестве индексов. К примеру, для обращения ко второму столбцу следует указать arr[:, 1].

```
arr = np.array([[-5, 4, 47], [-6, 8, -21], [13, -7, 9]])
print(arr[1, 1])
print(arr[:,1]) # второй столбец
print(arr[1,:]) # вторая строка
```

```
8
[ 4  8 -7]
[ -6   8 -21]
```

Если массивы более высокой размерности, то для выделения среза возможны два равноценных варианта записи. NumPy позволяет вместо одинаковых полных подряд идущих срезов использовать многоточие: `a[..., 1, 1]` эквивалентно `a[:, :, 1, 1]`.

Итерирование многомерных массивов начинается с нулевой оси с использованием вложенных циклов. Если необходимо перебрать массив поэлементно, как одномерный, так и многомерный, то для этого используется атрибут `flat`. Ниже приведен пример перебора элементов массива с помощью атрибута `flat`.

```
arr = np.array([[-5, 4, 47], [-6, 8, -21], [13, -7, 9]])
for element in arr.flat: #использование атрибута flat
    print(element, end=' ')
```

```
-5 4 47 -6 8 -21 13 -7 9
```

Другой возможностью перебора элементов является вложенный цикл:

```
arr = np.array([[-5, 4, 47], [-6, 8, -21], [13, -7, 9]])
for row in arr:
    for element in row: #итерация двумерного массива
        print(element, end=' ')
```

```
-5 4 47 -6 8 -21 13 -7 9
```

## Операции над массивами в NumPy

В отличии от работы со стандартными структурами Python, где необходимо использовать циклы для выполнения арифметических операций, в NumPy все математические операции над массивами выполняются сразу поэлементно – как с массивом и числовым значением, так и с элементами нескольких массивов. Основные арифметические операции представлены в таблице ниже.

Операция	Описание
+	Сложение элементов массива с числом или с элементами массива
-	Вычитание элементов массива с числом или с элементами массива
*	Умножение элементов массива на число или на элементы массива
/	Деление элементов массива на число или на элементы массива
//	Целочисленное деление элементов массива на число или на элементы массива
%	Вычисление остатка от деления элементов массива на число или на элементы массива
**	Возведение элементов массива в степень (указывается определенное число или элементы массива)

Приведем примеры арифметических операций над одномерным массивом и числовым значением:

```
import numpy as np
arr = np.array([[1, 3, 5], [7, 9, 11]])
a=5
print('+5 -- ',arr+a)
print('-5 -- ',arr-a)
print('*5 -- ',arr*a)
print('/5 -- ',arr/a)
print('**5 -- ',arr**a)

+5 -- [[ 6  8 10]
 [12 14 16]]
-5 -- [[-4 -2  0]
 [ 2  4  6]]
*5 -- [[ 5 15 25]
 [35 45 55]]
/5 -- [[0.2 0.6 1. ]
 [1.4 1.8 2.2]]
**5 -- [[    1    243   3125]
 [16807 59049 161051]]
```

Как видно, каждая из операций выполняется над каждым элементом массива arr.

Рассмотрим операции над двухмерными массивами. Так как все операции между массивами совершаются поэлементно, то размерности массивов должны совпадать, однако в numpy существуют алгоритмы транслирования массивов: если длина первого массива больше длины второго, то второй будет дополнен копиями своих элементов до необходимости размерности по определенным правилам.

Правила транслирования массивов:

- 1) к массиву с меньшим числом осей добавляются новые так, чтобы размерности совпадали (добавление всегда происходит при axis = 0);
- 2) оси с одним элементом расширяются копиями этого элемента до необходимой размерности.

```

import numpy as np
arr = np.array([[1, 3, 5], [7, 9, 11]])
arr1 = np.array([[2, 4, 6], [8, 10, 12]])
print(arr+arr1)
print(arr-arr1)
print(arr*arr1)
print(arr/arr1)

```

```

[[ 3  7 11]
 [15 19 23]]
 [[-1 -1 -1]
 [-1 -1 -1]]
 [[ 2 12 30]
 [ 56 90 132]]
 [[0.5          0.75        0.83333333]
 [0.875       0.9         0.91666667]]

```

В библиотеке numpy так же представлено множество математических операций для работы с массивами. На входе каждый из методов принимает массив ndarray.

Операция	Описание
<b>np.abs(arr)</b>	Вычисление модуля от элементов массива
<b>np.around(arr)</b>	Округление элементов массива до ближайшего целого
<b>np.log(arr)</b>	Вычисление натурального логарифма от элементов массива
<b>np.log10(arr)</b>	Вычисление десятичного логарифма от элементов массива
<b>np.sin(arr), np.cos(arr), np.tan(arr), np.cotan(arr)</b>	Вычисление тригонометрических функций от элементов массива

## Практическое занятие №1

### Тема: Основные возможности пакета numpy.linalg

*Цель:* изучить операции линейной алгебры, реализованные в библиотеке numpy.linalg

#### Вопросы для подготовки к практическому занятию:

- Перечислите способы создания объекта numpy.ndarray
- Перечислите основные методы библиотеки numpy, позволяющие выполнять математические операции над массивами
- Перечислите арифметические операции над массивами в numpy и приведите примеры
- Продемонстрируйте примеры применения методов hsplit() и vsplit(), реализованных в библиотеки numpy
- Продемонстрируйте примеры применения методов hstack() и vstack(), реализованных в библиотеки numpy
- В чем разница между методами reshape() и resize()?

7. Что такое транслирование массивов и перечислите правила транслирования массивов?
8. Что такое срез? Как из матрицы размерности 4x4 извлечь элементы второй и третьей строки и 1,2 и 3-го столбца?
9. Перечислите основные операции линейной алгебры, реализованные в numpy.linalg
10. Какие методы позволяют вычислить определитель матрицы и получить обратную матрицу.

#### **Литература для подготовки к практическому занятию:**

1. Н. Кайда Самоучитель по Python для начинающих. Часть 24: Основы работы с NumPy. [Электронный ресурс] // Proplib. URL: <https://proplib.io/p/samouchitel-po-python-dlya-nachinayushchih-chast-24-osnovy-raboty-s-numpy-2023-07-10> (дата обращения: 11.02.2024).
2. 100 NumPy задач // Python 3 для начинающих. URL: <https://pythonworld.ru/numpy/100-exercises.html> (дата обращения: 11.02.2024).
3. Нилаб Нисчал Python — это просто. Пошаговое руководство по программированию и анализу данных // БХВ. - 2023. ISBN: 978-5-9775-6849-4.
4. Кристиан Хилл Научное программирование на Python // ДМК:Пресс. - 2021. ISBN: 978-5-97060-914-9

#### **Ход занятия:**

Библиотека numpy.linalg обеспечивает эффективную низкоуровневую реализацию стандартных алгоритмов линейной алгебры. Методы библиотеки numpy.linalg позволяют вычислять ранг, определитель, собственные значения матрицы, матричное и векторное произведение и т. п., а также решать линейные и тензорные уравнения.

Пусть дана матрица

$$A = \begin{pmatrix} 1 & 2 & 4 \\ -1 & 5 & 2 \\ 0 & 1 & -2 \end{pmatrix}$$

Требуется найти след матрицы, ранг матрицы, обратную матрицу и вычислить определитель.

Зададим матрицу как объект numpy.ndarray

```
import numpy as np  
  
A = np.array([[1, 2, 4],  
             [-1, 5, 2],  
             [0, 1, -2]])  
print(A)
```

```
[[ 1  2  4]  
 [-1  5  2]  
 [ 0  1 -2]]
```

1. Для вычисления следа матрицы будем использовать метод trace() из numpy:

```
tr = np.trace(A)  
print(tr)
```

4

2. Для вычисления ранга матрица будем использовать метод matrix\_rank() из numpy.linalg:

```
r = np.linalg.matrix_rank(A)  
print(r)
```

3

3. Для построения обратной матрицы будем использовать метод inv() из numpy.linalg:

```
B = np.linalg.inv(A)  
print(B)
```

```
[[ 0.6 -0.4  0.8 ]  
 [ 0.1  0.1  0.3 ]  
 [ 0.05  0.05 -0.35]]
```

4. Для вычисления определителя используется метод det() из numpy.linalg:

```
d = np.linalg.det(A)  
print(d)
```

-19.99999999999996

Пусть задана система линейных уравнений:

$$\begin{cases} 4x - 2y = 22, \\ 6x + y = 45. \end{cases}$$

Требуется найти решение методами библиотеки numpy.linalg.

Используем метод solve() для получения решения системы двух линейных уравнений:

```
import numpy as np

a = np.array([[4, -2], [6, 1]])
b = np.array([22, 45])
r = np.linalg.solve(a, b)
print(np.linalg.solve(a, b))
```

[7. 3.]

### Задания для самостоятельной работы

Выполните задания<sup>1</sup> согласно варианту по таблице:

Вариант №	Номера заданий	Вариант №	Номера заданий
1	1, 8, 14, 18, 27	11	5, 9, 11, 19, 26
2	2, 7, 16, 2, 24,	12	6, 8, 13, 21, 30
3	3, 8, 13, 20, 25	13	3, 14, 19, 22, 25
4	4, 9, 11, 14, 23	14	1, 7, 16, 22, 29
5	5, 10, 17, 26, 30	15	2, 8, 15, 19, 28
6	6, 9, 17, 21, 28	16	4, 9, 16, 21, 26
7	3, 7, 15, 20, 29	17	6, 11, 18, 21, 24
8	2, 8, 19, 22, 25	18	5, 13, 16, 25, 27
9	1, 10, 14, 23, 26	19	3, 7, 15, 19, 26
10	4, 9, 15, 20, 27	20	2, 9, 20, 26, 30

Примечание:  $N$  – номер варианта,  $n = N+10$ ,  $m = N+1$

1. Создать вектор размерности  $n$ , у которого элементы на чётных позициях равны 0, а на нечётных – единице.
2. Создать вектор размерности  $n$ , заполненный Вашим порядковым номером в списке группы.
3. Создать вектор и заполнить его числами в диапазоне  $[0, n]$ .
4. Создать вектор размерности  $n$ , заполненный случайными числами и развернуть этот вектор.
5. Вычислить количество нулевых элементов в векторе размерности  $n$  со случайными значениями.
6. Сформировать вектор размерности  $n$  со случайными значениями в диапазоне  $[0, 9]$ . Вывести на экран индексы элементов, значения которых меньше 5.
7. В заданной матрице найти наименьший элемент в каждой строке.
8. В заданной матрице найти наибольший элемент в каждом столбце.
9. Задана матрица. Вывести на экран первый и последний столбцы.
10. Задана матрица. Вывести на экран первую и последнюю строки.
11. Дана матрица. Вывести на экран все элементы, находящиеся в строках с четными номерами.

<sup>1</sup>Задания составлены на основе ресурса

[https://www.w3resource.com/python-exercises/numpy/numpy\\_100\\_exercises\\_with\\_solutions.php](https://www.w3resource.com/python-exercises/numpy/numpy_100_exercises_with_solutions.php)

12. Сформировать матрицу  $n \times m$ , вывести на экран  $N$ -строку, где  $N$  – номер вашего варианта.
13. Данна матрица  $n \times n$ . Вывести на экран элементы, стоящие на главной диагонали.
14. Данна квадратная матрица. Вывести на экран сумму элементов, стоящих на побочной диагонали.
15. Вывести  $k$ -ю строку и  $r$ -й столбец заданной матрицы. Значения  $k$ ,  $r$  ввести с клавиатуры.
16. Создать нулевую матрицу, элементы, находящиеся в крайних столбцах и строках приравнять к единице.
17. Создать квадратную матрицу  $n \times n$  из целых чисел в диапазоне  $[0, 9]$ . Число  $n$  в данном задании ввести с клавиатуры.
18. Сформировать матрицу  $n \times m$ , состоящую из нулей, угловые элементы равны единице.
19. Сформировать матрицу, в каждой строке которой на случайном месте находится нулевой элемент.
20. Сформировать случайную верхнюю треугольную матрицу размерности  $8 \times 8$  (в верхней треугольной матрице все элементы, находящиеся ниже главной диагонали, равны нулю).
21. Найдите сумму всех элементов матрицы.
22. Найти минимальное, максимальное и среднее значение элементов в матрице размерности  $4 \times 4$  со случайными значениями.
23. В произвольной матрице отсортировать по возрастанию элементы столбцов.
24. В произвольной матрице вычислить след матрицы (сумму элементов на главной диагонали).
25. Задать матрицу размером  $3 \times 3$ , вывести на экран определитель матрицы.
26. Сформировать две матрицы  $A$  и  $B$  размерности  $n \times n$  из случайных элементов, вывести на экран сумму, разность и произведение данных матриц.
27. В матрице поменять местами первую и последнюю строки.
28. В матрице поменять местами первый и последний столбцы.
29. Найти наибольший элемент матрицы и заменить все нечетные элементы на него.
30. Сформировать матрицу размерности  $n \times n$ , заполнить её единицами в шахматном порядке, используя метод `tile`
31. Решить систему уравнений 3 способами, используя метод `solve()` и метод `det()` (метод Крамера), а также метод `inv()` (метод обратной матрицы):

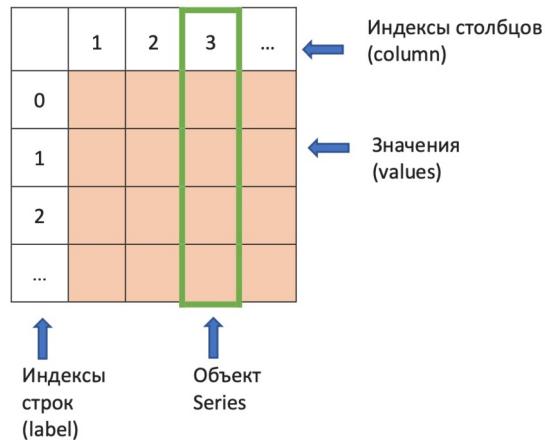
$$\begin{cases} 3x_1 + x_2 + 2x_3 = 4 \\ -x_1 + 2x_2 - 3x_3 = 1 \\ -2x_1 + x_2 + x_3 = -2 \end{cases}$$

## Раздел 2. Основные возможности библиотеки pandas

### 2.1. Основные методы работы с объектом Series в pandas

Pandas («panel data» - «панельные данные») – это высокоуровневая библиотека Python для работы с табличными данными, позволяющая обрабатывать, структурировать и анализировать их. Работа с таблицами строится поверх ранее изученной библиотеки numpy, которая является более низкоуровневой.

Основными объектами Pandas являются массивы данных: **Series** и **DataFrame**. Разберем их более детально.



**Объект Series** — объект, схожий с одномерным массивом, однако содержащий индексированные данные. Особенностью является то, что Series состоит из двух связанных массивов, один из которых содержит данные (правый столбец), а второй – индексы (левый столбец). Синтаксис класса **Pandas Series** выглядит следующим образом:

```
class Pandas.Series (data = None, index = None, dtype = None, name = None, copy = False)
```

*data* – массив данных, на основе которых будет создан объект Series;

*index* – массив индексов, используемых для доступа к массиву данных (длины заданных массивов должны совпадать);

*dtype* – аргумент, определяющий тип данных объекта Series;

*name* – имя объекта Series;

*copy* – параметр, создающий копию массива, если значение равно True (по умолчанию значение параметра False).

Вместе с тем, в большинстве случаев при создании объекта Series передаются в качестве аргументов только *data* и *index* (является необязательным). Существует несколько различных вариантов *data*, на базе которых возможно создать объект Series:

- списки Python;
- словари Python;
- массивы numpy;
- скалярные величины.

Рассмотрим их более детально.

1. Самый простой способ создать объект Series – преобразовать список Python, передав его в качестве параметра.

```
import pandas as pd #импортируем библиотеку
series = pd.Series([15, -3, 7, 11]) #создание объекта Series из списка
print(series)

0    15
1   -3
2     7
3    11
dtype: int64
```

2. При создании объекта Series из словаря ключи автоматически станут индексами, а значения словаря – значениями объекта.

```
import pandas as pd
dict1 = {'a':10, 'b':20, 'c':30}
series = pd.Series(dict1)
print(series)

a    10
b    20
c    30
dtype: int64
```

3. В качестве аргумента data также может быть передан массив numpy. Индексы элементов будут определены автоматически.

```
#импортируем библиотеки
import pandas as pd
import numpy as np
ndarr = np.array([10, 20, 30, 40, 50])
series = pd.Series(ndarr)
print(series)

0    10
1    20
2    30
3    40
4    50
dtype: int64
```

4. Задать объект Series так же возможно с использованием константы. Достаточно передать в качестве аргумента константу и массив индексов. Каждому элементу будет присвоено значение константы.

```
const = 5
series = pd.Series(const, index = ['a', 'b', 'c'])
print(series)

a    5
b    5
c    5
dtype: int64
```

Как было сказано выше, отличительной чертой объектов Pandas является наличие индекса, который может быть задан отдельно при помощи атрибута `index` значениями любого типа:

```
series = pd.Series([15, -3, 7, 11], index=['a', 'b', 'c', 'd']) # явное описание индексов
print(series)

a    15
b   -3
c     7
d    11
dtype: int64
```

В случае, если индекс не был явно описан пользователем, то он будет задан автоматически последовательностью от 0 до N-1, где N – количество элементов объекта Series. Также одним из атрибутов объекта Series и его индекса является атрибут `name`, при помощи которого задается имя объекта и индекса соответственно.

```
series = pd.Series([15, -3, 7, 11], index=['a', 'b', 'c', 'd'])
series.name = 'numbers'
series.index.name = 'letters'
print(series)

letters
a    15
b   -3
c     7
d    11
Name: numbers, dtype: int64
```

Обращение к элементам объекта или получения среза возможно по индексу. При числовом индексе – аналогично как при работе с элементами списка Python, для индексов, заданных явно, – как при обращении к ключу словаря Python.

```
series = pd.Series([15, -3, 7, 11])
print(series[1])

series = pd.Series([15, -3, 7, 11], index=['a', 'b', 'c', 'd'])
print(series['b'])

series = pd.Series([15, -3, 7, 11])
print(series[:2])

-3
-3
0    15
1   -3
dtype: int64
```

Для элементов объекта Series возможно осуществить групповое присваивание значений, достаточно указать индексы элементов:

```
series = pd.Series([15, -3, 7, 11], index=['a', 'b', 'c', 'd'])
series[['a', 'b']] = 0
print(series)

a    0
b    0
c    7
d   11
dtype: int64
```

Для того, чтобы провести поиск данных по критерию зададим условие для объекта Series:

```
series = pd.Series([15, -3, 7, 11])
print(series[series < 0])

1   -3
dtype: int64
```

В pandas есть возможность производить математические вычисления со значениями объекта Series. Рассмотрим некоторые основные методы:

- ***min()*** – нахождение минимального элемента объекта Series;
- ***max()*** – нахождение максимального элемента объекта Series;
- ***mean()*** – метод, который вычисляет среднее арифметическое всех элементов объекта Series;
- ***median()*** – вычисление медианного значения элементов объекта Series;
- ***std()*** – стандартное отклонение элементов объекта Series;
- ***unique()*** – метод, возвращающий уникальные значения элементов объекта Series;
- ***value\_counts()*** – нахождение частоты встречаемости каждого уникального элемента объекта Series;

- ***sort\_values()*** – сортировка элементов объекта Series по возрастанию или убыванию;
- ***head()*** – метод, возвращающий первые 5 элементов объекта Series;
- ***tail()*** – метод, возвращающий последние 5 элементов объекта Series.

## Практическое занятие №2

### Тема: Основные операции с объектом Series в pandas

Цель: изучить основные операции над объектом Series

#### Вопросы для подготовки к практическому занятию:

1. Перечислите способы создания объекта Series
2. Опишите как создать объект Series на основе словаря
3. Приведите примеры того, как можно работать с объектами Series по аналогии с векторами: складывать, умножать вектор на число и т.п.
4. Как получить пересекающиеся элементы двух объектов Series?
5. Как вывести уникальные элементы объекта Series?
6. Как найти индексы объекта Series кратные 11?
7. Как вывести все чётные индексы объекта Series?
8. Как получить доступ к конкретному элементу объекта Series по индексу?  
Приведите пример
9. Как объединить два объекта Series по вертикали и по горизонтали?
10. Как извлекать данные из объекта Series с помощью задания условия?

#### Литература для подготовки к практическому занятию:

1. Изучаем pandas. Урок 2. Структуры данных Series и DataFrame [Электронный ресурс] // DevPractice. URL: <https://devpractice.ru/pandas-series-and-dataframe-part2/> (дата обращения: 11.02.2024).
2. 120 задач Pandas. Часть 1. Series. URL: [https://grossmend.com/blog/post/pandas\\_120\\_part\\_1/](https://grossmend.com/blog/post/pandas_120_part_1/) (дата обращения: 11.02.2024)
3. Шамаев И. Python 3 Pandas: Объекты Series и DataFrame. Построение Index. URL: <https://python.ivan-shamaev.ru/> (дата обращения: 11.02.2024)
4. Нилаб Нисчал Python — это просто. Пошаговое руководство по программированию и анализу данных. — БХВ, 2023. ISBN: 978-5-9775-6849-4.
5. Пасхавер Б. Pandas в действии — СПб.: Питер, 2023. - 512 с.
5. Кристиан Хилл Научное программирование на Python // ДМК:Пресс. - 2021. ISBN: 978-5-97060-914-9

#### Ход занятия:

Требуется создать два объекта Series и вывести на экран: 1) те элементы, которые есть в первом объекте и нет во втором; затем те элементы, которые есть во втором, но нет в первом; 2) те элементы, которые находятся в пересечении двух объектов Series; 3) непересекающиеся элементы двух

объектов; 4) те, элементы первого объекта, которые расположены на нечетных местах, а затем те элементы второго объекта, которые расположены на четных местах.

Создадим два объекта Series:

```
import pandas as pd
ser1 = pd.Series([15, 18, -53, 37, -78])
ser2 = pd.Series([36, 15, -77, -53, 12])
print(ser1)
print(ser2)
```

```
0    15
1    18
2   -53
3    37
4   -78
dtype: int64
0    36
1    15
2   -77
3   -53
4    12
dtype: int64
```

1) Для того, чтобы отобрать элементы, которые есть в первом объекте Series, но нет во втором воспользуемся методом, реализующим теоретико-множественную разность — setdiff1d() из numpy, при этом возвращается в качестве результата объект numpy.ndarray:

```
res = np.setdiff1d(ser1, ser2)
print(res)
```

```
[-78  18  37]
```

Другой способ для достижения аналогичного результата позволяет вернуть объект Series. Отберем элементы из ser1, которые принадлежат ser2 с помощью метода isin(), который возвращает либо True либо False и применим отрицание ~. В результате получим элементы из ser1 за исключением общих элементов у ser1 и ser2. Результат работы метода isin() с отрицанием приведен ниже:

```
print(~ser1.isin(ser2))
```

```
0    False
1     True
2    False
3     True
4     True
dtype: bool
```

Затем выведем на экран элементы из ser1, которые получили значение True:

```
res = ser1[~ser1.isin(ser2)]
print(res)
```

```
1    18
3    37
4   -78
dtype: int64
```

Аналогично можно получить элементы из ser2, которых нет в объекте ser1:

```
res = ser2[~ser2.isin(ser1)]
print(res)
```

```
0    36
2   -77
4    12
dtype: int64
```

2) Выведем на экран те элементы, которые находятся в пересечении двух объектов Series. Это можно сделать с помощью метода intersect1d из библиотеки numpy и преобразуем к типу Series:

```
series_inter = pd.Series(np.intersect1d(ser1, ser2))
print(series_inter)
```

```
0   -53
1    15
dtype: int64
```

3) Выведем на экран непересекающиеся элементы двух объектов Series. Для этого сначала вычтем из первого множества элементов объекта ser1 элементы второго множества ser2. Затем вычтем из второго множества элементов объекта ser2 элементы первогомножества ser1 и объединим с помощью метода union1d() из numpy, преобразовав результат к типу Series:

```
res1 = ser1[~ser1.isin(ser2)]
res2 = ser2[~ser2.isin(ser1)]
res_union = pd.Series(np.union1d(res1, res2))
print(res_union)
```

```
0   -78
1   -77
2    12
3    18
4    36
5    37
dtype: int64
```

или тоже самое с помощью метода setdiff1d() из numpy:

```
res1 = pd.Series(np.setdiff1d(ser1, ser2))
res2 = pd.Series(np.setdiff1d(ser2, ser1))
res_union1 = pd.Series(np.union1d(res1, res2))
print(res_union1)

0   -78
1   -77
2    12
3    18
4    36
5    37
dtype: int64
```

4) Выведем на экран те, элементы первого объекта, которые расположены на нечетных местах, а затем те элементы второго объекта, которые расположены на четных местах. Отберем нечетные индексы объекта ser1 с помощью условия `ser1.index%2!=0` и сохраним индексы в переменную `ind1`, выведем элементы на нечетные местах — `ser1[ind1]`:

```
ind1 = ser1[ser1.index % 2 != 0].index
print(ind1)
print(ser1[ind1])
```

```
Int64Index([1, 3], dtype='int64')
1    18
3    37
dtype: int64
```

Аналогично выведем на экран элементы объекта `ser2`, имеющие четные индексы:

```
ind2 = ser2[ser2.index % 2 == 0].index
print(ind2)
print(ser2[ind2])
```

```
Int64Index([0, 2, 4], dtype='int64')
0    36
2   -77
4    12
dtype: int64
```

## Задания для самостоятельной работы

*Примечание: четные варианты выполняют задания с четными номерами, нечетные варианты – нечетные.*

1. Создать объект Series, состоящий из числовых элементов. Найти все элементы, кратные 7. На экран вывести их индексы.
2. Создать произвольный объект Series. Подсчитать количество символов в объекте.
3. Создать два произвольных объекта Series: А и В. Вывести на экран все непересекающиеся элементы и их индексы.

4. Создать произвольный объект Series. Подсчитать количество повторений (частоту) элементов.
5. Создать два произвольных объекта Series: А и В. Объединить объекты вертикально и горизонтально.
6. Создать два произвольных объекта Series А и В, состоящие из числовых элементов. Найти евклидово расстояние между объектами.

#### **Дополнительное задание:**

Создайте объект Series, содержащий список группы. В качестве индексов используйте номер в журнале старосты. Индекс старосты задать «starosta».

Выведите: 1) список всех элементов;

2) каждого второго из списка;

3) создать два объекта Series для подгрупп, распределив в первую подгруппу студентов с нечётным индексом, а во вторую – с чётным.

## **2.2. Основные методы работы с объектом DataFrame в pandas**

Объект **DataFrame** является аналогом двумерного массива с заданными индексами строк и имен столбцов, и представляет собой табличную структуру данных, которые называются значения. Конструктор класса DataFrame имеет следующий вид:

```
class pandas.DataFrame (data = None, index = None, columns = None, dtype = None,  
copy = False)
```

*data* – двумерный массив данных или словарь значений, на основе которых будет создан объект DataFrame;

*index* – массив индексов, являющийся именами строк таблицы;

*columns* – массив значений для имён столбцов таблицы;

*dtype* – аргумент, определяющий тип данных объекта DataFrame;

*copy* – параметр, создающий копию массива, если значение равно True (по умолчанию значение параметра False).

Аналогично объекту Series объект DataFrame имеет атрибут **index** – список меток для записей, являющимися именами строк таблицы. К именам столбцов таблицы обеспечивает доступ атрибут **columns**. Если имена строк и столбцов не заданы, то метки задаются автоматически числовыми последовательностями.

Создать объект **DataFrame** можно несколькими способами:

1) на основе двумерного массива numpy:

```

import numpy as np
import pandas as pd
arr = np.array([[1, 2, 3], [10, 20, 30]])
df = pd.DataFrame(arr) #создание DataFrame на основе двумерного массива
print(df)

```

	0	1	2
0	1	2	3
1	10	20	30

2) на основе словаря объектов Series:

```

import pandas as pd #импортируем библиотеку
#создание DataFrame из списка словарей
d = {"column1":pd.Series([1, 2, 3], index=['v1', 'v2', 'v3']), \
      "column2": pd.Series([10, 20, 30], index=['v1', 'v2', 'v3'])}
df1 = pd.DataFrame(d)
print(df1)

```

	column1	column2
v1	1	10
v2	2	20
v3	3	30

3) импортировать данные из файлов формата .csv, .html, .xlsx при помощи методов read\_csv(), read\_html(), read\_excel():

```

import pandas as pd
df = pd.read_csv('путь к файлу')

```

*Рассмотрим работу с объектом DataFrame на примере.*

Для начала создадим DataFrame о крупнейших городах Российской Федерации на основе словаря объектов Series, включающий в себя информацию о населении (млн) и площасти (тыс. квадратных км).

```

import pandas as pd # импортируем библиотеку
df = pd.DataFrame({'город': ['Москва', 'Санкт-Петербург', 'Новосибирск'],
                   'население': [13.02, 5.6, 1.7],
                   'площадь': [2551, 1439, 506] })
print(df)

```

	город	население	площадь
0	Москва	13.02	2551
1	Санкт-Петербург	5.60	1439
2	Новосибирск	1.70	506

Для обращения к столбцу используется операция выбора столбца с указанием метки df['col'], которая возвращает объект Series. Стоит обратить внимание, что обращение к столбцам через атрибут df.страна равносильно обращению через нотацию словарей df['страна'].

```
df = pd.DataFrame({'город': ['Москва', 'Санкт-Петербург', 'Новосибирск'],
                   'население': [13.02, 5.6, 1.7],
                   'площадь': [2551, 1439, 506] })
print(df['город'])
```

```
0      Москва
1    Санкт-Петербург
2    Новосибирск
Name: город, dtype: object
```

Для получения среза данных необходимо указать необходимый диапазон строк либо через числовые значения, либо через обращение к меткам:

```
df = pd.DataFrame({'город': ['Москва', 'Санкт-Петербург', 'Новосибирск'],
                   'население': [13.02, 5.6, 1.7], 'площадь': [2551, 1439, 506] })
print(df[0:2])
```

```
      город   население   площадь
0     Москва     13.02      2551
1  Санкт-Петербург     5.60      1439
```

Изменить автоматически заданные индексы можно через атрибут `index`, через атрибут `name` — задать название для данного столбца:

```
df = pd.DataFrame({'город': ['Москва', 'Санкт-Петербург', 'Новосибирск'],
                   'население': [13.02, 5.6, 1.7], 'площадь': [2551, 1439, 506] })
df.index = ['МОС', 'СПБ', 'НОВ']
df.index.name = 'Аббревиатура'
print(df)
```

```
      город   население   площадь
Аббревиатура
МОС           Москва     13.02      2551
СПБ       Санкт-Петербург     5.60      1439
НОВ       Новосибирск     1.70       506
```

Метод `rename()` используется для переименования столбцов:

```
df = pd.DataFrame({'город': ['Москва', 'Санкт-Петербург', 'Новосибирск'],
                   'население': [13.02, 5.6, 1.7], 'площадь': [2551, 1439, 506] },
                   index = ['МОС', 'СПБ', 'НОВ'])
df.index.name = 'Аббревиатура'
df = df.rename(columns={'город': 'мегаполис'}) #переименуем столбец «город» в «мегаполис»
print(df)
```

Для извлечения определенного подмножества данных необходимо в операции выбора строк записать условие выбора через логическое выражение:

```

df = pd.DataFrame({'город': ['Москва', 'Санкт-Петербург', 'Новосибирск'],
                   'население': [13.02, 5.6, 1.7], 'площадь': [2551, 1439, 506]},
                   index = ['МОС', 'СПБ', 'НОВ'])
df.index.name = 'Аббревиатура'
#выборка городов с населением больше 5 млн
print(df[df.население > 5])

```

Аббревиатура	город	население	площадь
МОС	Москва	13.02	2551
СПБ	Санкт-Петербург	5.60	1439

Доступ к строкам и столбцам объекта DataFrame так же можно осуществить по числовому индексу:

- .loc — доступ по строковой метке, принимает как аргумент индекс, заданный пользователем;

```

df = pd.DataFrame({
    'город': ['Москва', 'Санкт-Петербург', 'Новосибирск'],
    'население': [13.02, 5.6, 1.7], 'площадь': [2551, 1439, 506]},
    index = ['МОС', 'СПБ', 'НОВ'])
df.index.name = 'Аббревиатура'
#выбор строки по метке
print(df.loc['СПБ'])

```

город	Санкт-Петербург
население	5.6
площадь	1439
Name: СПБ, dtype: object	

- .iloc — доступ по числовому значению номера строки; принимает в качестве аргумента как один индекс, так и срез.

```

df = pd.DataFrame({'город': ['Москва', 'Санкт-Петербург', 'Новосибирск'],
                   'население': [13.02, 5.6, 1.7], 'площадь': [2551, 1439, 506]})
#выбор строки по метке
print(df.iloc[1])

```

город	Санкт-Петербург
население	5.6
площадь	1439
Name: 1, dtype: object	

В pandas есть возможность производить математические вычисления со значениями объекта DataFrame. Например, рассчитаем плотность населения городов и добавим полученные значения в новый столбец «плотность»:

```

df = pd.DataFrame({'город': ['Москва', 'Санкт-Петербург', 'Новосибирск'],
                   'население': [13.02, 5.6, 1.7], 'площадь': [2551, 1439, 506]},
                  index = ['МОС', 'СПБ', 'НОВ'])
df.index.name = 'Аббревиатура'
df['плотность'] = df['население'] / df['площадь'] * 1000000
print(df)

```

	город	население	площадь	плотность
Аббревиатура				
МОС	Москва	13.02	2551	5103.880831
СПБ	Санкт-Петербург	5.60	1439	3891.591383
НОВ	Новосибирск	1.70	506	3359.683794

Для удаления данных из DataFrame воспользуемся методом drop, передав в качестве параметра указанные строки или столбцы. Стоит обратить внимание, что при удалении столбцов необходимо так же передать ось (axis = 1), когда при удалении строк ось (axis = 0) задается автоматически. Пример удаления столбца «плотность»:

```

df = pd.DataFrame({'город': ['Москва', 'Санкт-Петербург', 'Новосибирск'],
                   'население': [13.02, 5.6, 1.7], 'площадь': [2551, 1439, 506]},
                  index = ['МОС', 'СПБ', 'НОВ'])
df.index.name = 'Аббревиатура'
df['плотность'] = df['население'] / df['площадь'] * 1000000
df = df.drop(['плотность'], axis = 1)
print(df)

```

	город	население	площадь
Аббревиатура			
МОС	Москва	13.02	2551
СПБ	Санкт-Петербург	5.60	1439
НОВ	Новосибирск	1.70	506

Рассмотрим пример удаления строки с индексом «МОС»:

```

df = pd.DataFrame({'город': ['Москва', 'Санкт-Петербург', 'Новосибирск'],
                   'население': [13.02, 5.6, 1.7], 'площадь': [2551, 1439, 506]},
                  index = ['МОС', 'СПБ', 'НОВ'])
df.index.name = 'Аббревиатура'
df['плотность'] = df['население'] / df['площадь'] * 1000000
df = df.drop(index = 'МОС')
print(df)

```

	город	население	площадь	плотность
Аббревиатура				
СПБ	Санкт-Петербург	5.6	1439	3891.591383
НОВ	Новосибирск	1.7	506	3359.683794

Допускается так же использование стандартного del, при этом необходимо указать имя столбца, который будет удален:

```

df = pd.DataFrame({
    'город': ['Москва', 'Санкт-Петербург', 'Новосибирск'],
    'население': [13.02, 5.6, 1.7], 'площадь': [2551, 1439, 506]
}, index = ['МОС', 'СПБ', 'НОВ'])
df.index.name = 'Аббревиатура'
df['плотность'] = df['население'] / df['площадь'] * 1000000
del df['плотность']
print(df)

```

	город	население	площадь
Аббревиатура			
МОС	Москва	13.02	2551
СПБ	Санкт-Петербург	5.60	1439
НОВ	Новосибирск	1.70	506

### Практическое занятие №3

**Тема: Основные операции с объектом DataFrame в pandas. Отбор данных по критерию**

*Цель:* изучить способы написания условий для извлечения данных из датафрейма, а также способы вставки и удаления столбцов

**Вопросы для подготовки к практическому занятию:**

1. Перечислите способы создания объекта DataFrame
2. В чем разница между методами loc() и iloc()?
3. С помощью какого метода можно переименовать столбец в структуре DataFrame? Приведите пример
4. Перечислите способы удаления столбца в структуре DataFrame? Приведите пример
5. Перечислите основные атрибуты объекта DataFrame.
6. Как изменить числовые значения в конкретном столбце объекта DataFrame? Приведите пример
7. Как извлекать данные из объекта DataFrame с помощью задания условия?

**Литература для подготовки к практическому занятию:**

1. Изучаем pandas. Урок 2. Структуры данных Series и DataFrame [Электронный ресурс] // DevPractice. URL: <https://devpractice.ru/pandas-series-and-dataframe-part2/> (дата обращения: 11.02.2024).
2. 120 задач Pandas. Часть 1. Series. URL: [https://grossmend.com/blog/post/pandas\\_120\\_part\\_1/](https://grossmend.com/blog/post/pandas_120_part_1/) (дата обращения: 11.02.2024)
3. Шамаев И. Python 3 Pandas: Объекты Series и DataFrame. Построение Index. URL: <https://python.ivan-shamaev.ru/> (дата обращения: 11.02.2024)

4. Нилаб Нисчал Python — это просто. Пошаговое руководство по программированию и анализу данных // БХВ. - 2023. ISBN: 978-5-9775-6849-4.
4. Пасхавер Б. Pandas в действии — СПб.: Питер, 2023. - 512 с.
5. Кристиан Хилл Научное программирование на Python // ДМК:Пресс. - 2021. ISBN: 978-5-97060-914-9

### Ход занятия:

Требуется создать объект DataFrame на основе словаря

```
my_dict = { 'Surname' : ['Петров', 'Иванов', 'Сидоров',
    'Смирнов', 'Арканов', 'Дибров'],
    'Age' : [58, 32, 44, 21, 69, 38],
    'Salary' : [22580, 37660, 23800, 19750, 32490, 53125],
    'Category': ['вторая', 'первая', 'вторая', 'вторая', 'первая',
    'высшая']} и выполнить фильтрацию DataFrame по значениям столбцов: 1) отобрать данные о сотрудниках высшей и первой категории и отдельно данные о сотрудниках второй категории; 2) данные о сотрудниках, зарплата которых больше 32000 и возраст не равен 32 годам; 3) извлечь данные о сотрудниках, чьи фамилии начинаются с буквы «С» и отдельно вывести данные о сотрудниках, чьи фамилии оканчиваются на «ров»; 4) создать новый столбец с данными о зарплате, которая была увеличена на 10%; 5) добавить столбец 'Experience'(стаж) с данными [27,10,21,2,45,16]; 6) удалить из датафрейма столбец 'Salary' и удалить из датафрейма строки по критерию «стаж < 20»; 7) добавить столбец 'Name'(Имя) и выполнить слияние столбцов 'Name' и 'Surname' в один 'Full_Name', после чего удалить столбцы 'Name' и 'Surname'.
```

Создадим объект DataFrame на основе словаря:

```
import pandas as pd
my_dict = { 'Surname' : ['Петров', 'Иванов', 'Сидоров', 'Смирнов', 'Арканов', 'Дибров'],
    'Age' : [58, 32, 44, 21, 69, 38],
    'Salary' : [22580, 37660, 23800, 19750, 32490, 53125],
    'Category': ['вторая', 'первая', 'вторая', 'вторая', 'первая',
    'высшая']}
df = pd.DataFrame(my_dict)
print(df)
```

	Surname	Age	Salary	Category
0	Петров	58	22580	вторая
1	Иванов	32	37660	первая
2	Сидоров	44	23800	вторая
3	Смирнов	21	19750	вторая
4	Арканов	69	32490	первая
5	Дибров	38	53125	высшая

Извлечем из датафрейма (таблицы) данные о сотрудниках высшей и первой категории и отдельно данные о сотрудниках второй категории. Для этого создадим список из двух элементов 'первая' и 'высшая' и с помощью метода `isin` отберем нужные значения из столбца 'Category':

```

c = ['первая', 'высшая']
df1= df[df['Category'].isin(c)]
print(df1)

```

	Surname	Age	Salary	Category
1	Иванов	32	37660	первая
4	Арканов	69	32490	первая
5	Дибров	38	53125	высшая

Отдельно выведем данные о сотрудниках второй категории с помощью отрицания ~ предыдущего условия, которое мы использовали для извлечения данных о сотрудниках первой и высшей категорий:

```

c = ['первая', 'высшая']
df3= df[~df['Category'].isin(c)]
print(df3[['Surname', 'Age', 'Salary']])

```

	Surname	Age	Salary
0	Петров	58	22580
2	Сидоров	44	23800
3	Смирнов	21	19750

Извлечем из датафрейма (таблицы) данные о сотрудниках, зарплата которых больше 32000 и возраст не равен 32 годам:

```

df2= df[(df['Salary']>32000) & (df['Age']!=32)]
print(df2)

```

	Surname	Age	Salary	Category
4	Арканов	69	32490	первая
5	Дибров	38	53125	высшая

Извлечем данные о сотрудниках, чьи фамилии начинаются с буквы «С» и отдельно выведем данные о сотрудниках, чьи фамилии оканчиваются на «ров». Для этого будем использовать методы для работы со строками — startswith() и endswith() соответственно:

- данные о сотрудниках, фамилии которых начинаются с буквы «С»

```

df4 = df[df['Surname'].str.startswith('С')]
print(df4)

```

	Surname	Age	Salary	Category
2	Сидоров	44	23800	вторая
3	Смирнов	21	19750	вторая

- данные о сотрудниках, фамилии которых оканчиваются подстрокой «ров»

```
df5 = df[df['Surname'].str.endswith('ров')]
print(df5)
```

	Surname	Age	Salary	Category
0	Петров	58	22580	вторая
2	Сидоров	44	23800	вторая
5	Дибров	38	53125	высшая

Создадим новый столбец «New\_Salary» с данными о зарплате, которая была увеличена на 10%:

- 1 способ — с помощью методы apply и lambda-выражения:

```
def increase_salary(s):
    s = 1.1*s
    return s
df['New_Salary'] = df['Salary'].apply(lambda x: increase_salary(x))
print(df)
```

	Surname	Age	Salary	Category	New_Salary
0	Петров	58	22580	вторая	24838.0
1	Иванов	32	37660	первая	41426.0
2	Сидоров	44	23800	вторая	26180.0
3	Смирнов	21	19750	вторая	21725.0
4	Арканов	69	32490	первая	35739.0
5	Дибров	38	53125	высшая	58437.5

- 2

- способ — операции с данными столбца «Salary»:

```
df['New_Salary'] = 1.1*df['Salary']
print(df)
```

	Surname	Age	Salary	Category	New_Salary
0	Петров	58	22580	вторая	24838.0
1	Иванов	32	37660	первая	41426.0
2	Сидоров	44	23800	вторая	26180.0
3	Смирнов	21	19750	вторая	21725.0
4	Арканов	69	32490	первая	35739.0
5	Дибров	38	53125	высшая	58437.5

Добавим столбец 'Experience'(стаж) с данными [27,10,21,2,45,16].

- 1 способ — df['Название\_столбца'] = список значений

```

experience = [27,10,21,2,45,16]
df['Experience'] = experience
print(df)

```

	Surname	Age	Salary	Category	New_Salary	Experience
0	Петров	58	22580	вторая	24838.0	27
1	Иванов	32	37660	первая	41426.0	10
2	Сидоров	44	23800	вторая	26180.0	21
3	Смирнов	21	19750	вторая	21725.0	2
4	Арканов	69	32490	первая	35739.0	45
5	Дибров	38	53125	высшая	58437.5	16

2 способ — использование метода `insert()` для объекта DataFrame. Метод `insert()` содержит следующие параметры — номер столбца для вставки (нумерация начинается с нуля), название столбца, список значений столбца, параметр `allow_duplicates = True`, если допустимо создавать столбцы с одинаковыми названиями, по умолчанию этот параметр равен `False`.

```

experience = [27,10,21,2,45,16]
df.insert(1,'Experience',experience, allow_duplicates=True)
print(df)

```

	Surname	Experience	Experience	Age	Salary	Category	New_Salary
0	Петров	27	27	58	22580	вторая	24838.0
1	Иванов	10	10	32	37660	первая	41426.0
2	Сидоров	21	21	44	23800	вторая	26180.0
3	Смирнов	2	2	21	19750	вторая	21725.0
4	Арканов	45	45	69	32490	первая	35739.0
5	Дибров	16	16	38	53125	высшая	58437.5

	Experience
0	27
1	10
2	21
3	2
4	45
5	16

3 способ — использование метода `assign()` для создания датафрейма на основе существующего с добавлением столбца:

```

my_dict = { 'Surname' : ['Петров', 'Иванов', 'Сидоров', 'Смирнов', 'Арканов', 'Дибров'],
            'Age' : [58, 32, 44, 21, 69, 38],
            'Salary' : [22580, 37660, 23800, 19750, 32490, 53125],
            'Category': ['вторая', 'первая', 'вторая', 'вторая', 'первая', 'высшая']}
df = pd.DataFrame(my_dict)
experience = [27,10,21,2,45,16]
df1 = df.assign(Experience = experience)
print(df)
print('\n Новый датафрейм \n')
print(df1)

```

	Surname	Age	Salary	Category
0	Петров	58	22580	вторая
1	Иванов	32	37660	первая
2	Сидоров	44	23800	вторая
3	Смирнов	21	19750	вторая
4	Арканов	69	32490	первая
5	Дибров	38	53125	высшая

Новый датафрейм

	Surname	Age	Salary	Category	Experience
0	Петров	58	22580	вторая	27
1	Иванов	32	37660	первая	10
2	Сидоров	44	23800	вторая	21
3	Смирнов	21	19750	вторая	2
4	Арканов	69	32490	первая	45
5	Дибров	38	53125	высшая	16

Удалим из датафрейма столбец 'Salary' и удалим из датафрейма строки по критерию «стаж < 20» с помощью метода drop(). Метод drop() содержит следующие основные параметры: список индексов или столбцов для удаления, ось (axis) — указываем 1 для столбцов и 0 — для строк; inplace=True — для выполнения удаления непосредственно в этом датафрейме.

```
df1['New_Salary'] = 1.1*df['Salary']

df1.drop(['Salary'], axis=1, inplace=True)
print(df1)
```

	Surname	Age	Category	Experience	New_Salary
0	Петров	58	вторая	27	24838.0
1	Иванов	32	первая	10	41426.0
2	Сидоров	44	вторая	21	26180.0
3	Смирнов	21	вторая	2	21725.0
4	Арканов	69	первая	45	35739.0
5	Дибров	38	высшая	16	58437.5

Приведем пример удаления строк по условию «стаж < 20»:

```
df1.drop(df1[df1['Experience']<20].index, inplace=True)
print(df1)
```

	Surname	Age	Category	Experience	New_Salary
0	Петров	58	вторая	27	24838.0
2	Сидоров	44	вторая	21	26180.0
4	Арканов	69	первая	45	35739.0

Добавим столбец 'Name'(Имя) и выполним слияние столбцов 'Name' и 'Surname' в один 'Full\_Name', после чего удалим столбцы 'Name' и 'Surname'. Слияние значений двух столбцов выполним с помощью метода cat() с возможностью указания параметра разделитель sep между значениями двух столбцов, подлежащих слиянию.

```
names = ['Иван', 'Григорий', 'Артём']
df1['Name'] = names
df1['Full_Name'] = df1['Surname'].str.cat(df1['Name'], sep = ' ')
df1.drop(['Name', 'Surname'], axis=1, inplace=True)
print(df1)
```

	Age	Category	Experience	New_Salary	Full_Name
0	58	вторая	27	24838.0	Петров Иван
2	44	вторая	21	26180.0	Сидоров Григорий
4	69	первая	45	35739.0	Арканов Артём

### Задания для самостоятельной работы

Для выполнения заданий использовать ссылку<sup>2</sup> с датасетом о пассажирах Титаника:

Вариант №	Номера заданий	Вариант №	Номера заданий
1	1, 17, 8, 14	11	11, 9, 19, 7
2	2, 7, 16, 11	12	12, 8, 14, 17
3	3, 20, 8, 14	13	13, 7, 19, 3
4	4, 14, 11, 8	14	14, 19, 9, 2
5	5, 17, 8, 10	15	15, 8, 1, 10
6	6, 17, 3, 10	16	16, 7, 17, 3
7	7, 20, 16, 3	17	17, 6, 9, 11
8	8, 12, 5, 19	18	18, 13, 5, 7
9	9, 17, 13, 6	19	19, 6, 2, 15
10	10, 20, 15, 4	20	20, 9, 2, 18

1. Переименовать столбец в DataFrame: «Name» на «ФИО»
2. Переименовать столбец в DataFrame: «Age» на «Возраст»
3. Отобразить имена пассажиров, у которых отсутствует информация о возрасте.
4. Отобрать всех пассажиров в возрасте от 30 до 50 лет.
5. Подсчитать какой процент пассажиров выжили.
6. Подсчитать какой процент пассажиров путешествовали 1 классом.
7. Найти средний возраст всех пассажиров мужчин.
8. Отобразить имена всех выживших пассажирах. Подсчитать их

2 <https://raw.githubusercontent.com/Grossmend/CSV/master/titanic/data.csv>

количество.

9. Отобрать всех пассажиров мужчин старше 40.
10. Отобрать всех пассажиров женщин младше 50.
11. Отобразить имя и возраст каждого десятого пассажира.
12. Отобрать каждого четного пассажира и напечатать следующую информацию: имя, возраст, данные о том, выжил или нет.
13. Отсортировать все столбцы по наименованию
14. Отобразить всю информацию о каждом сотом пассажире.
15. Подсчитать количество выживших пассажиров женского пола.
16. Задать количество строк и столбцов для отображения объектов Pandas:  
10 строк, 5 столбцов
17. Задать количество строк и столбцов для отображения объектов Pandas:  
14 строк, 3 столбца
18. Создать новый столбец со значениями 1.
19. Изменить местами столбцы «Name» и «Age» объект DataFrame.
20. Поменять местами первую и десятую строки объекта DataFrame.

#### **Дополнительное задание:**

*Примечание: выполняется по вариантам (нечетные номера в списке выполняют 1 вариант, четные – 2 вариант):*

**1 вариант.** Создать объект DataFrame с данными о планетах солнечной системы (название, диаметр планеты, масса, расстояние от Солнца)

- 1) Отобрать планеты, что расположены дальше от Солнца, чем Земля.
- 2) Отобрать 3 крупнейшие по массе планеты.
- 3) Отсортировать планеты по возрастанию по столбцу «диаметр».

**2 вариант.** Создать объект DataFrame с данными о континентах Земли (название, площадь, численность населения, часть занимаемой суши в процентном соотношении)

- 1) Отобрать континенты, название которых начинается на «А».
- 2) Отобрать 3 крупнейших терриориально континента.
- 3) Отсортировать континенты по убыванию по части занимаемой суши.

### **Практическое занятие №4**

**Тема: Основные операции с объектами DataFrame в pandas. Способы соединения датафреймов**

**Цель:** изучить способы соединения датафреймов

#### **Вопросы для подготовки к практическому занятию:**

1. Перечислите способы создания датафреймов DataFrame
2. Перечислите способы соединения датафреймов DataFrame
3. Приведите пример левого внешнего соединения датафреймов

#### 4. Приведите пример внутреннего соединения датафреймов

##### **Литература для подготовки к практическому занятию:**

1. Изучаем pandas. Урок 2. Структуры данных Series и DataFrame [Электронный ресурс] // DevPractice. URL: <https://devpractice.ru/pandas-series-and-dataframe-part2/> (дата обращения: 11.02.2024).
2. 120 задач Pandas. Часть 1. Series. URL: [https://grossmend.com/blog/post/pandas\\_120\\_part\\_1/](https://grossmend.com/blog/post/pandas_120_part_1/) (дата обращения: 11.02.2024)
3. Шамаев И. Python 3 Pandas: Объекты Series и DataFrame. Построение Index. URL: <https://python.ivan-shamaev.ru/> (дата обращения: 11.02.2024)
4. Нилаб Нисчал Python — это просто. Пошаговое руководство по программированию и анализу данных // БХВ. - 2023. ISBN: 978-5-9775-6849-4.
4. Пасхавер Б. Pandas в действии — СПб.: Питер, 2023. - 512 с.
5. Кристиан Хилл Научное программирование на Python // ДМК:Пресс. - 2021. ISBN: 978-5-97060-914-9

##### **Ход занятия:**

Требуется создать два датафрейма на основе двух словарей: my\_dict1 = { 'Surname' : ['Петров', 'Иванов', 'Сидоров', 'Смирнов', 'Арканов', 'Дибров'], 'Age' : [58, 32, 44, 21, 69, 38], 'Salary' : [22580, 37660, 23800, 19750, 32490, 53125], 'Category' : ['вторая', 'первая', 'вторая', 'вторая', 'первая', 'высшая']} и my\_dict2 = { 'Surname' : ['Афанасьев', 'Рысков', 'Романов', 'Янышев'], 'Age' : [34, 29, 32, 57], 'Salary' : [23410, 18850, 222580, 48760], 'Category' : ['первая', 'вторая', 'первая', 'высшая']} и выполнить: 1) соединение датафреймов методом concat(); 2) выполнить различные способы соединения таблиц (датафреймов) по ключам в соответствие с таблицей:

Параметры how метода соединения merge в pandas	Название соединения	Описание
left	Левое внешнее соединение	Используются ключи только из левого датафрейма
right	Правое внешнее соединение	Используются ключи только из правого датафрейма
outer	Полное внешнее соединение	Используется объединение ключей из двух датафреймов
inner	Внутреннее соединение	Используется пересечение ключей из двух датафреймов

Создадим два датафрейма на основе словарей и соединим датафреймы методом concat():

```

import pandas as pd
my_dict1 = { 'Surname' : ['Петров', 'Иванов', 'Сидоров', 'Смирнов', 'Арканов', 'Дибров'],
             'Age' : [58, 32, 44, 21, 69, 38],
             'Salary' : [22580, 37660, 23800, 19750, 32490, 53125],
             'Category': ['вторая', 'первая', 'вторая', 'вторая', 'первая', 'высшая']}
my_dict2 = { 'Surname' : ['Афанасьев', 'Рысков', 'Романов', 'Янышев'],
             'Age' : [34, 29, 32, 57], 'Salary' : [23410, 18850, 222580, 48760],
             'Category': ['первая', 'вторая', 'первая', 'высшая']}
df1 = pd.DataFrame(my_dict1, index = [0,1,2,3,4,5])
df2 = pd.DataFrame(my_dict2, index = [6,7,8,9])
frames = [df1, df2]
res1 = pd.concat(frames)
print(res1)

```

	Surname	Age	Salary	Category
0	Петров	58	22580	вторая
1	Иванов	32	37660	первая
2	Сидоров	44	23800	вторая
3	Смирнов	21	19750	вторая
4	Арканов	69	32490	первая
5	Дибров	38	53125	высшая
6	Афанасьев	34	23410	первая
7	Рысков	29	18850	вторая
8	Романов	32	222580	первая
9	Янышев	57	48760	высшая

2. Выполним левое внешнее соединение, правое внешнее соединение, полное внешнее соединение и внутреннее соединение двух фреймов. Создадим два датафрейма с ключами:

```

import pandas as pd
my_dict1 = {'key1':['K0','K1','K2','K3','K4','K5'],
            'key2':['K0','K1','K0','K1','K0','K1'],
            'Surname' : ['Петров', 'Иванов', 'Сидоров', 'Смирнов', 'Арканов', 'Дибров'],
            'Age' : [58, 32, 44, 21, 69, 38]}

my_dict2={'key1':['K0','K1','K2','K3','K4','K5'],
          'key2':['K0','K1','K2','K0','K1','K2'],
          'Salary' : [22580, 37660, 23800, 19750, 32490, 53125],
          'Category': ['вторая', 'первая', 'вторая', 'вторая', 'первая', 'высшая']}

df1 = pd.DataFrame(my_dict1)
df2 = pd.DataFrame(my_dict2)
print(df1)
print('\n',df2)

```

	key1	key2	Surname	Age
0	K0	K0	Петров	58
1	K1	K1	Иванов	32
2	K2	K0	Сидоров	44
3	K3	K1	Смирнов	21
4	K4	K0	Арканов	69
5	K5	K1	Дибров	38

	key1	key2	Salary	Category
0	K0	K0	22580	вторая
1	K1	K1	37660	первая
2	K2	K2	23800	вторая
3	K3	K0	19750	вторая
4	K4	K1	32490	первая
5	K5	K2	53125	высшая

Выполним левое внешнее соединение. Для соединения используются ключи левого датафрейма, отсутствующие значения заполняются NaN:

```
res_left = pd.merge(df1,df2,how='left',on=['key1','key2'])
print(res_left)
```

	key1	key2	Surname	Age	Salary	Category
0	K0	K0	Петров	58	22580.0	вторая
1	K1	K1	Иванов	32	37660.0	первая
2	K2	K0	Сидоров	44	NaN	NaN
3	K3	K1	Смирнов	21	NaN	NaN
4	K4	K0	Арканов	69	NaN	NaN
5	K5	K1	Дибров	38	NaN	NaN

Выполним правое внешнее соединение. Для соединения используются ключи правого датафрейма, отсутствующие значения заполняются NaN:

```
res_right = pd.merge(df1,df2,how='right',on=['key1','key2'])
print(res_right)
```

	key1	key2	Surname	Age	Salary	Category
0	K0	K0	Петров	58.0	22580	вторая
1	K1	K1	Иванов	32.0	37660	первая
2	K2	K2	NaN	NaN	23800	вторая
3	K3	K0	NaN	NaN	19750	вторая
4	K4	K1	NaN	NaN	32490	первая
5	K5	K2	NaN	NaN	53125	высшая

Выполним полное внешнее соединение. Для соединения используются все ключи как левого, так и правого датафреймов, отсутствующие значения заполняются NaN:

```
res_outer = pd.merge(df1,df2,how='outer',on=['key1','key2'])
print(res_outer)
```

	key1	key2	Surname	Age	Salary	Category
0	K0	K0	Петров	58.0	22580.0	вторая
1	K1	K1	Иванов	32.0	37660.0	первая
2	K2	K0	Сидоров	44.0	NaN	NaN
3	K3	K1	Смирнов	21.0	NaN	NaN
4	K4	K0	Арканов	69.0	NaN	NaN
5	K5	K1	Дибров	38.0	NaN	NaN
6	K2	K2	NaN	NaN	23800.0	вторая
7	K3	K0	NaN	NaN	19750.0	вторая
8	K4	K1	NaN	NaN	32490.0	первая
9	K5	K2	NaN	NaN	53125.0	высшая

Выполним внутреннее соединение. Для соединения используются ключи общие ключи левого и правого датафреймов, отсутствующие значения заполняются NaN:

```
res_inner = pd.merge(df1,df2,how='inner',on=['key1','key2'])
print(res_inner)
```

	key1	key2	Surname	Age	Salary	Category
0	K0	K0	Петров	58	22580	вторая
1	K1	K1	Иванов	32	37660	первая

## Практическое занятие №5

**Тема: Основные операции с объектами DataFrame в pandas. Метод построения запросов query()**

**Цель:** изучить способы построения запросов с помощью метода query()

**Вопросы для подготовки к практическому занятию:**

1. Перечислите способы создания датафреймов DataFrame
2. Перечислите способы извлечения данных из датафреймов DataFrame
3. Приведите примеры использования метода query()

**Литература для подготовки к практическому занятию:**

- Устинов А. Метод Pandas query(): Запрос DataFrame в Python [Электронный ресурс] // DevPractice. URL: <https://dev-gang.ru/article/metod-pandas-query-zapros-dataframe-v-python-7fz3yfomkc/> (дата обращения: 11.02.2024).
- 120 задач Pandas. Часть 1. Series. URL: [https://grossmend.com/blog/post/pandas\\_120\\_part\\_1/](https://grossmend.com/blog/post/pandas_120_part_1/) (дата обращения: 11.02.2024)
- Шамаев И. Python 3 Pandas: Объекты Series и DataFrame. Построение Index. URL: <https://python.ivan-shamaev.ru/> (дата обращения: 11.02.2024)
- Нилаб Нисчал Python — это просто. Пошаговое руководство по программированию и анализу данных // БХВ. - 2023. ISBN: 978-5-9775-6849-5.
- Кристиан Хилл Научное программирование на Python // ДМК:Пресс. - 2021. ISBN: 978-5-97060-914-9

### **Ход занятия:**

Для датафрейма, созданного на основе словаря `my_dict = {'Surname': ['Петров', 'Иванов', 'Сидоров', 'Смирнов', 'Арканов', 'Дибров'], 'Age': [58, 32, 44, 21, 69, 38], 'Salary': [22580, 37660, 23800, 19750, 32490, 53125], 'Category': ['вторая', 'первая', 'вторая', 'вторая', 'первая', 'высшая']}` с помощью метода `query()` требуется построить следующие запросы: 1) извлечь данные о сотрудниках второй категории; 2) вывести данные о сотрудниках второй категории с заработной платой выше 20000; 3) создайте отдельную переменную со значением равным фамилия сотрудника, например, Смирнов и выведите данные об этом сотруднике, используя в запросе переменную; 4) вывести данные о сотрудниках, чьи фамилии начинаются на «С» или заканчиваются на «нов».

Создадим датафрейм на основе словаря `my_dict = { 'Surname' : ['Петров', 'Иванов', 'Сидоров', 'Смирнов', 'Арканов', 'Дибров'], 'Age' : [58, 32, 44, 21, 69, 38], 'Salary' : [22580, 37660, 23800, 19750, 32490, 53125], 'Category' : ['вторая', 'первая', 'вторая', 'вторая', 'первая', 'высшая']}`:

```
import pandas as pd
my_dict = { 'Surname' : ['Петров', 'Иванов', 'Сидоров', 'Смирнов', 'Арканов', 'Дибров'],
            'Age' : [58, 32, 44, 21, 69, 38],
            'Salary' : [22580, 37660, 23800, 19750, 32490, 53125],
            'Category' : ['вторая', 'первая', 'вторая', 'вторая', 'первая', 'высшая']}
df = pd.DataFrame(my_dict)
print(df)
```

	Surname	Age	Salary	Category
0	Петров	58	22580	вторая
1	Иванов	32	37660	первая
2	Сидоров	44	23800	вторая
3	Смирнов	21	19750	вторая
4	Арканов	69	32490	первая
5	Дибров	38	53125	высшая

Извлечем данные о сотрудниках второй категории:

```
df_second = df.query('Category == "вторая"')
print(df_second[['Surname', 'Age', 'Salary']])
```

	Surname	Age	Salary
0	Петров	58	22580
2	Сидоров	44	23800
3	Смирнов	21	19750

Выведем данные о сотрудниках второй категории с заработной платой выше 20000:

```
df_second_20000 = df.query('Category == "вторая" and Salary > 20000')
print(df_second_20000[['Surname', 'Age']])
```

	Surname	Age
0	Петров	58
2	Сидоров	44

Создадим отдельную переменную со значением равным фамилия сотрудника, например, Смирнов и выведем данные об этом сотруднике, используя в запросе переменную. В тексте запроса перед именем переменной надо поставить @:

```
surname = 'Смирнов'
df_Smirnov = df.query('Surname == @surname')
print(df_Smirnov)
```

	Surname	Age	Salary	Category
3	Смирнов	21	19750	вторая

Выведем данные о сотрудниках, чьи фамилии начинаются на «С» или заканчиваются на «нов»:

```
df_start_or_end = df.query("Surname.str.startswith('C') or Surname.str.endswith('нов')")  
print(df_start_or_end)
```

	Surname	Age	Salary	Category
1	Иванов	32	37660	первая
2	Сидоров	44	23800	вторая
3	Смирнов	21	19750	вторая
4	Арканов	69	32490	первая

## Раздел 3. Основные возможности библиотеки matplotlib

Библиотека **Matplotlib** – это объектно-ориентированная библиотека на языке Python для визуализации данных различной сложности, включающая широкий набор инструментов для работы с двумерной и трехмерной графикой. С помощью встроенного модуля **pyplot** автоматически создается координатная плоскость, графики функций, диаграммы, а также вспомогательные элементы: оси, подписи данных, легенда и т.д. Для работы с данным модулем его необходимо импортировать:

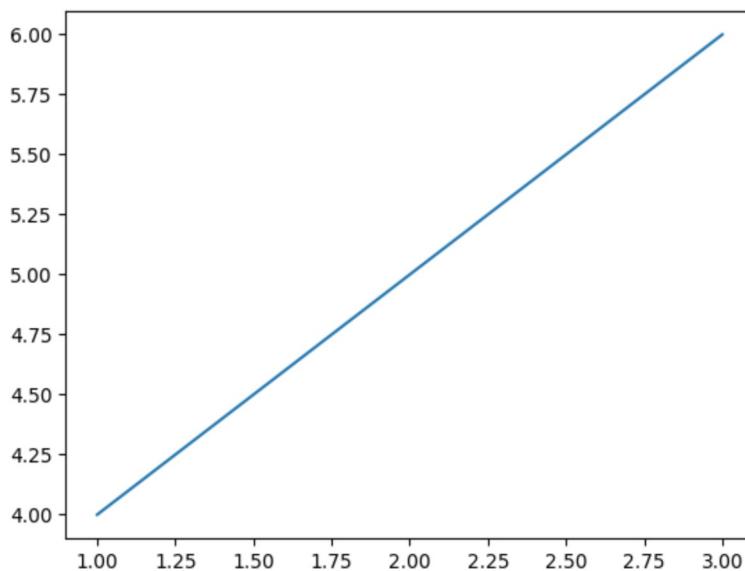
```
import matplotlib.pyplot as plt
```

### 3.1. Построение графиков функций

Для начала построим линейный график по заданным точкам, используя методы:

- **plot()** – метод построения графика, в качестве аргументов принимает массивы данных, являющиеся координатами точек, а также дополнительные параметры (необязательные), рассмотренные ниже;
- **show()** – метод, которая выводит построенный график на экран.

```
import matplotlib.pyplot as plt  
plt.plot([1, 2, 3], [4, 5, 6])  
plt.show()
```



Рассмотрим пример построения графика линейной функции. Необходимо задать массив значений  $x$ , для этого воспользуемся методом `linspace()` библиотеки NumPy. Массив значений  $y$  рассчитаем по формуле:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 10, 10)
y = x
plt.plot(x, y)
plt.show()
```

### ***Отображение осей и легенды на графике***

Как видно выше, данный график не полностью отображает необходимую информацию, поэтому добавим:

- 1) название графика – метод `title()`;
- 2) названия осей абсцисс и ординат – методы `xlabel()` и `ylabel()` соответственно;
- 3) отображение сетки – метод `grid()`.

По умолчанию Matplotlib при построении графика не отображает сетку. При вызове команды `grid()` сетка появится, её размеры автоматически рассчитываются относительно размера осей. Данная сетка считается основной, но есть также и дополнительная. Выбор типа сетки зависит от параметра `which`, который принимает значения: `major`, `minor`, `both` для основной, вспомогательной и одновременного включения и той, и другой сетки соответственно. Следует обратить внимание, что так как сетки привязаны к единичному отрезку на осях, то для работы с вспомогательной сеткой, необходимо вызвать метод `minortick_on()` для отображения вспомогательных меток на осях.

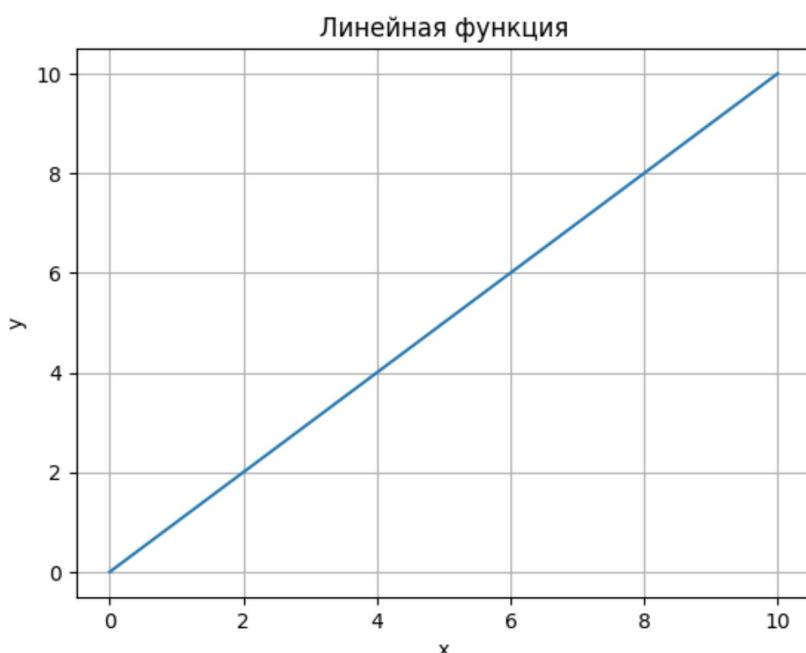
Если на координатной плоскости отображено несколько различных графиков, то необходимо добавить легенду – пояснение. Существует два варианта отображения легенды:

- 1) указать значение параметра `label` в методе построения графика `plot()`;
- 2) использовать отдельный метод `legend()`, указав метку в виде текста как аргумент, а так же определить расположение легенды относительно графика параметром `loc`.

```

import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 10, 10)
y = x
plt.plot(x, y)
plt.title('Линейная функция') #заголовок
plt.xlabel('x') #ось абсцисс
plt.ylabel('y') #ось ординат
plt.grid() #отображение сетки
plt.show()

```



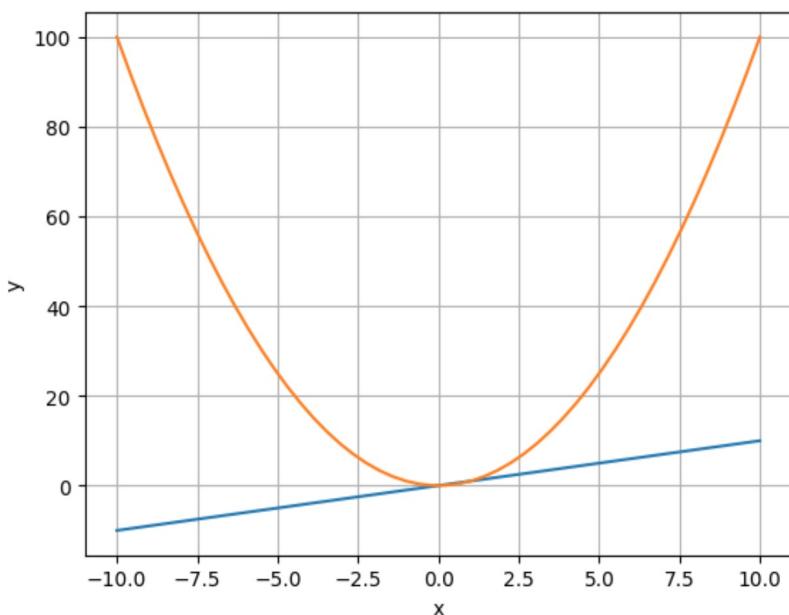
Рассмотрим дополнительные параметры функции plt.plot() для более точной визуализации данных:

Аргумент		Возможные значения
Color	Цвет линии	'r' – red, красный цвет; 'g' – green, зеленый цвет; 'k' – black, черный цвет; 'b' – blue, синий цвет; 'y' – yellow, желтый цвет и другие.
Linestyle	Тип линии	'-' – сплошная линия; '--' – штриховая линия; ':' – пунктирная («точечная») линия; '-.' – штрих-пунктирная линия

linewidth или lw	Толщина линии	По умолчанию равна 1,5; любое значение больше 0 типа float.
Marker	Тип маркера (точки)	'o' - окружность 's' - квадрат 'd' – ромб и другие.
markersize	Размер маркера	Любое значение больше 0 типа float.
Alpha	Степень прозрачности графика	Значение от 0 до 1, где 0 – полностью прозрачный, 1 – непрозрачный.

Для отображения нескольких графиков на одной координатной плоскости достаточно в методе plot() указать поочередно аргументы для каждого графика. Каждый график будет отображаться независимо от диапазона, на котором он задан.

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-10, 10, 50)
y1 = x
y2 = x**2
plt.plot(x, y1, x, y2)
plt.xlabel('x') #ось абсцисс
plt.ylabel('y') #ось ординат
plt.grid() #отображение сетки
plt.show()
```



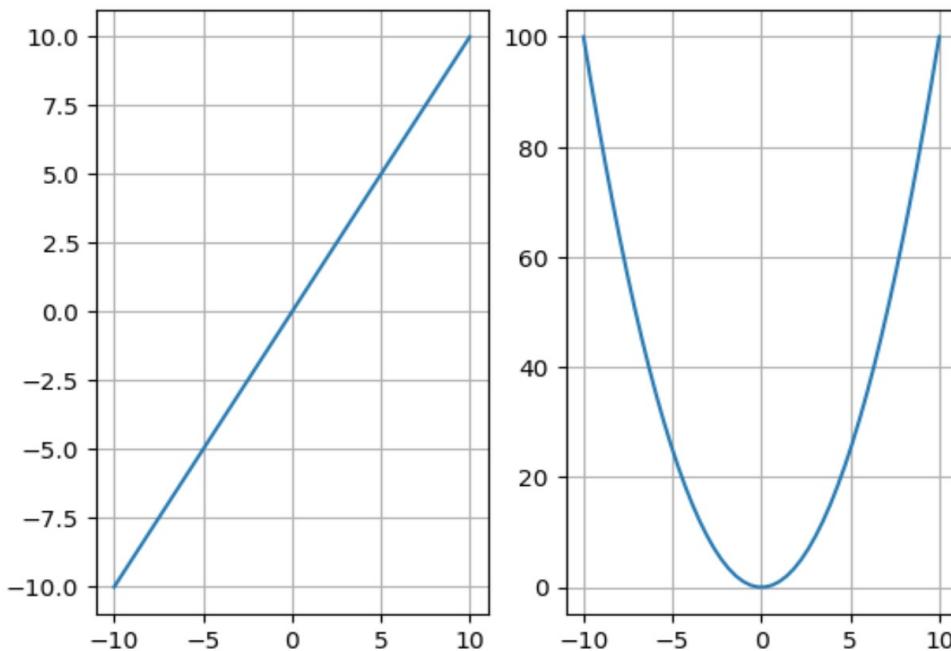
## **Отображение нескольких осей в одном окне**

Для отображения нескольких координатных плоскостей в одном окне используется метод ***subplot()***, аргументами которого являются:

- nrows – количество строк;
- ncols – число столбцов;
- index – индекс текущей координатной плоскости.

Следует учитывать, что параметры прорисовки для каждого графика указываются отдельно.

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-10, 10, 50)
y1 = x
y2 = x**2
plt.subplot(1, 2, 1)
plt.plot(x, y1)
plt.grid()
plt.subplot(1, 2, 2)
plt.plot(x, y2)
plt.grid() #отображение сетки
plt.show()
```



## Построение трёхмерного графика

Инструментарий Matplotlib позволяет строить так же и трехмерные графики. Следует подключить модуль ***matplotlib3d***:

```
from mpl_toolkits import matplotlib3d
```

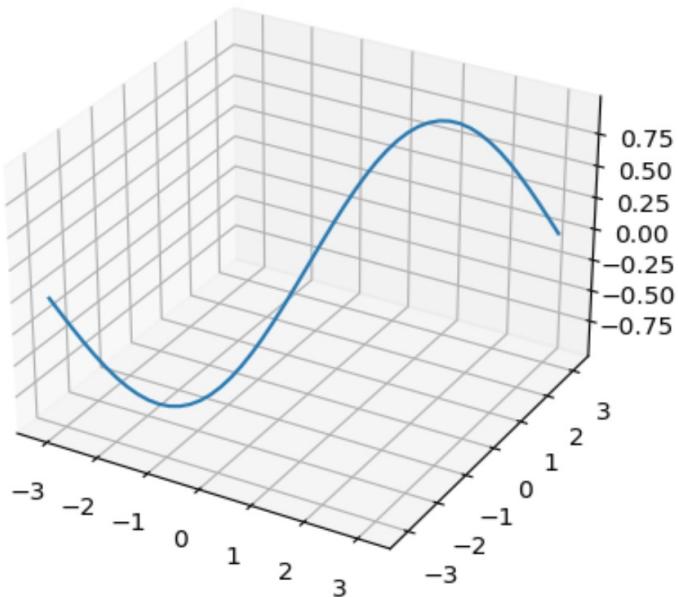
После импорта данного модуля, следует указать в методах построения графика параметр ***projection***, и установить значение равное ‘3d’, и передавать в качестве значений три массива: x, y, z.

Варианты используемых функций для построения трёхмерных графиков:

- `plot()` – линейный двумерный график в трех измерениях;
- `step()` – ступенчатый двумерный график в трех измерениях;
- `scatter()` – точечный трёхмерный график;
- `plot_wireframe()` – построение каркасной поверхности;
- `plot_surface()` – построение непрерывной поверхности.

Рассмотрим пример построения синусоиды в 3D:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(-np.pi, np.pi, 50)
y = x
z = np.sin(x)
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot(x, y, z)
```



Отличительной чертой данных графиков является *интерактивность*. Каждый график вращается вокруг своей оси, что полезно при изучении полученной визуализации.

### Задания для самостоятельной работы

1. Построить график функции  $y = \sin(x) + x \cdot \cos(N \cdot x)$ , где  $N$  – номер варианта.
2. Протабулировать функции  $y_1(x) — y_6(x)$  (см. таблицу ниже) на отрезке  $[-10; 10]$ , если  $a = 2$ ,  $b = 3$ ,  $c = 1$ ,  $d = 4$ . Построить на одном полотне графики функций  $y_1(x), y_2(x), y_3(x)$ , на втором — функций  $y_4(x), y_5(x), y_6(x)$ .

Вар.	$y_1(x)$	$y_2(x)$	$y_3(x)$	$y_4(x)$	$y_5(x)$	$y_6(x)$
1	$\cos(x)$	$\cos(ax)$	$\cos(ax+b)$	$x^2$	$ax^2+bx+c$	$(a+bx)^2$
2	$\sin(x)$	$\sin(a+x)$	$\sin(ax)+b$	$x^2$	$ax^2+c$	$b+(a+x)^2$
3	$\cos(bx)$	$\cos(b+x)$	$a \cos(bx)$	$ax^2$	$ax^2+c$	$ax+bx^2$
4	$ x $	$ a+x $	$a^* x $	$x^2$	$ax^2+cx$	$b-x^2$
5	$\operatorname{arctg}(bx)$	$\operatorname{arctg}(b+x)$	$a^*\operatorname{arctg}(bx)$	$ax^2$	$ax^2+d$	$dx+ax^2$
6	$\cos(x)+a$	$\cos(a+x)$	$\cos(ax^2)$	$-x^2$	$(d-x)^2$	$d-x^2$
7	$b^*\sin(x)$	$b+\sin(x)$	$\sin(b+x)$	$x^3$	$ax^3+c$	$b+(a+x)^3$
8	$\cos(bx)$	$\cos(b+x)$	$a^*\cos(bx)$	$ax^3$	$ax^3+c$	$ax^3+bx^2$
9	$ a^*x $	$a^* x $	$ a+x $	$-x^3$	$ax^2-x^3$	$b-x^3$
10	$b^*\operatorname{arctg}(-x)$	$\operatorname{arctg}(b-x)$	$a^*\operatorname{arctg}(x)+b$	$bx$	$ax^2+d$	$d+bx+ax^2$
11	$\sin(x+a)$	$a+\sin(x)$	$\sin(ax)$	$ax^3$	$ax^3+c$	$(c+ax)^3$
12	$b^*\cos(x)$	$\cos(bx)$	$b+\cos(x)$	$cx^3$	$ax^2+cx^3$	$ax+cx^3$
13	$\sin(x)+c$	$\sin(c+x)$	$\sin(cx^2)$	$dx^2$	$dx^2+b$	$cx+ax^2$
14	$ bx $	$b^* x-c $	$ a-dx^2-b $	$-cx^2$	$(a-x)^3$	$b-x^2$
15	$\operatorname{arctg}(bx)$	$\operatorname{arctg}(b+x)$	$a^*\operatorname{arctg}(bx)$	$ax^2$	$ax^2+d$	$dx+ax^2$
16	$\cos(x)+a$	$\cos(a+x)$	$\cos(ax^2)$	$-x^2$	$(d-x)^2$	$d-x^2$
17	$b^*\operatorname{arctg}(-x)$	$\operatorname{arctg}(b-x)$	$a^*\operatorname{arctg}(x)+b$	$bx$	$ax^2+d$	$d+bx+ax^2$
18	$\sin(x+a)$	$a+\sin(x)$	$\sin(ax)$	$ax^3$	$ax^3+c$	$(c+ax)^3$
19	$\cos(bx)$	$\cos(b+x)$	$a \cos(bx)$	$ax^2$	$ax^2+c$	$ax+bx^2$
20	$ x $	$ a+x $	$a^* x $	$x^2$	$ax^2+cx$	$b-x^2$

#### Дополнительное задание:

Примечание: задание выполняется по вариантам, номер по списку соответствует номеру варианта (для  $N > 7$  - номер варианта равен целочисленной части от  $N/2$ )

Построить трехмерную модель (поверхность и каркас) следующих поверхностей по вариантам. Для построения каркаса используйте метод `plot_wireframe()`, для построения поверхности - `plot_surface()`:

$$1. \quad u \in [-\pi; \pi], v \in \left[ \frac{-\pi}{2}; \frac{\pi}{2} \right]$$

$$x(u, v) = \cos(u) \cdot \cos(v)$$

$$y(u, v) = \sin(u) \cdot \cos(v)$$

$$z(u, v) = \sin(v)$$

### 3.2. Построение диаграмм

Набор инструментов библиотеки Matplotlib позволяет также строить диаграммы для отображения категориальных данных:

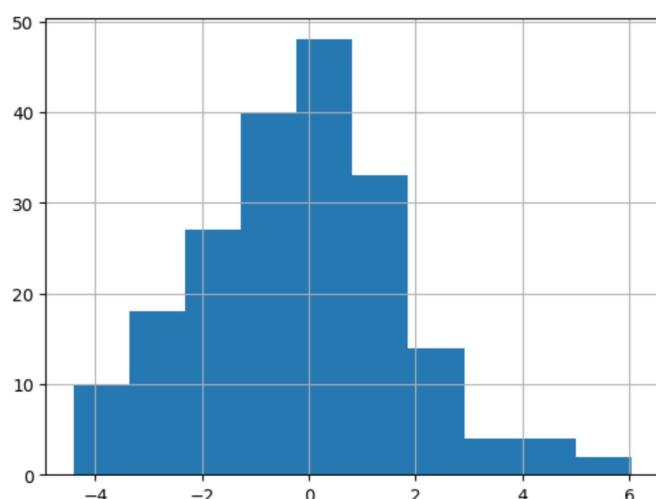
- гистограммы;
- столбчатые диаграммы;
- круговые диаграммы;
- диаграммы рассеяния;
- пузырьковые диаграммы и так далее.

#### Гистограмма

Для построения гистограммы по набору значений используется функция `hist()`, принимающая два аргумента: `data` – набор значений, `bins` – количество бинов (разбиений) гистограммы. По умолчанию значение `bins` равно 10, но может принимать любое значение типа `int`. Если же передать кортеж значений, то функция автоматически рассчитает границы бинов и построит гистограмму с произвольным разбиением.

Для наглядности рассмотрим пример гистограммы распределения случайной величины:

```
import numpy as np
import matplotlib.pyplot as plt
data = np.random.normal(0, 2, 200)
plt.hist(data)
plt.grid()
plt.show()
```



Гистограмма построена на основе вектора из 200 случайных величин. Высота каждого столбца определяется количеством величин, находящихся в определенном интервале. Так как количество бинов не было указано, то метод разбил диапазон на 10 равных интервалов.

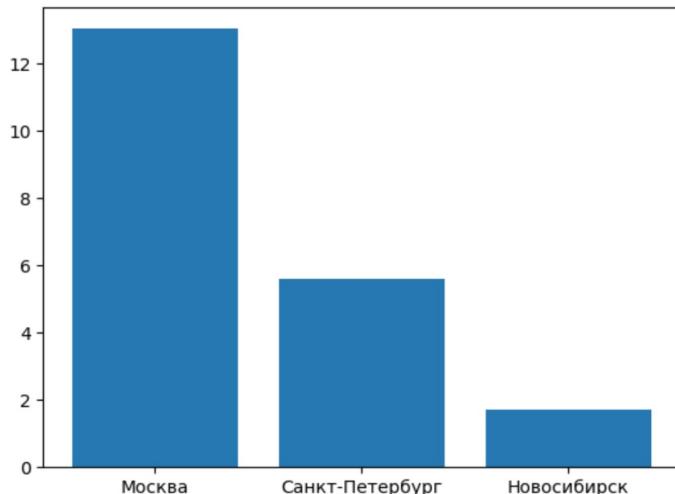
### Столбчатая диаграмма

В отличии от гистограммы, в столбчатой диаграмме на горизонтальной оси располагаются не числовые значения, а категориальные. Для построения такой диаграммы применимы два метода на выбор:

- *bar()* – столбцы будут располагаться вертикально;
- *barh()* – столбцы будут располагаться горизонтально.

Например, воспользуемся знакомыми вам данными о населении крупнейших городов России. Зададим два массива: *city* – названия городов, *population* – массив, содержащий численность населения в миллионах человек, и передадим их в метод *bar()*.

```
import matplotlib.pyplot as plt
city = ['Москва', 'Санкт-Петербург', 'Новосибирск']
population = [13.02, 5.6, 1.7]
plt.bar(city, population)
plt.show()
```



Обязательными аргументами функции *bar()* являются: *x* – набор величин, определяющий названия столбцов; *height* – числовой массив для определения высоты каждого столбца.

Ознакомимся с некоторыми дополнительными параметрами:

- *width* - массив или скалярная величина, определяющая ширину столбцов, по умолчанию равно 0,8;
- *bottom* - скалярная величина, массив для у координата базовой линии, по умолчанию значение параметра - *optional*;

- *align* {‘center’, ‘edge’, optional} – выравнивание столбцов по горизонтали, по умолчанию установлено значение: ‘center’;
- *color* – параметр, определяющий цвет столбцов диаграммы;
- *edgecolor* – цвет границы столбцов;
- *linewidth* – скалярная величина для ширины границы;
- *tick\_label* – массив значений меток для каждого столбца;
- *alpha* – прозрачность столбцов, принимает значения от 0 до 1, где 0 – абсолютная прозрачность, а 1 – отсутствие прозрачности.

## *Круговая диаграмма*

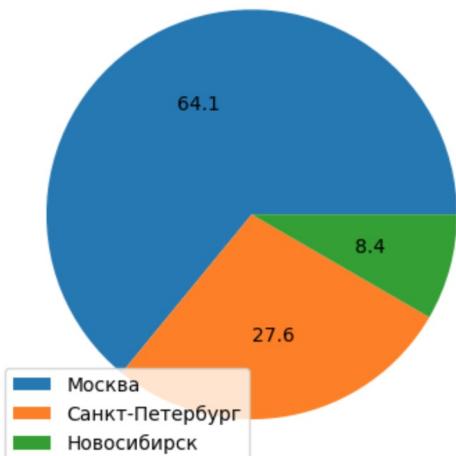
Для построения круговых диаграмм в Matplotlib используется метод **pie()**. Данный метод принимает в качестве обязательных аргументов список значений, и автоматически рассчитывает, какой именно сектор займет каждое отдельное значение. Рассмотрим некоторые другие параметры:

- *labels* {list, optional} – текстовые метки для каждого сектора в определенном порядке, по умолчанию значение *None*;
- *colors* – последовательность цветов секторов;
- *autopct* {str, функция, optional} – формат текстовой метки сектора, значение по умолчанию *None*;
- *labeldistance* {float, *None*, optional} – расстояние от центра диаграммы, на котором будут расположены подписи на секторах, по умолчанию равно 1,1; при значении *None* – метки не будут отображены;
- *radius* – радиус диаграммы;
- *center* – центр диаграммы, значение по умолчанию: (0, 0)
- *frame* – отображение рамки вокруг диаграммы, если параметр равен *True*, в ином случае значение параметра – *False*;
- *wedgeprops* – внешний вид сектора.

Пример построения круговой диаграммы:

```
import matplotlib.pyplot as plt
city = ['Москва', 'Санкт-Петербург', 'Новосибирск']
population = [13.02, 5.6, 1.7]
plt.pie(population, autopct='%.1f')
plt.legend(loc = 'lower left', labels = city)
plt.title("Население крупнейших городов РФ (%)")
plt.show()
```

Население крупнейших городов РФ (%)

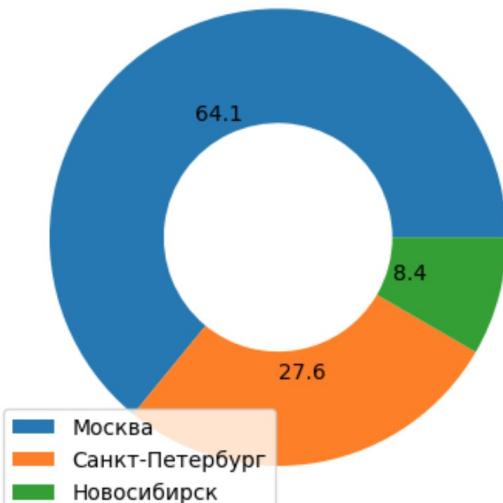


В данном примере также использованы методы `legend()` и `title()` для дополнительного оформления диаграммы. Первый принимает как аргумент массив названий для каждого отдельного сектора и выводит на экран легенду. Местоположение легенды на диаграмме зависит от значения параметра `loc`. Второй метод выводит на экран заголовок диаграммы, расположенный сверху.

В Matplotlib есть также возможность построения круговой диаграммы в виде бублика. Для этого достаточно в методе `plt.pie()` указать значение параметра `wedgeprops`, отвечающего за внешний вид сектора.

```
city = ['Москва', 'Санкт-Петербург', 'Новосибирск']
population = [13.02, 5.6, 1.7]
plt.pie(population, autopct='%.1f', wedgeprops=dict(width=0.5))
plt.legend(loc = 'lower left', labels = city)
plt.title("Население крупнейших городов РФ (%)")
plt.show()
```

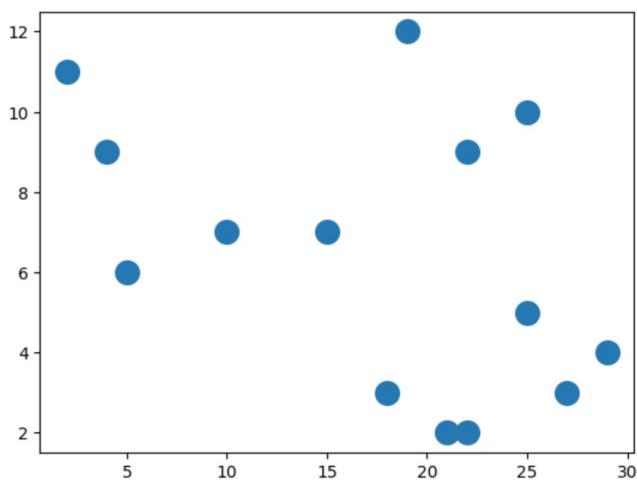
Население крупнейших городов РФ (%)



## Диаграмма рассеяния

Ещё одним видом диаграммы, доступным для построения в Matplotlib, является диаграмма рассеяния. С помощью данной диаграммы удобно визуализировать распределение событий относительно двух измерений: каждой точке соответствует пара значений из набора данных. Для построения диаграммы рассеяния используется метод `plt.scatter()`, в качестве аргументов передаются два массива данных одинаковой размерности.

```
import pandas as pd
import matplotlib.pyplot as plt
#create DataFrame
df = pd.DataFrame({'x': [25, 10, 15, 4, 19, 22, 5, 29, 27, 21, 25, 22, 2, 18],
                    'y': [5, 7, 7, 9, 12, 9, 6, 4, 3, 2, 10, 2, 11, 3]})  
plt.scatter (df.x, df.y, s=200)
```



Аналогично другим методам, этот метод позволяет задавать цвет, тип и размер маркеров через параметры `color`, `marker` и `size` соответственно.

## Задания для самостоятельной работы

1. Построить столбчатую диаграмму, демонстрирующую соотношение диаметров планет Солнечной системы
2. Построить круговую диаграмму, демонстрирующую химический состав земной коры в процентном соотношении.
3. Построить диаграмму на основе данных dataset по вариантам:

*Примечание: для выполнений заданий воспользуйтесь ссылкой<sup>3</sup> на датасет о пассажирах Титаника и ссылкой<sup>4</sup> на датасет о винах.*

*Номер задания совпадает с номером по списку, для двузначных номеров – (N-10), где N – номер по списку*

<sup>3</sup> <https://raw.githubusercontent.com/Grossmend/CSV/master/titanic/data.csv>

<sup>4</sup><https://raw.githubusercontent.com/davestroud/Wine/master/winemag-data-130k-v2.csv>

1. На основе dataset о пассажирах Титаника построить гистограмму, показывающую распределение выживших в зависимости от пола.
2. На основе dataset о пассажирах Титаника построить круговую диаграмму, показывающую распределение выживших в зависимости от класса каюты.
3. На основе dataset о пассажирах Титаника построить гистограмму, показывающую возрастное распределение выживших пассажиров.
4. На основе dataset о пассажирах Титаника построить круговую гистограмму, показывающую распределение выживших в зависимости от класса каюты.
5. На основе dataset о пассажирах Титаника построить гистограмму, показывающую распределение выживших пассажиров 1 класса в зависимости от пола.
6. На основе dataset о пассажирах Титаника построить гистограмму, показывающую распределение выживших пассажиров 3 класса в зависимости от пола.
7. На основе dataset о винах построить гистограмму, показывающую распределение производства вин в зависимости от страны происхождения – 10 самых популярных стран (для определения первых 10 стран можете воспользоваться функцией head()).
8. На основе dataset о винах построить частотную гистограмму, показывающую распределение вин по баллам экспертов.
9. На основе dataset о винах построить круговую диаграмму, показывающую распределение производства вин в зависимости от страны происхождения – 10 самых популярных стран (для определения первых 10 стран можете воспользоваться функцией head()).
10. На основе dataset о винах построить рассеянную диаграмму, показывающую распределение вин по цене и баллам экспертов.

**Индивидуальное задание:**

Построить круговую диаграмму расходов за месяц. Выделить сектор с наибольшими расходами.

## **Список использованной и рекомендуемой литературы**

1. Н. Кайда Самоучитель по Python для начинающих. Часть 24: Основы работы с NumPy. [Электронный ресурс] // Proglib. URL: <https://proglib.io/p/samouchitel-po-python-dlya-nachinayushchih-chast-24-osnovy-raboty-s-numpy-2023-07-10> (дата обращения: 11.02.2024).
2. 100 NumPy задач // Python 3 для начинающих. URL: <https://pythonworld.ru/numpy/100-exercises.html> (дата обращения: 11.02.2024).
3. Нилаб Нисчал Python — это просто. Пошаговое руководство по программированию и анализу данных. — БХВ, 2023. ISBN: 978-5-9775-6849-4.
4. Кристиан Хилл Научное программирование на Python. ДМК:Пресс. - 2021. ISBN: 978-5-97060-914-9
5. Изучаем pandas. Урок 2. Структуры данных Series и DataFrame [Электронный ресурс] // DevPractice. URL: <https://devpractice.ru/pandas-series-and-dataframe-part2/> (дата обращения: 11.02.2024).
6. 120 задач Pandas. Часть 1. Series. URL: [https://grossmend.com/blog/post/pandas\\_120\\_part\\_1/](https://grossmend.com/blog/post/pandas_120_part_1/) (дата обращения: 11.02.2024)
7. Шамаев И. Python 3 Pandas: Объекты Series и DataFrame. Построение Index. URL: <https://python.ivan-shamaev.ru/> (дата обращения: 11.02.2024)
8. Пасхавер Б. Pandas в действии. — СПб.: Питер, 2023. - 512 с.
9. Кристиан Хилл Научное программирование на Python. — ДМК:Пресс, 2021. ISBN: 978-5-97060-914-9
10. Устинов А. Метод Pandas query(): Запрос DataFrame в Python [Электронный ресурс] // DevPractice. URL: <https://dev-gang.ru/article/metod-pandas-query-zapros-dataframe-v-python-7fz3yfomkc/> (дата обращения: 11.02.2024).
11. Robert Johansson Numerical Python: Scientific Computing and Data Science Applications with Numpy, SciPy and Matplotlib. — Apress, 2015. — 503 p.
12. Васильев Ю. Python для Data Science. — СПБ.: Питер, 2023. — (Библиотека программиста). — ISBN:978-5-4461-2392-6.
13. Кеннеди Б. Основы Python для Data Science. — СПб.: Питер, 2023. — (Библиотека программиста). — ISBN: 978-5-4461-2251-6.
14. Силен Д., Мейсман А., Али М. Основы Data Science и Big Data. Python и наука о данных. СПб.: Питер, 2017. — 336 с. — (Библиотека программиста). — ISBN: 9785496025171.
15. D.Y. Chen Pandas for everyone. Python Data Analysis. — Addison-Wesley, 2023.— 1148 p.

**Учебное издание**

**Интеллектуальный анализ данных: numpy, pandas, matplotlib, часть 1:  
учебно-методическое пособие**

**Авторы: Дюличева Юлия Юрьевна, Маршалок Мария Михайловна**

**Редакция авторов**

---

**Подписано к печати Формат 60x84/16. Бумага тип. ОП  
Объем 3,75 п. л. Тираж - 300 Заказ –**

---

**295007, Симферополь, пр. Академика Вернадского, 4 ФГАОУ ВО «КФУ им.  
В. И. Вернадского»**