

## ПРАКТИЧЕСКАЯ РАБОТА №3

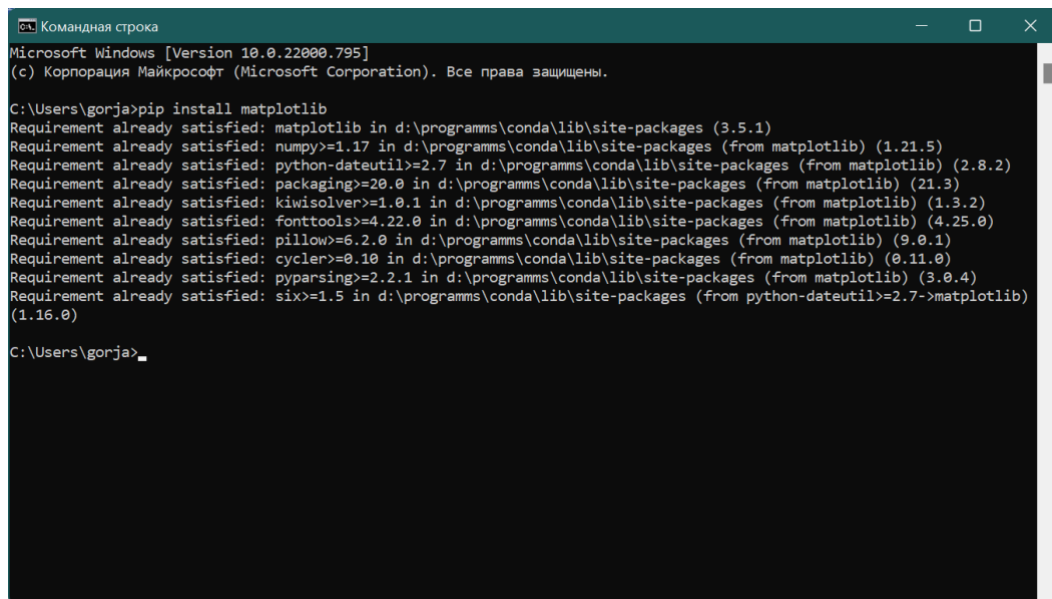
### Визуализация в языке программирования Python

**Цель:** ознакомиться с различными библиотеками визуализации данных (matplotlib, pyplot) и особенностями работы с ними в среде программирования Python.

#### 1. Matplotlib

**Matplotlib** — это обширная библиотека для создания статических, анимированных и интерактивных визуализаций на Python. Она является одной из наиболее популярных библиотек для визуализации данных. С её помощью можно создать любой график, для этого представлен широкий функционал. Но для создания презентабельных графиков зачастую необходимо настраивать несколько параметров.

В случае использования пакета Anaconda Matplotlib уже входит в состав предустановленных библиотек. В ином случае интересующую библиотеку можно установить через менеджер пакетов **pip**, введя в командной строке операционной системой **pip install matplotlib** (ниже представлен пример установки библиотеки. В данном случае библиотека уже была установлена, поэтому представлен такой вывод)



```
Командная строка
Microsoft Windows [Version 10.0.22000.795]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\gorja>pip install matplotlib
Requirement already satisfied: matplotlib in d:\programms\conda\lib\site-packages (3.5.1)
Requirement already satisfied: numpy>=1.17 in d:\programms\conda\lib\site-packages (from matplotlib) (1.21.5)
Requirement already satisfied: python-dateutil>=2.7 in d:\programms\conda\lib\site-packages (from matplotlib) (2.8.2)
Requirement already satisfied: packaging>=20.0 in d:\programms\conda\lib\site-packages (from matplotlib) (21.3)
Requirement already satisfied: kiwisolver>=1.0.1 in d:\programms\conda\lib\site-packages (from matplotlib) (1.3.2)
Requirement already satisfied: fonttools>=4.22.0 in d:\programms\conda\lib\site-packages (from matplotlib) (4.25.0)
Requirement already satisfied: pillow>=6.2.0 in d:\programms\conda\lib\site-packages (from matplotlib) (9.0.1)
Requirement already satisfied: cycler>=0.10 in d:\programms\conda\lib\site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: pyparsing>=2.2.1 in d:\programms\conda\lib\site-packages (from matplotlib) (3.0.4)
Requirement already satisfied: six>=1.5 in d:\programms\conda\lib\site-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)

C:\Users\gorja>
```

Через Jupyter Notebook:

```

In [1]: 1 pip install matplotlib

Requirement already satisfied: matplotlib in d:\programms\conda\lib\site-packages (3.5.1)
Requirement already satisfied: numpy>=1.17 in d:\programms\conda\lib\site-packages (from matplotlib) (1.21.5)
Requirement already satisfied: pyparsing>=2.2.1 in d:\programms\conda\lib\site-packages (from matplotlib) (3.0.4)
Requirement already satisfied: kiwisolver>=1.0.1 in d:\programms\conda\lib\site-packages (from matplotlib) (1.3.2)
Requirement already satisfied: packaging>=20.0 in d:\programms\conda\lib\site-packages (from matplotlib) (21.3)
Requirement already satisfied: fonttools>=4.22.0 in d:\programms\conda\lib\site-packages (from matplotlib) (4.25.0)
Requirement already satisfied: pillow>=6.2.0 in d:\programms\conda\lib\site-packages (from matplotlib) (9.0.1)
Requirement already satisfied: python-dateutil>=2.7 in d:\programms\conda\lib\site-packages (from matplotlib) (2.8.2)
Requirement already satisfied: cycler>=0.10 in d:\programms\conda\lib\site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: six>=1.5 in d:\programms\conda\lib\site-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
Note: you may need to restart the kernel to use updated packages.

In [2]: 1 import matplotlib

In [3]: 1 print(matplotlib.__version__)
3.5.1

```

В Matplotlib график разделён на 3 условных части:

1. Рисунок (Figure) — объект самого верхнего уровня, на котором располагаются холст, область рисования, элементы рисунка
2. Область рисования (Axes) — часть изображения, на котором содержатся данные. Может содержать 2 или 3 координатных оси для отображения двух- и трёхмерных данных
3. Координатная ось (Axis) — определяет область изменения данных, например деления и подписи к делениям

Pyplot – интерфейс для построения графиков простых функций.

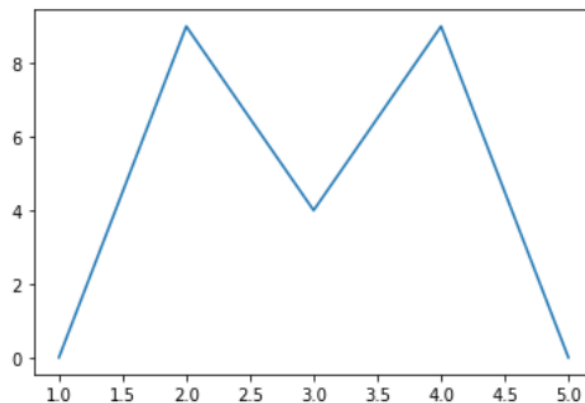
Рисунки в matplotlib создаются путём последовательного вызова команд. Графические элементы (точки, линии, фигуры и т.д.) наслаиваются одна на другую последовательно. При этом последующие перекрывают предыдущие, если они занимают общие участки на рисунке, но их наслаивание можно регулировать методом **zorder**.

Рисунок создается при помощи метода **figure()** – желательно с него начинать создавать какой-либо графический объект. Связано это с тем, что работа происходит с определённой областью рисования, в данном случае мы начинаем работу с самого верхнего уровня. Далее выбирается вид графика и для отображения результата используют **show()**. Ниже представлен пример простого вывода графика

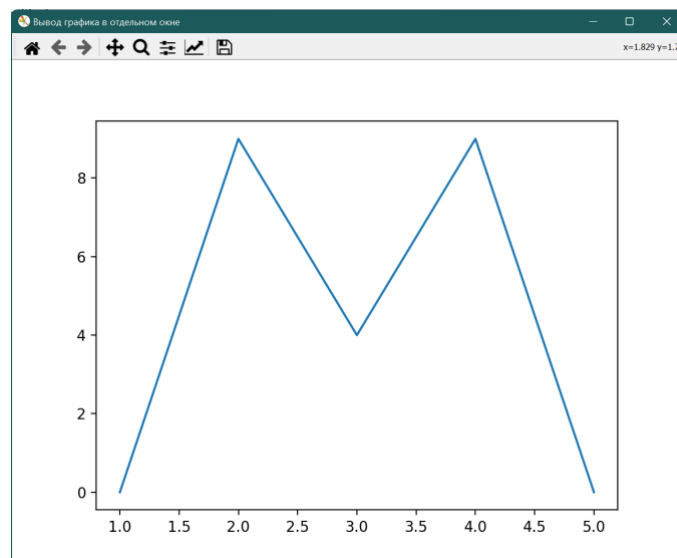
```
#инициализация необходимых библиотек
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
```

```
#инициализация данных для отображения графика
a = np.array([1,2,3,4,5])
b = np.array([0,9,4,9,0])
```

```
plt.figure() #создание объекта Figure
plt.plot(a,b)
plt.show()
```



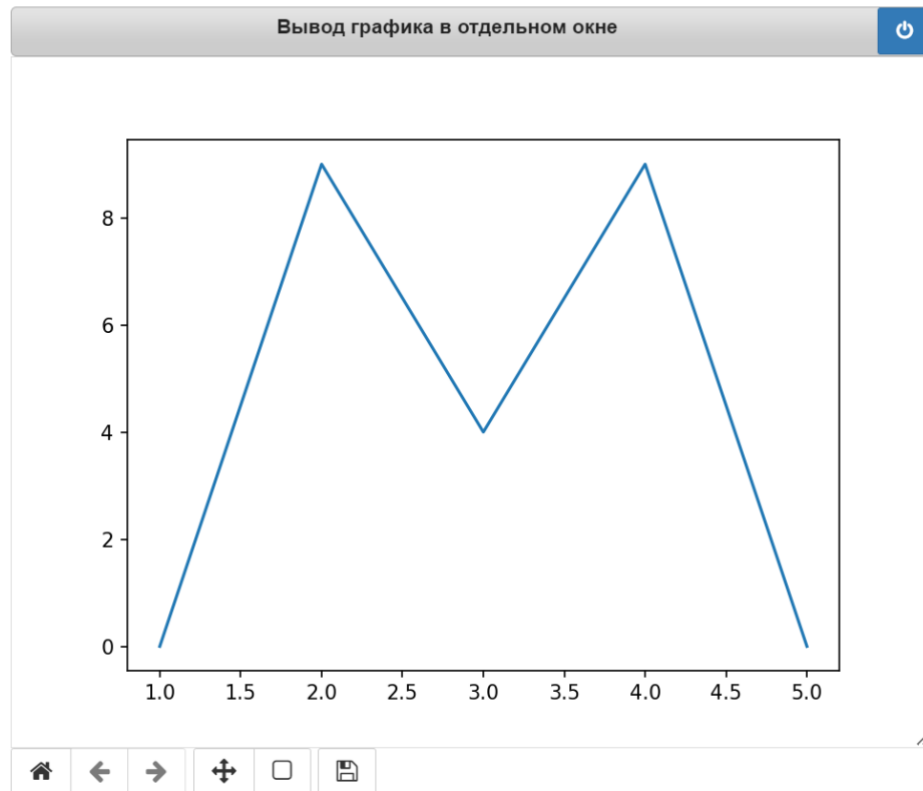
В Jupyter Notebook есть магические команды, которые позволяют выводить не просто рисунок, а интерактивный график. Одна из команд вывода – **%matplotlib**. В результате отобразится интерактивный график в отдельном окне.



Магическую команду **%matplotlib** (и другие) необходимо писать до или после подключения библиотеки и перед отрисовкой графика.

Другим методом отображения интерактивных графиков является **%matplotlib notebook**, в этом случае вывод происходит внутри блокнота.

```
1 %matplotlib notebook
2 fig = plt.figure('Вывод графика в отдельном окне') #создание объекта Figure
3 plt.plot(a,b)
4 plt.show()
```



Загрузим данные для дальнейшей работы с графиками.

*#Загрузка данных из файла*

```
import pandas as pd
data = pd.read_csv('goldprices.csv', sep=';') #загружаем данные, в качестве разделителя используем точку с запятой
data.head()
```

	id	date	price
0	1	1950-02-01	34.73
1	2	1950-03-01	34.73
2	3	1950-04-01	34.73
3	4	1950-05-01	34.73
4	5	1950-06-01	34.73

Графики можно настраивать, для этого есть несколько способов:

1. Задать параметры при создании графика, например:

```

1 %matplotlib notebook
2
3 fig = plt.figure('График с заданными параметрами', figsize=(6,6)) #название и размер внешних границ рисунка
4 plt.grid(True) #включаем сетку у графика
5 plt.title('Динамика цены золота на бирже', fontsize = 15) #название графика и его размер
6 plt.xlabel('t, день', fontsize = 12) # подпись оси x и её размер
7 plt.ylabel('P, цена, $', fontsize = 12) # подпись оси y и её размер
8 plt.xticks(np.arange(start = 0, stop = len(data), step = 100), rotation = 17, size = 10) #изменение шага меток оси X,
9 #а также их поворот и размер
10 plt.yticks(size = 10) #изменение размера для оси Y
11
12 plt.plot(data['date'], #значения для оси X
13          data['price'], #значения для оси Y
14          marker='.', #стиль меток на функции
15          color='crimson', #цвет функции
16          markerfacecolor = 'paleturquoise', #основной цвет метки
17          markeredgecolor = 'black', #цвет контура метки
18          linewidth=2, #толщина линии функции
19          markersize=3) #размер метки функции
20
21 plt.show()

```

График с заданными параметрами



x=1998-04-01 y=1699.

## 2. Задать параметры графика заранее, например:

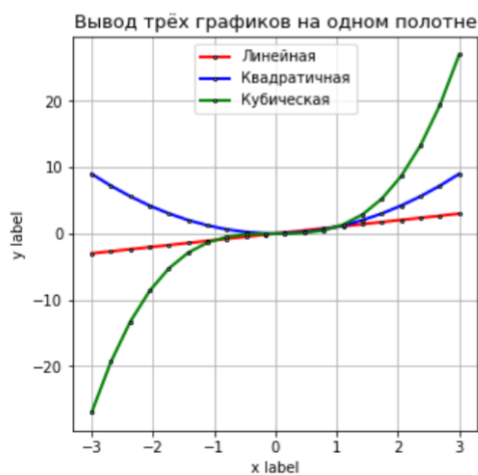
```
1 # определяем параметры для графиков
2 %matplotlib notebook
3
4 #размеры элементов графика
5 big = 13
6 med = 10
7 small = 2
8
9 #параметры, которые будут применяться для всех графиков в Notebook
10 params = {'figure.figsize': (5,5), #размер внешних границ графика
11          'axes.titlesize': big, #размер заголовка графика
12          'axes.labelsize': med, #размер наименований осей
13          'lines.linewidth': small, #толщина линии графика
14          'lines.markersize': small*2, #толщина маркеров графика
15          'xtick.labelsize': med, #размер меток оси абсцисс
16          'ytick.labelsize': med, #размер меток оси ординат
17          'lines.marker': '.', #стиль меток на функции
18          'lines.markerfacecolor': 'paleturquoise', #основной цвет метки
19          'lines.markeredgecolor': 'black', #цвет контура метки
20          'axes.grid': True} # включение сетки
21
22 plt.rcParams.update(params) #применить параметры ко всем графикам
23
24 data = pd.read_excel('energy.xlsx') #считываем данные
25
26 fig = plt.figure('График с заданными параметрами')
27 plt.title('Динамика потребления основной мировой энергии') # название графика
28 plt.xlabel('t, год') # подпись оси x
29 plt.ylabel('E, Эксаджоули') # подпись оси y
30 plt.plot(data['Year'], data['Consumption'], color = 'crimson') #создание графика, определение данных для осей
31 plt.show() #вывод графика
```



В таком случае эти параметры будут применены ко всем графикам, которые вы будете создавать далее. Больше настраиваемых параметров вы можете найти на официальном сайте [Matplotlib](https://matplotlib.org/).

Для вывода нескольких функций на одном полотне достаточно задать значения для каждой из них.

```
1 %matplotlib inline
2 x = np.linspace(-3, 3, 20) # Создаём массив данных со значениями, равно распределёнными на всём интервале.
3 fig = plt.figure('3 графика на одном холсте', figsize = (5,5))
4 plt.plot(x, x, label='Линейная') #Первая функция
5 plt.plot(x, x**2, label='Квадратичная') # Вторая функция
6 plt.plot(x, x**3, label='Кубическая') #Третья функция
7 plt.xlabel('x label')
8 plt.ylabel('y label')
9 plt.title("Вывод трёх графиков на одном полотне")
10 plt.legend(loc='upper center')
11
12 plt.show()
```

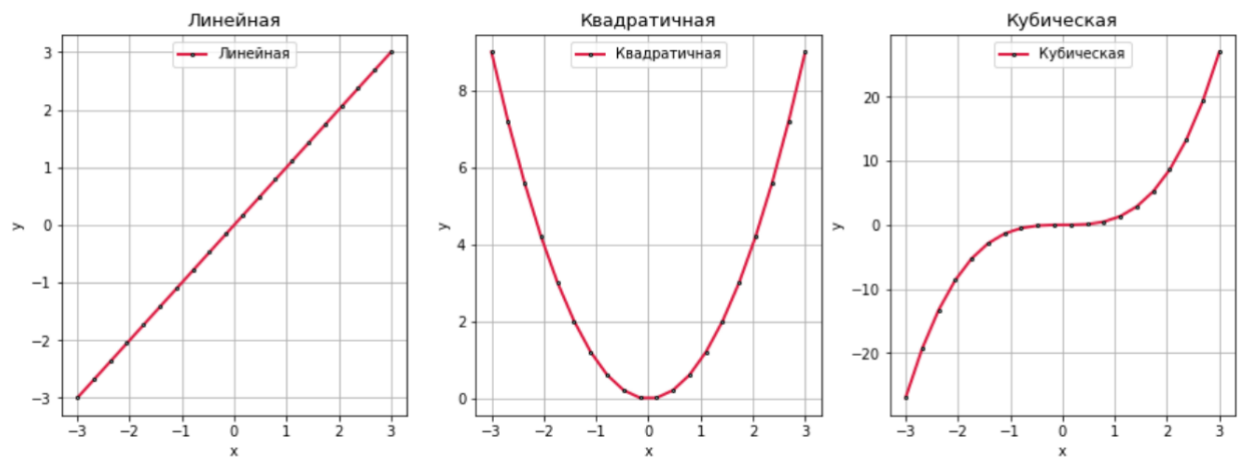


Для вывода нескольких функций в отдельных ячейках необходимо дописать команду **subplots**:

```

1 x = np.linspace(-3, 3, 20) # Создаём массив данных со значениями, равно распределёнными на всём интервале.
2 fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize = (15,5))
3
4 ax1.plot(x, x, label='Линейная') #Первая функция
5 ax1.set_xlabel('x')
6 ax1.set_ylabel('y')
7 ax1.set_title("Линейная")
8 ax1.legend(loc = 'upper center')
9 ax2.plot(x, x**2, label='Квадратичная') # Вторая функция
10 ax2.set_xlabel('x')
11 ax2.set_ylabel('y')
12 ax2.set_title("Квадратичная")
13 ax2.legend(loc = 'upper center')
14 ax3.plot(x, x**3, label='Кубическая') #Третья функция
15 ax3.set_xlabel('x')
16 ax3.set_ylabel('y')
17 ax3.set_title("Кубическая")
18 ax3.legend(loc = 'upper center')
19
20 plt.show()

```



В скобках, в строчке под номером 2 указанные переменные образуют кортеж, к которому далее идёт обращение для настройки каждого из отдельных графиков, также можно указать одну переменную и обращаться к графику при помощи индекса.

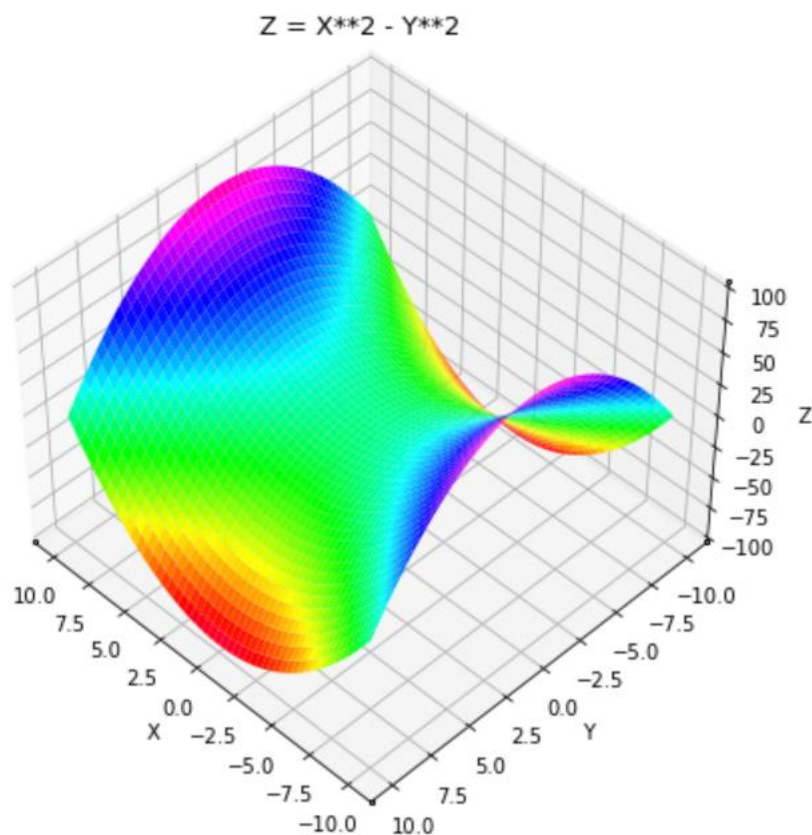
Также можно создавать 3d графики, указав дополнительную ось.



```

1 fig = plt.figure(figsize=(7,7))
2 axes = fig.add_subplot(projection='3d')
3
4 X = np.arange(-10, 10, 0.05)
5 Y = np.arange(-10, 10, 0.05)
6 X, Y = np.meshgrid(X, Y)
7 Z = X**2 - Y**2
8
9 surf = axes.plot_surface(X, Y, Z, cmap = 'gist_rainbow')
10 axes.set_xlabel ('X')
11 axes.set_ylabel ('Y')
12 axes.set_zlabel ('Z')
13 axes.set_title ('Z = X**2 - Y**2')
14 axes.view_init(45, 135) #повернуть график
15 plt.show()
16

```

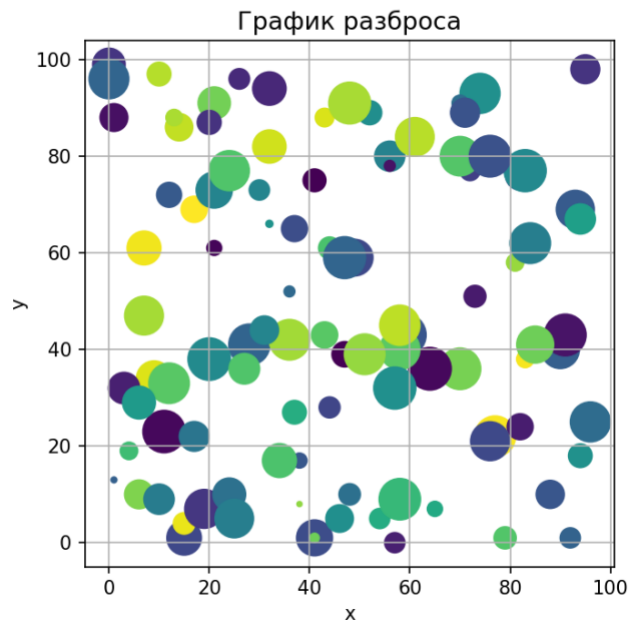


Другим распространённым видом графика является **scatter()**. Как правило, эта диаграмма требует незначительного изменения внешнего вида. Однако иногда видимость данных сильно ограничена из-за небольшого размера диаграммы, размеров и цветов используемых маркеров.

```

1 np.random.seed(2718281828) #выбранный seed, необходим, чтобы случайный результат не изменялся при новых запусках.
2 fig = plt.figure('График разброса', figsize = (5,5))
3 x = np.random.randint(0,100,100)
4 y = np.random.randint(0,100,100)
5 s = np.random.randint(0,500,100)
6 c = np.random.randint(0,500,100)
7
8 plt.title('График разброса')
9 plt.xlabel('x')
10 plt.ylabel('y')
11 plt.scatter(x,y,s,c)
12 plt.show()

```



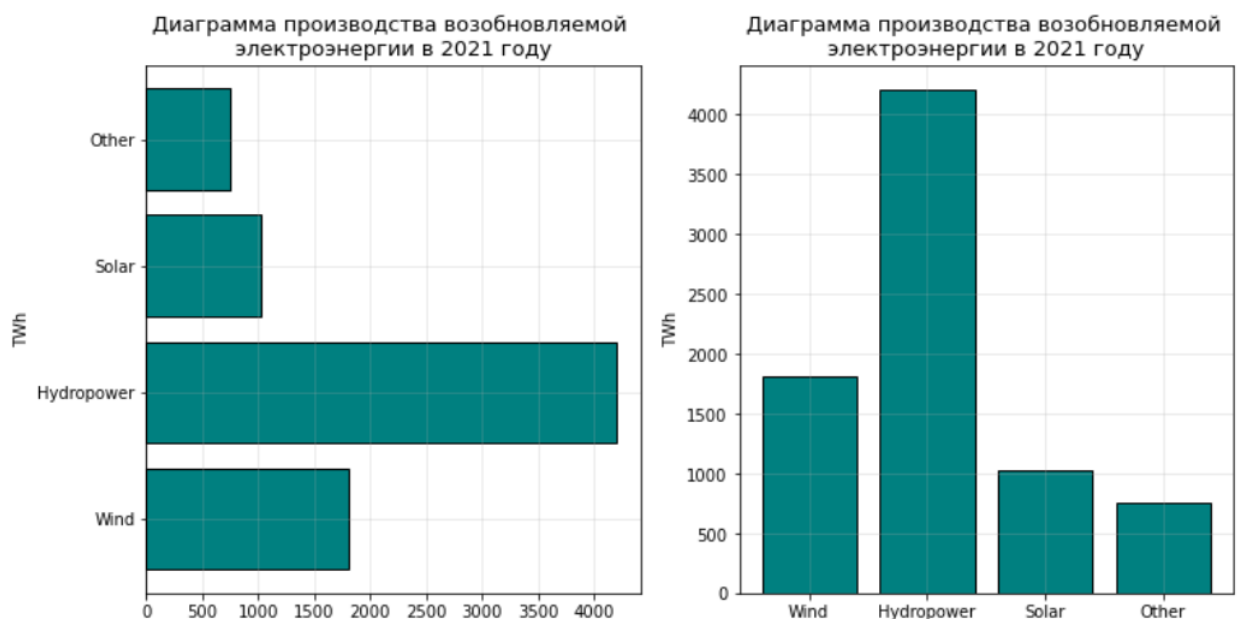
В данном случае график является четырёхмерным. Первые две размерности определяются осями, третья — размером окружностей, а четвёртая — цветом.

Ещё один часто используемая визуализация — столбчатые диаграммы **bar()**. В основном они используются для отображения категориальных данных. Простой пример отображения вертикальной и горизонтальной диаграммы.

```

1 fig, ax = plt.subplots(1,2,figsize=(12,6))
2 energy = [1813.70,4206.14,1023.10,749.99]
3 name = ['Wind','Hydropower','Solar','Other']
4
5 ax[1].bar(name,energy, color = 'teal', edgecolor = 'black')
6 ax[1].grid(alpha = 0.3, zorder = 1)
7 ax[1].set_title('Диаграмма производства возобновляемой \пэлектроэнергии в 2021 году')
8 ax[1].set_ylabel('TWh')
9
10 ax[0].barh(name,energy, color = 'teal', edgecolor = 'black')
11 ax[0].grid(alpha = 0.3, zorder = 1)
12 ax[0].set_title('Диаграмма производства возобновляемой \пэлектроэнергии в 2021 году')
13 ax[0].set_ylabel('TWh')
14
15
16 plt.show()

```



## 2. Plotly

Это библиотека с открытым исходным кодом, которую можно использовать для визуализации данных. Plotly поддерживает различные типы графиков и вы с большой вероятностью получите оптимальный для восприятия результат, не настраивая дополнительно параметры. Также компания разрабатывает и предоставляет библиотеки научного построения графиков для Arduino, Julia, MATLAB, Perl, Python, R и REST, что позволяет работать и в других языках программирования.

Перед началом работы необходимо установить библиотеку командой **!pip install plotly (!conda install plotly)**, так как она не входит в стандартный пакет. После этого вы можете убедиться, что пакет установился корректно.

```
1 !conda install plotly
```

```
Collecting plotly
  Downloading plotly-5.9.0-py2.py3-none-any.whl (15.2 MB)
Requirement already satisfied: tenacity>=6.2.0 in d:\programms\conda\lib\site-packages (from plotly) (8.0.1)
Installing collected packages: plotly
Successfully installed plotly-5.9.0
Collecting package metadata (current_repodata.json): ...working... done
Solving environment: ...working... done

# All requested packages already installed.
```

```
1 import plotly
2 print (plotly.__version__)
```

```
5.9.0
```

Для работы с Plotly понадобятся и дополнительные модули, входящие в библиотеку, поэтому извлечём их и будем обращаться к ним через другие переменные для удобства работы.

**import plotly.graph\_objs as go** — содержит объекты (рисунок, макет, данные и графики, такие как: диаграмма рассеяния, линейный график и т.д.), которые отвечают за создание графиков.

**import plotly.express as px** — модуль `plotly.express` может создать весь рисунок за один раз. Он автоматически использует `graph_objects` и возвращает `graph_objects`.

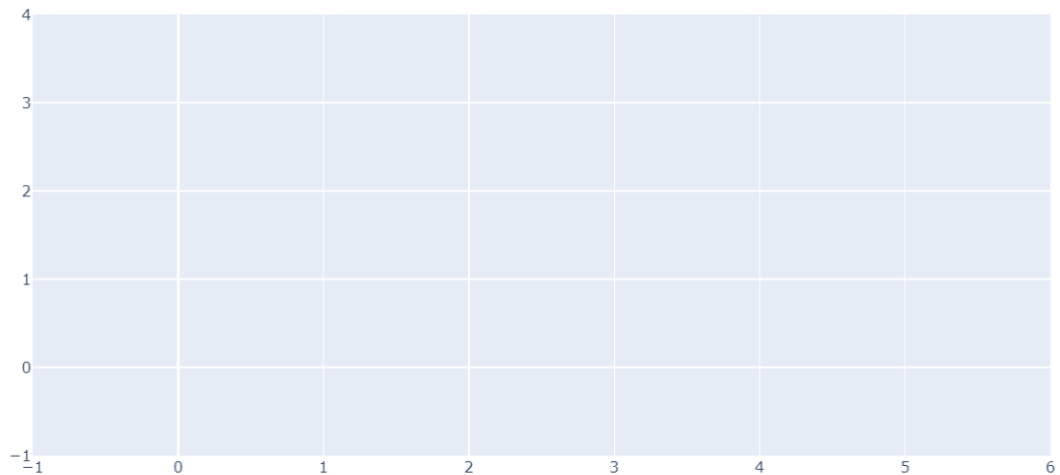
Попробуем вывести рабочую область

```
1 #как выглядит полотно в matplotlib
2 fig = plt.figure()
3 plt.show()
```

<Figure size 432x288 with 0 Axes>

```
1 import plotly.graph_objs as go
2 import plotly.express as px
```

```
1 fig = go.Figure() #также создаём полотно в Plotly
2 fig.show()
```

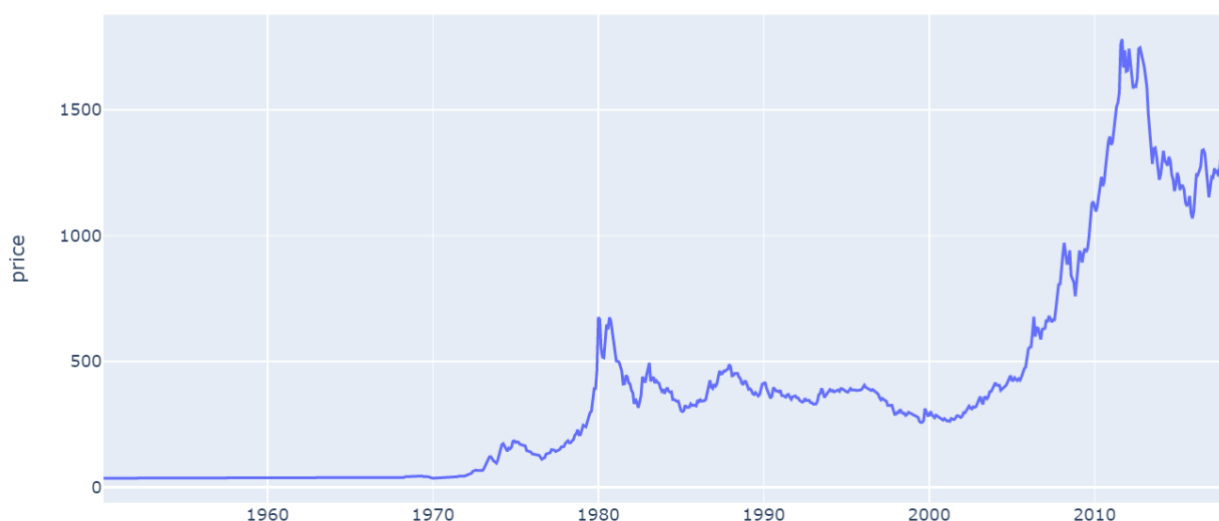


Как мы видим, здесь уже есть отличие с `matplotlib`. В первом случае создаётся просто полотно без осей. В `Plotly` сразу создаётся рабочая плоскость с двумя осями.

Попробуем отобразить предыдущий график. В случае, если необходимо быстро отрисовать график, и проанализировать показатели, можно воспользоваться экспресс-модулем **`plotly.express`**, который самостоятельно создаёт графические объекты. Необходимо указать тип графика, а в скобках передать его параметры. В частности, к какому объекту обратиться, чтобы взять данные. В нашем случае это `DataFrame`. И указать набор данных для абсциссы и ординаты, которые будут использоваться как подписи данных для осей и в маркерах, при наведении на точки графика.

```
1 fig = px.line(data, x="date", y="price", title = "Динамика цены золота на бирже")
2 fig.show()
```

Динамика цены золота на бирже



В Plotly есть инструмент наведения. В данном случае отображается интерактивный график, с которым вы сразу можете взаимодействовать. При наведении курсора на функцию, отображаются координаты каждой из точек.

Но в экспресс-модуле Plotly может не хватать функционала для работы с графиками, например, при создании соосных объектов, фасеточных графиков с разными типами данных, дашбордов (создание нескольких графиков на одном графическом окне), также с его помощью нельзя создать 3D фигуры. Поэтому попробуем отобразить самый простой график рассеяния, используя `graph_objs`. Посмотрим, как в нём сопоставляются данные:

```
1 print(go.Scatter(x=data['date'], y=data['price'])) #при помощи этой строки
2 #выбираются данные для осей и сопоставляются друг с другом
```

```
Scatter({
  'x': array(['1950-02-01', '1950-03-01', '1950-04-01', ..., '2017-12-01',
             '2018-01-01', '2018-02-01'], dtype=object),
  'y': array([ 34.73 ,  34.73 ,  34.73 , ..., 1265.674, 1332.809, 1333.775])
})
```

Мы видим, что данная структура напоминает словарь в языке Python. То есть фигуры в Plotly могут быть представлены в виде словаря, как и другие настройки. Но обычно используются словари не в чистом виде, а используются специальные графические объекты (`graph_objects`), которые

применяются для представления фигур и имеют преимущества перед словарями, например, обеспечивается проверка на ошибку. При неверном введении имени свойства или его значения, будет вызвано исключение, понятное пользователю, а также поддерживаются вспомогательные функции более высокого уровня для тонкой настройки уже построенных фигур.

Теперь добавим команду для добавления графического объекта к фигуре (**add\_trace**) и команду отображения графика (**fig.show()**).

```
1 fig = go.Figure()
2 '''Минимум для вывода графика'''
3 print(go.Scatter(x=data['date'], y=data['price'])) #при помощи этой строчки выбираются данные для осей и сопоставляются
4 #друг с другом
5 fig.add_trace(go.Scatter(x=data['date'], y=data['price'])) #совместив предыдущую строчку и команду за скобками,
6 #будет отображён линейный график
7 fig.show()

Scatter({
  'x': array(['1950-02-01', '1950-03-01', '1950-04-01', ..., '2017-12-01',
             '2018-01-01', '2018-02-01'], dtype=object),
  'y': array([ 34.73,  34.73,  34.73, ..., 1265.674, 1332.809, 1333.775])
})
```



Обновим макет графика, используя дополнительные параметры. На скриншоте ниже в 8 строчке вводится команда изменения макета, а в скобках указываются название графика, расположение названия (по оси y и x, а также его команды для его выравнивания по центру сверху с размером текста, равным 20). На 11 и 12 строках указываются названия и размер текста для осей x и y. На 13 строке размеры рисунка (в данном случае высота равна 500 пикселей, а ширина выбирается автоматически, занимая ширину рабочей зоны Jupyter). На 14 строчке кода изменяются начальные отступы рисунка

слева, справа, сверху и снизу соответственно, так как изначально в Plotly имеются отступы, которые снижают размеры графика, особенно сильно он виден сверху (рисунок предыдущий).

```
1 fig = go.Figure()
2 '''Минимум для вывода графика'''
3 #print(go.Scatter(x=data['date'], y=data['price'])) #при помощи этой строчки выбираются данные для осей и сопоставляются
4 #друг с другом
5 fig.add_trace(go.Scatter(x=data['date'], y=data['price'])) #совместив предыдущую строчку и команду за скобками,
6 #будет отображён линейный график
7
8 '''Дополнительные параметры для вывода графика'''
9 fig.update_layout(title = 'Динамика цены золота на бирже', #вводим заголовок для графика
10                   title_y = 0.96, title_x = 0.55, title_xanchor = 'center', title_yanchor = 'top', title_font_size = 20,
11                   xaxis_title="t, день", xaxis_title_font_size = 16,
12                   yaxis_title="P, цена, $", yaxis_title_font_size = 16,
13                   width=None, height=500,
14                   margin=dict(l=0, r=0, t=40, b=0))
15 fig.show()
```

Отобразим получившийся результат.



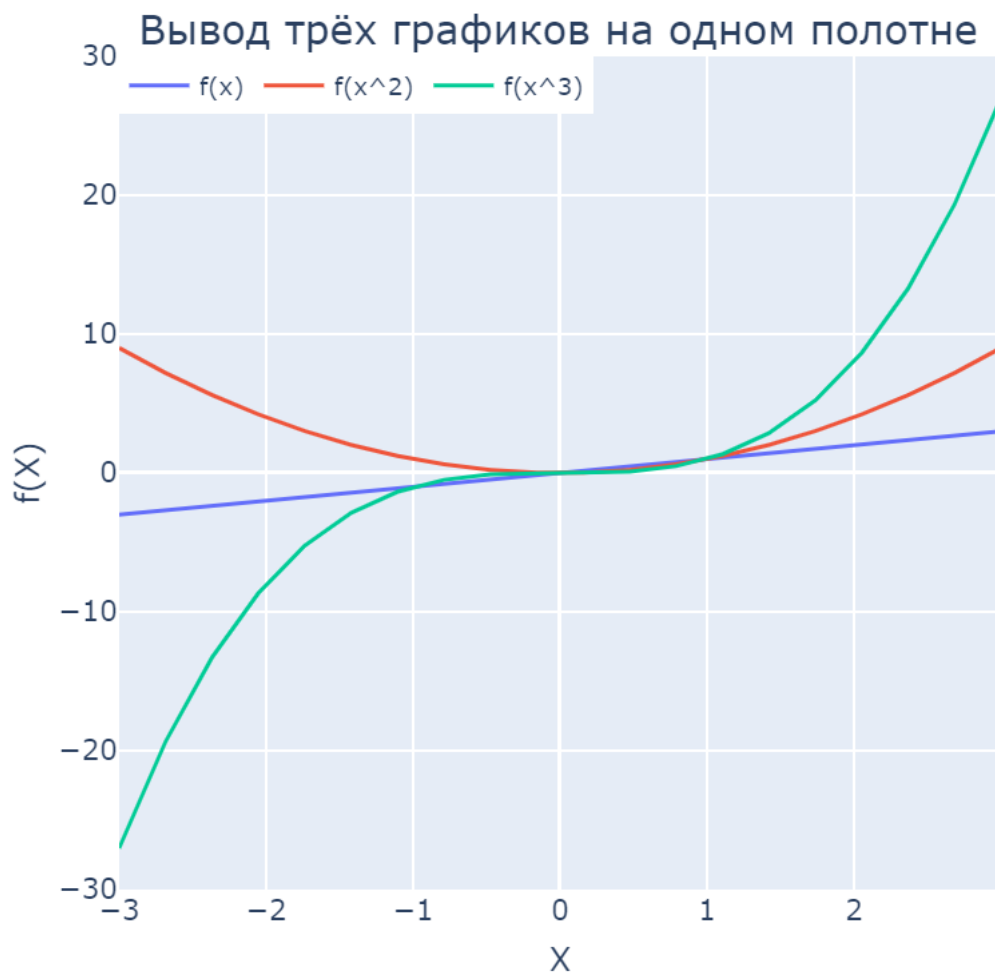
Теперь отобразим 3 графика на одном рисунке. Ниже представлен код. На скриншоте видно, что параметры для заголовка, осей и другие можно задавать через словарь указывая какой объект вы хотите изменить (заголовок, оси и т.д.) и параметры для этого объекта (название, размер текста, положение и т.д.). Для отображения дополнительных функций на одном рисунке достаточно добавлять trace с соответствующими массивами для осей.



```

1 x = np.linspace(-3, 3, 20) # Создаём массив данных со значениями равно распределёнными на всём интервале.
2
3 fig = go.Figure()
4 fig.update_layout(title={'text': 'Вывод трёх графиков на одном полотне', 'font_size': 20,
5                           'y':0.96, 'x':0.55, 'xanchor': 'center', 'yanchor': 'top'},
6
7                     xaxis = dict(title='X', title_font_size = 16, tickfont_size = 14),
8                     yaxis = dict(title = 'f(X)', title_font_size = 16, tickfont_size = 14),
9
10                    width=500, height=500,
11                    margin=dict(l=0, r=0, t=40, b=0),
12                    legend = dict(orientation="h", xanchor="left", yanchor="top", y=1, x=0))
13
14
15 fig.add_trace(go.Scatter(x=x, y=x, name = 'f(x)'))
16 fig.add_trace(go.Scatter(x=x, y=(x**2), name = 'f(x^2)'))
17 fig.add_trace(go.Scatter(x=x, y=(x**3), name = 'f(x^3)'))
18
19 fig.show()

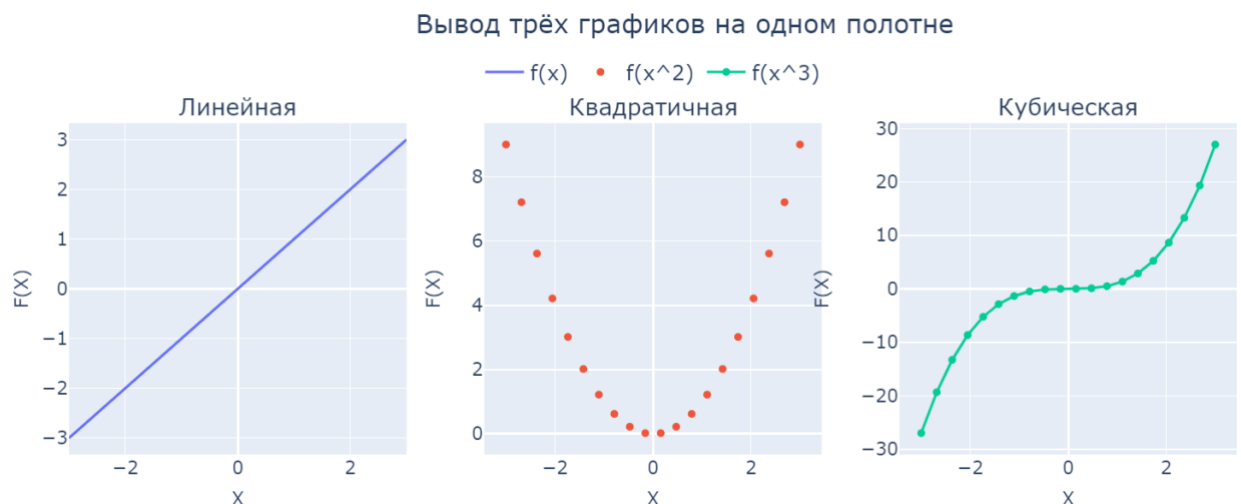
```



Разделим графики. Аналогично с matplotlib создаём subplots. Созданные ячейки можно объединять, если необходимо изменить их размеры. Для каждого графика можно индивидуально настроить оси, что показано на примере изменения настроек оси X (выбирая строку и колонку ячейки). Или применить настройки ко всем ячейкам (изменение параметров оси Y). Метод `add_trace` также принимает номер строки и столбца, для отображения графика

в определённой ячейке с данными, передаваемыми в оси, и соответствующими настройками, например для изменения типа линий.

```
1 from plotly.subplots import make_subplots
2
3
4 fig = go.Figure(make_subplots(rows=1, cols=3, subplot_titles=('Линейная', 'Квадратичная', 'Кубическая')))
5 fig.update_annotations(font_size=18)
6
7 fig.update_layout(title={'text': 'Вывод трёх графиков на одном полотне', 'font_size': 20,
8                          'y':0.96, 'x':0.55, 'xanchor': 'center', 'yanchor': 'top'},
9
10                    width=None, height=400,
11                    margin=dict(l=0, r=0, t=100, b=0),
12                    legend = dict(orientation="h", xanchor="center", yanchor="top", y=1.22, x=0.5), font_size = 16)
13
14 # Изменяем настройки оси X
15 fig.update_xaxes(title_text='X', title_font_size = 14, tickfont_size = 14, row=1, col=1)
16 fig.update_xaxes(title_text='X', title_font_size = 14, tickfont_size = 14, row=1, col=2)
17 fig.update_xaxes(title_text='X', title_font_size = 14, tickfont_size = 14, row=1, col=3)
18
19 # Изменяем настройки оси Y
20 fig.update_yaxes(title_text='F(X)', title_font_size = 14, tickfont_size = 14)
21 # fig.update_yaxes(title_text='F(X)', title_font_size = 14, tickfont_size = 14, row=1, col=2)
22 # fig.update_yaxes(title_text='F(X)', title_font_size = 14, tickfont_size = 14, row=1, col=3)
23
24 fig.add_trace(go.Scatter(x=x, y=x, name = 'f(x)', mode='lines'), row = 1 , col = 1)
25 fig.add_trace(go.Scatter(x=x, y=(x**2), name = 'f(x^2)', mode='markers'), row = 1 , col = 2)
26 fig.add_trace(go.Scatter(x=x, y=(x**3), name = 'f(x^3)', mode='lines+markers'), row = 1 , col = 3)
```



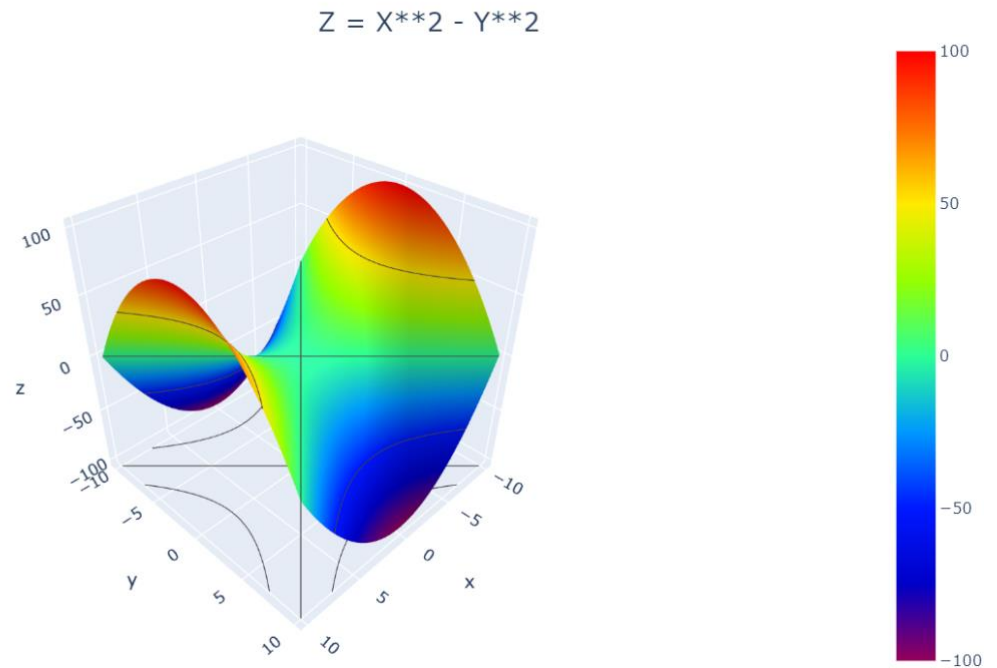
Если необходимо создать трёхмерный график рассеяния, то достаточно обратиться к методу **px.scatter\_3d** и передать в него параметры x, y и z.

Ниже представлен пример кода для создания графика поверхности через go.

```

1 X = np.arange(-10, 10, 0.05)
2 Y = np.arange(-10, 10, 0.05)
3 X, Y = np.meshgrid(X, Y)
4 Z = X**2 - Y**2
5
6 fig = go.Figure(data=[go.Surface(x=X, y=Y, z=Z, colorscale='rainbow')])
7 fig.update_traces(contours_z=dict(show=True, usecolormap=False, highlightcolor="limegreen", project_z=True))
8 fig.update_layout(title=dict(text='Z = X**2 - Y**2', 'font_size': 20,
9                               'y':0.96, 'x':0.55, 'xanchor': 'center', 'yanchor': 'top'),
10                  margin={'l':0, 'r':0, 't':40, 'b':0})
11 fig.show()

```



Далее представлен код для создания столбчатой диаграммы в Plotly. В экспресс-модуле также можно задать параметры для отображения значений столбцов и значение цвета в зависимости от переданных значений.

```

1 energy = [1813.70,4206.14,1023.10,749.99]
2 name = ['Wind','Hydropower','Solar','Other']
3
4
5 fig = go.Figure(px.bar(x=name, y=energy, color = name, text= energy))
6 fig.update_traces(textfont_size=14, textangle=0, textposition="outside", marker=dict(line=dict(color='black', width=2)))
7 fig.update_layout(
8     title = 'Диаграмма производства возобновляемой электроэнергии в 2021 году', title_font_size = 20,
9     xaxis_title = 'Вид электроэнергии', xaxis_title_font_size = 18, xaxis_tickfont_size = 16,
10    yaxis_title = 'TWh', yaxis_title_font_size = 18, yaxis_tickfont_size = 16,
11    margin=dict(l=0, r=0, t=30, b=0))
12 fig.show()

```



К дополнительным возможностям данной библиотеки стоит отнести возможность размещать интерактивные графики на сайте, с которыми можно взаимодействовать. А также создавать анимации, размещать кнопки, ползунки (слайдеры), всплывающие списки и работать с различными типами визуализации данных. На [сайте Plotly](#) представлены примеры всех графиков и всего функционала, предоставляющего возможности тонкой настройки для создания дашбордов.

## Задание

1. Найти и выгрузить многомерные данные с использованием библиотеки **pandas**. В отчёте описать найденные данные.
2. Вывести информацию о данных при помощи методов **.info()**, **.head()**. Проверить данные на наличие пустых значений. В случае их наличия удалить данные строки или интерполировать пропущенные значения. При необходимости дополнительно предобработать данные для дальнейшей работы с ними.
3. Построить столбчатую диаграмму (**.bar**) с использованием модуля **graph\_objs** из библиотеки **Plotly** со следующими параметрами:
  - 3.1. По оси X указать дату или название, по оси Y указать количественный показатель.
  - 3.2. Сделать так, чтобы столбец принимал цвет в зависимости от значения показателя (`marker=dict(color=признак, coloraxis="coloraxis")`).
  - 3.3. Сделать так, чтобы границы каждого столбца были выделены чёрной линией с толщиной равной 2.
  - 3.4. Отобразить заголовок диаграммы, разместив его по центру сверху, с 20 размером текста.
  - 3.5. Добавить подписи для осей X и Y с размером текста, равным 16. Для оси абсцисс развернуть метки так, чтобы они читались под углом, равным 315.
  - 3.6. Размер текста меток осей сделать равным 14.
  - 3.7. Расположить график во всю ширину рабочей области и присвоить высоту, равную 700 пикселей.
  - 3.8. Убрать лишние отступы по краям.
4. Построить круговую диаграмму (**go.Pie**), используя данные и стиль оформления из предыдущего графика. Сделать так, чтобы границы каждой доли были выделены чёрной линией с толщиной, равной 2.
5. Построить линейный график накопленных значений количественного показателя.

- 5.1. Сделать график с линиями и маркерами, цвет линии 'crimson', цвет точек 'white', цвет границ точек 'black', толщина границ точек равна 2.
- 5.2. Добавить сетку на график, сделать её цвет 'ivory' и толщину равную 2. (Можно сделать это при настройке осей с помощью `gridwidth=2`, `gridcolor='ivory'`).
6. Постараться создать аналогичные графики с использованием библиотеки `matplotlib`.
7. На основе проделанной работы составить отчёт с описанием и скриншотами полученных результатов, сделать выводы о выбранном организационном процессе на основе полученных графиков, сравнить библиотеки.