Практическое задание 2

Задание 1

Загрузите модуль pyplot библиотеки matplotlib с псевдонимом plt, а также библиотеку numpy с псевдонимом np. Примените магическую функцию %matplotlib inline для отображения графиков в Jupyter Notebook и настройки конфигурации ноутбука со значением 'svg' для более четкого отображения графиков. Создайте список под названием x с числами 1, 2, 3, 4, 5, 6, 7 и список y с числами 3.5, 3.8, 4.2, 4.5, 5, 5.5, 7. С помощью функции plot постройте график, соединяющий линиями точки с горизонтальными координатами из списка x и вертикальными - из списка y. Затем в следующей ячейке постройте диаграмму рассеяния (другие названия - диаграмма разброса, scatter plot).

In [1]:

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

In [2]:

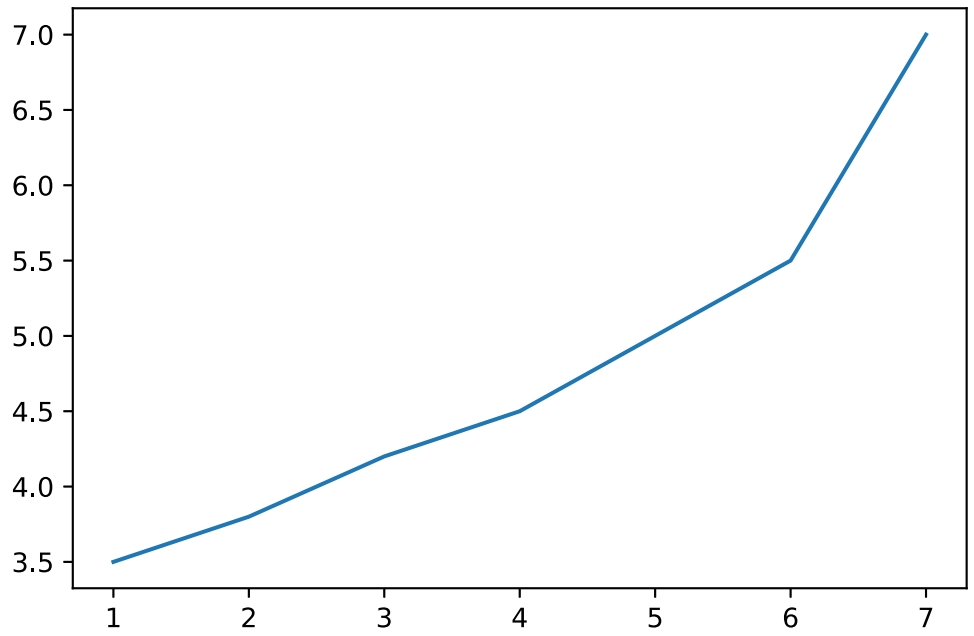
```
%matplotlib inline
%config InlineBackend.figure_format = 'svg'
```

In [3]:

```
x = [1, 2, 3, 4, 5, 6, 7]
y = [3.5, 3.8, 4.2, 4.5, 5, 5.5, 7]
```

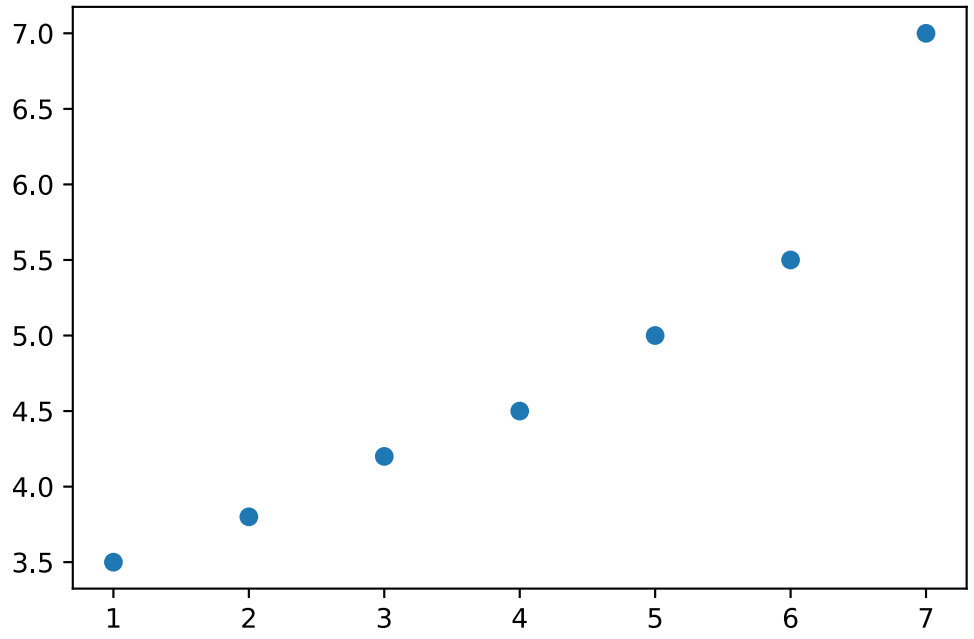
In [4]:

```
plt.plot(x, y)
plt.show()
```



In [5]:

```
plt.scatter(x, y)
plt.show()
```



Задание 2

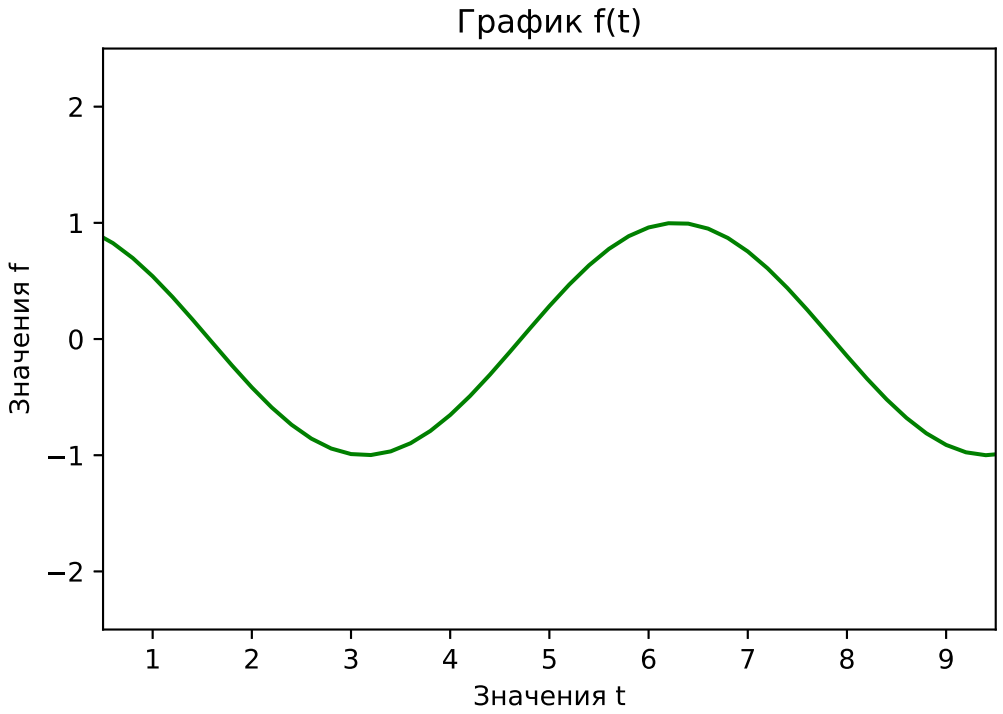
С помощью функции linspace из библиотеки Numpy создайте массив t из 51 числа от 0 до 10 включительно. Создайте массив Numpy под названием f, содержащий косинусы элементов массива t. Постройте линейную диаграмму, используя массив t для координат по горизонтали,а массив f - для координат по вертикали. Линия графика должна быть зеленого цвета. Выведите название диаграммы - 'График f(t)'. Также добавьте названия для горизонтальной оси - 'Значения t' и для вертикальной - 'Значения f'. Ограничьте график по оси x значениями 0.5 и 9.5, а по оси y - значениями -2.5 и 2.5.

In [6]:

```
t = np.linspace(0, 10, 51)
f = np.cos(t)
```

In [7]:

```
plt.plot(t, f, color="green")

plt.title("График f(t)")
plt.xlabel("Значения t")
plt.ylabel("Значения f")

plt.axis([0.5, 9.5, -2.5, 2.5])

plt.show()
```



***Задание 3**

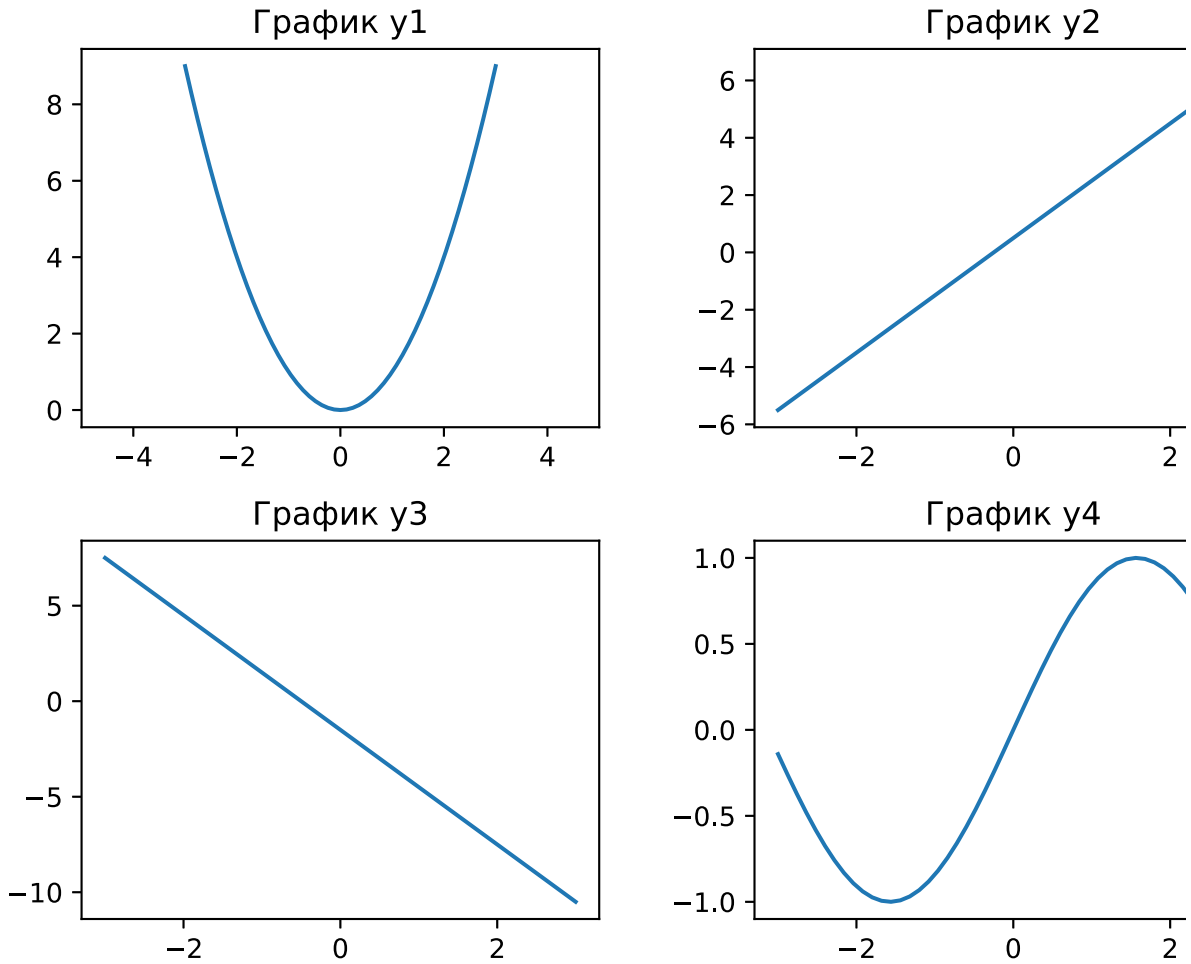
С помощью функции linspace библиотеки Numpy создайте массив x из 51 числа от -3 до 3 включительно. Создайте массивы y1, y2, y3, y4 по следующим формулам: y1 = x**2 y2 = 2 * x + 0.5 y3 = -3 * x - 1.5 y4 = sin(x) Используя функцию subplots модуля matplotlib.pyplot, создайте объект matplotlib.figure.Figure с названием fig и массив объектов Axes под названием ax,причем так, чтобы у вас было 4 отдельных графика в сетке, состоящей из двух строк и двух столбцов. В каждом графике массив x используется для координат по горизонтали.В левом верхнем графике для координат по вертикали используйте y1,в правом верхнем - y2, в левом нижнем - y3, в правом нижнем - y4.Дайте название графикам: 'График y1', 'График y2' и т.д. Для графика в левом верхнем углу установите границы по оси x от -5 до 5. Установите размеры фигуры 8 дюймов по горизонтали и 6 дюймов по вертикали. Вертикальные и горизонтальные зазоры между графиками должны составлять 0.3.

In [8]:

```
x = np.linspace(-3, 3, 51)
y1 = x**2
y2 = 2 * x + 0.5
y3 = -3 * x - 1.5
y4 = np.sin(x)
```

In [9]:

```
fig, ax = plt.subplots(nrows=2, ncols=2)
ax1, ax2, ax3, ax4 = ax.flatten()
fig.subplots_adjust(wspace=0.3, hspace=0.3)
fig.set_size_inches(8, 6)

ax1.plot(x, y1)
ax1.set_title("График y1")
ax1.set_xlim([-5, 5])

ax2.plot(x, y2)
ax2.set_title("График y2")

ax3.plot(x, y3)
ax3.set_title("График y3")

ax4.plot(x, y4)
ax4.set_title("График y4")

plt.show()
```



***Задание 4**

В этом задании мы будем работать с датасетом, в котором приведены данные по мошенничеству с кредитными данными: Credit Card Fraud Detection (информация об авторах: Andrea Dal Pozzolo, Olivier Caelen, Reid A. Johnson and Gianluca Bontempi. Calibrating Probability with Undersampling for Unbalanced Classification. In Symposium on Computational Intelligence and Data Mining (CIDM), IEEE, 2015).

Ознакомьтесь с описанием и скачайте датасет creditcard.csv с сайта Kaggle.com по ссылке: https://www.kaggle.com/mlg-ulb/creditcardfraud (https://www.kaggle.com/mlg-ulb/creditcardfraud) Данный датасет является примером несбалансированных данных, так как мошеннические операции с картами встречаются реже обычных. Импортруйте библиотеку Pandas, а также используйте для графиков стиль “fivethirtyeight”. Посчитайте с помощью метода value_counts количество наблюдений для каждого значения целевой переменной Class и примените к полученным данным метод plot, чтобы построить столбчатую диаграмму. Затем постройте такую же диаграмму, используя логарифмический масштаб. На следующем графике постройте две гистограммы по значениям признака V1 - одну для мошеннических транзакций (Class равен 1) и другую - для обычных (Class равен 0). Подберите значение аргумента density так, чтобы по вертикали графика было расположено не число наблюдений, а плотность распределения. Число бинов должно равняться 20 для обеих гистограмм, а коэффициент alpha сделайте равным 0.5, чтобы гистограммы были полупрозрачными и не загораживали друг друга. Создайте легенду с двумя значениями: “Class 0” и “Class 1”. Гистограмма обычных транзакций должна быть серого цвета, а мошеннических - красного. Горизонтальной оси дайте название “V1”.

In [10]:

```
import pandas as pd
plt.style.use('fivethirtyeight')
```

In [11]:

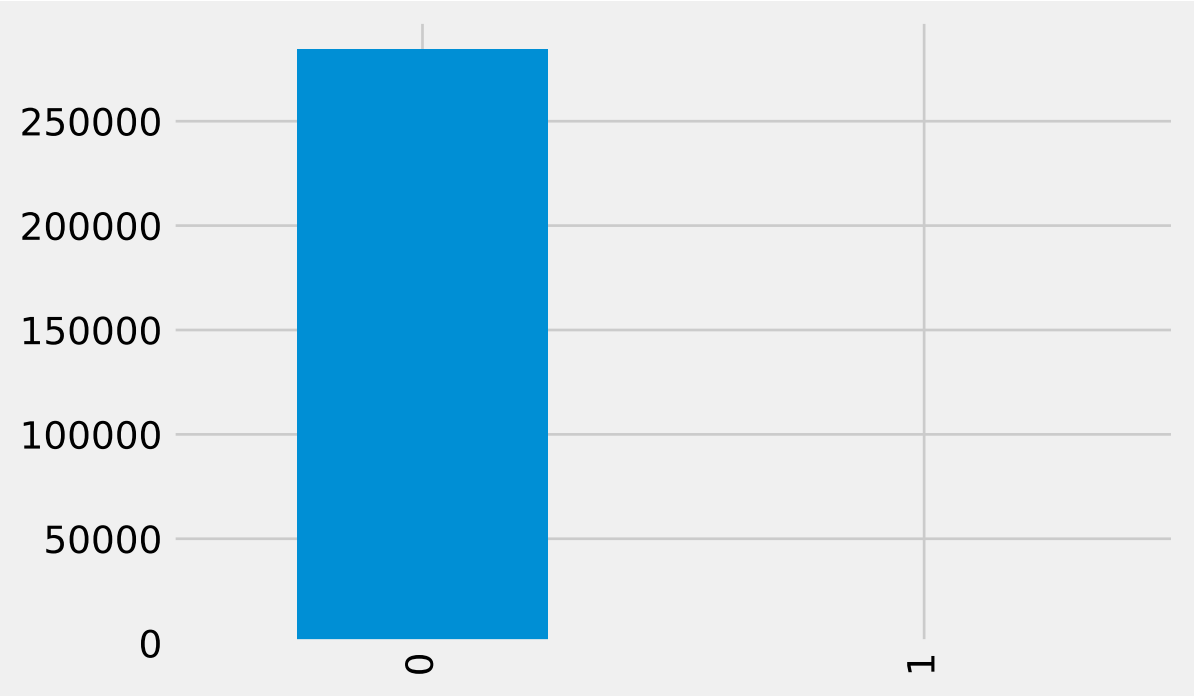
```
creditcard = pd.read_csv('creditcard.csv', sep=',', encoding='ANSI')
value_counts = creditcard["Class"].value_counts()
value_counts
```

Out[11]:

```
0    284315
1       492
Name: Class, dtype: int64
```

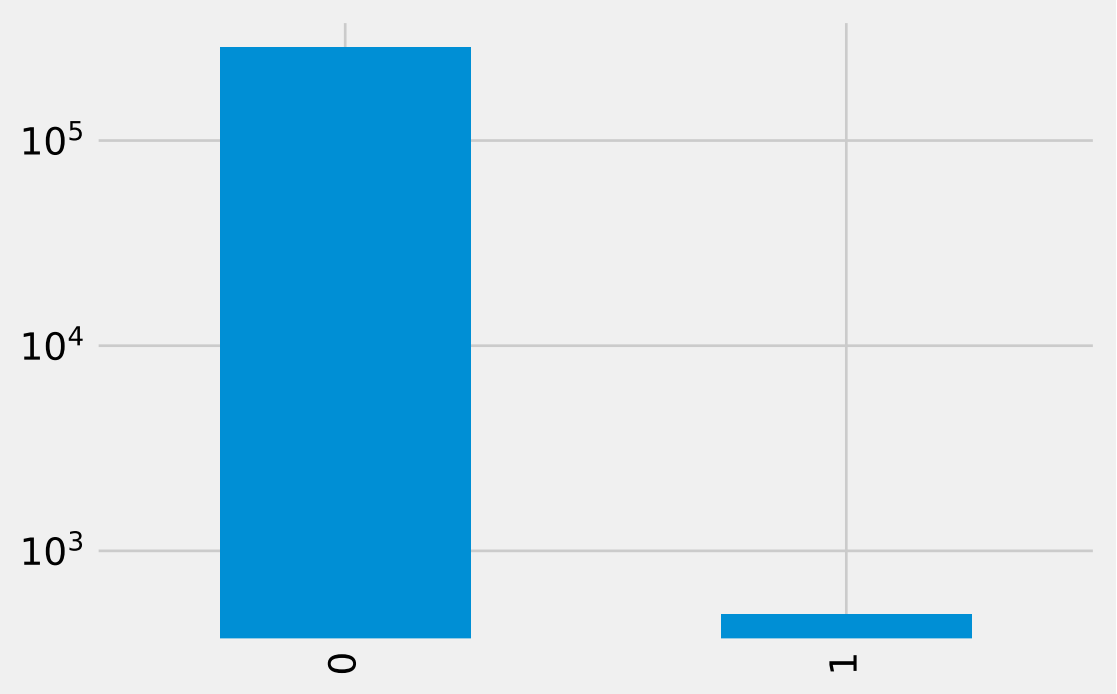
In [12]:

```
# Столбчадтая диаграмма по целевой переменой "Class" в обычном масштабе
value_counts.plot(kind = "bar")
plt.show()
```



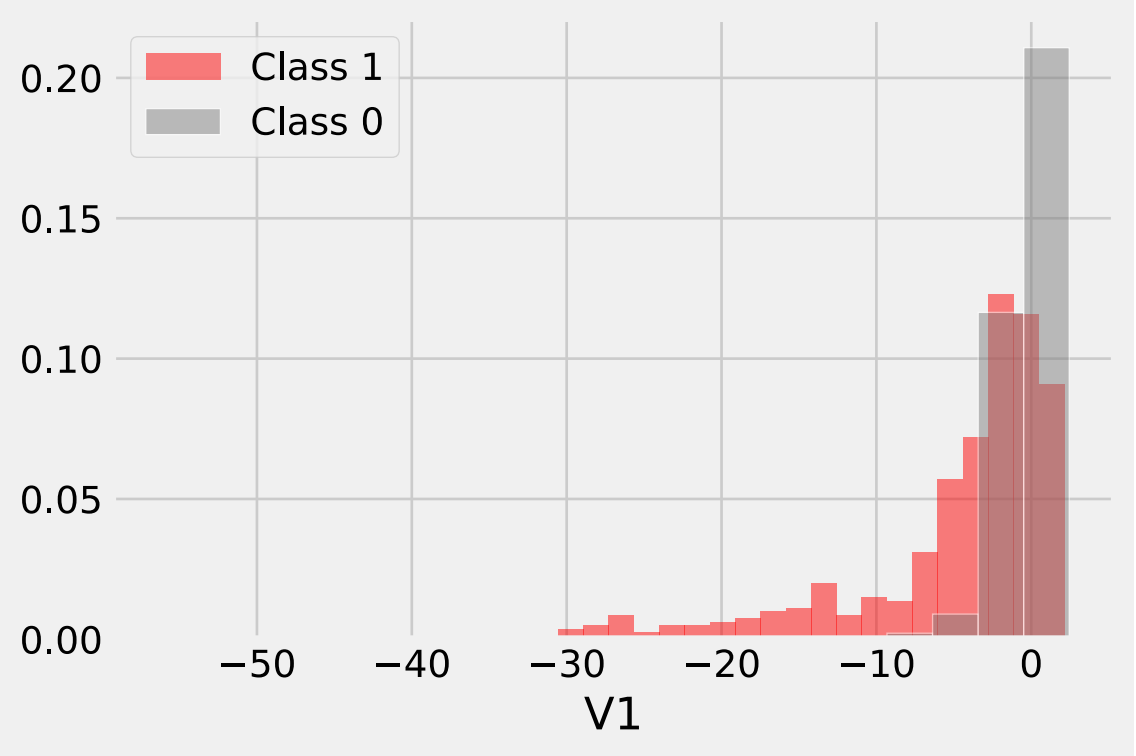
In [13]:

```
# Столбчадтая диаграмма по целевой переменой "Class" в логарифмическом масштабе
value_counts.plot(kind = "bar", logy=True)
plt.show()
```



In [14]:

```
# Гистограммы со значениям признака V1 - одна для мошеннических транзакций (Class равен
plt.hist(creditcard[creditcard["Class"] == 1]["V1"], bins = 20, density = True, alpha =
plt.hist(creditcard[creditcard["Class"] == 0]["V1"], bins = 20, density = True, alpha =
plt.xlabel("V1")
plt.legend()
plt.show()
```



**Задание на повторение материала

1 Создать одномерный массив Numpy под названием a из 12 последовательных целых чисел чисел от 12 до 24 невключительно

In [24]:

```
a = np.arange(12, 24)
a
```

Out[24]:

```
array([12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23])
```

2 Создать 5 двумерных массивов разной формы из массива a. Не использовать в аргументах метода reshape число -1.

In [40]:

```
1 a.reshape(2, 6)
```

Out[40]:

```
array([[12, 13, 14, 15, 16, 17],
       [18, 19, 20, 21, 22, 23]])
```

In [41]:

```
1 a.reshape(3, 4)
```

Out[41]:

```
array([[12, 13, 14, 15],
       [16, 17, 18, 19],
       [20, 21, 22, 23]])
```

In [42]:

```
1 a.reshape(4, 3)
```

Out[42]:

```
array([[12, 13, 14],
       [15, 16, 17],
       [18, 19, 20],
       [21, 22, 23]])
```

In [43]:

```
1 a.reshape(6, 2)
```

Out[43]:

```
array([[12, 13],
       [14, 15],
       [16, 17],
       [18, 19],
       [20, 21],
       [22, 23]])
```

In [44]:

```
1 a.reshape(12, 1)
```

Out[44]:

```
array([[12],
       [13],
       [14],
       [15],
       [16],
       [17],
       [18],
       [19],
       [20],
       [21],
       [22],
       [23]])
```

3 Создать 5 двумерных массивов разной формы из массива a. Использовать в аргументах метода reshape число -1 (в трех примерах - для обозначения числа столбцов, в двух - для строк).

In [46]:

```
1 a.reshape(2, -1)
```

Out[46]:

```
array([[12, 13, 14, 15, 16, 17],
       [18, 19, 20, 21, 22, 23]])
```

In [47]:

```
1 a.reshape(3, -1)
```

Out[47]:

```
array([[12, 13, 14, 15],
       [16, 17, 18, 19],
       [20, 21, 22, 23]])
```

In [48]:

```
1 a.reshape(4, -1)
```

Out[48]:

```
array([[12, 13, 14],
       [15, 16, 17],
       [18, 19, 20],
       [21, 22, 23]])
```

In [51]:

```
1 a.reshape(-1, 1)
```

Out[51]:

```
array([[12],
       [13],
       [14],
       [15],
       [16],
       [17],
       [18],
       [19],
       [20],
       [21],
       [22],
       [23]])
```

In [52]:

```
a.reshape(-1, 2)
```

Out[52]:

```
array([[12, 13],
       [14, 15],
       [16, 17],
       [18, 19],
       [20, 21],
       [22, 23]])
```

4 Можно ли массив Numpy, состоящий из одного столбца и 12 строк, назвать одномерным?

Нет. Это легко проверить, используя функцию $ndim$

In [53]:

```
a.reshape(12, 1).ndim
```

Out[53]:

```
2
```

Видим, что массив Numpy, состоящий из одного столбца и 12 строк является двумерным

5 Создать массив из 3 строк и 4 столбцов, состоящий из случайных чисел с плавающей запятой из нормального распределения со средним, равным 0 и среднеквадратичным отклонением, равным 1.0. Получить из этого массива одномерный массив с таким же атрибутом size, как и исходный массив.

In [56]:

```
a = np.random.randn(3, 4)
a
```

Out[56]:

```
array([[-1.1313837 ,  1.01297629,  0.88021151,  1.69299203],
       [-0.9128315 , -0.2622431 , -1.12290829, -1.48033181],
       [-0.05135252,  1.27869712, -0.01742394,  0.47727848]])
```

In [59]:

```
a.flatten()
```

Out[59]:

```
array([-1.1313837 ,  1.01297629,  0.88021151,  1.69299203, -0.9128315 ,
       -0.2622431 , -1.12290829, -1.48033181, -0.05135252,  1.27869712,
       -0.01742394,  0.47727848])
```

In [62]:

```
a.size == a.flatten().size
```

Out[62]:

```
True
```

Видим, что коичество элементов в данных массивах совпадает

6 Создать массив a, состоящий из целых чисел, убывающих от 20 до 0 невключительно с интервалом 2.

In [77]:

```
a = np.arange(20, 0, -2)
a
```

Out[77]:

```
array([20, 18, 16, 14, 12, 10,  8,  6,  4,  2])
```

7 Создать массив b, состоящий из 1 строки и 10 столбцов: целых чисел, убывающих от 20 до 1 невключительно с интервалом 2. В чем разница между массивами a и b?

In [80]:

```
b = np.arange(20, 1, -2).reshape(1, 10)
b
```

Out[80]:

```
array([[20, 18, 16, 14, 12, 10,  8,  6,  4,  2]])
```

In [84]:

```
a.ndim
```

Out[84]:

```
1
```

In [85]:

```
b.ndim
```

Out[85]:

```
2
```

Разница между массивами a и b заключатся в том, что a - это одномерный массив, а b - двумерный.

8 Вертикально соединить массивы a и b. a - двумерный массив из нулей, число строк которого больше 1 и на 1 меньше, чем число строк двумерного массива b, состоящего из единиц. Итоговый массив v должен иметь атрибут size, равный 10.

In [94]:

```
a = np.zeros((2, 2))
a
```

Out[94]:

```
array([[0., 0.],
       [0., 0.]])
```

In [95]:

```
b = np.ones((3, 2))
b
```

Out[95]:

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

In [97]:

```
np.concatenate((a, b), axis=0)
```

Out[97]:

```
array([[0., 0.],
       [0., 0.],
       [1., 1.],
       [1., 1.],
       [1., 1.]])
```

In [98]:

```
np.concatenate((a, b), axis=0).size
```

Out[98]:

```
10
```

9 Создать одномерный массив a, состоящий из последовательности целых чисел от 0 до 12. Поменять форму этого массива, чтобы получилась матрица A (двумерный массив Numpy), состоящая из 4 строк и 3 столбцов. Получить матрицу At путем транспонирования матрицы A. Получить матрицу B, умножив матрицу A на матрицу At с помощью матричного умножения. Какой размер имеет матрица B? Получится ли вычислить обратную матрицу для матрицы B и почему?

0 является целым числом. О том, что 12 не включается в последовательность не сказано.
Следовательно массив a, состоящий из последовательности целых чисел от 0 до 12 состоит из 13 чисел, и как следствие из него нельзя получить матрицу, состоящую из 4 строк и 3 столбцов.
Однако если предположить, что 12 не включается в исходный массbв, тогда решение задания будет следующим:

In [104]:

```
a = np.arange(0, 12)
a
```

Out[104]:

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

In [106]:

```
A = a.reshape(4, 3)
A
```

Out[106]:

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

In [108]:

```
At = A.T
At
```

Out[108]:

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

In [110]:

```
B = A@At
B
```

Out[110]:

```
array([[  5,  14,  23,  32],
       [ 14,  50,  86, 122],
       [ 23,  86, 149, 212],
       [ 32, 122, 212, 302]])
```

In [113]:

```
print(f"Размера матрицы B равен {B.shape}")
```

```
Размера матрицы B равен (4, 4)
```

In [116]:

```
print(f"Определитель матрицы B равен {np.linalg.det(B)}")
```

```
Определитель матрицы B равен 0.0
```

Поскольку обратную матрицу можно посчитать только для матрицы, определитель которой не равен 0, то вычислить обратную матрицу для матрицы B невозможно.

10 Инициализируйте генератор случайных числе с помощью объекта seed, равного 42.

In [168]:

```
import random
random.seed(42)
```

11 Создайте одномерный массив c, составленный из последовательности 16-ти случайных равномерно распределенных целых чисел от 0 до 16 невключительно.

In [175]:

```
c = np.random.randint(0, 16, (16))
c
```

Out[175]:

```
array([ 8,  7,  0, 11,  7,  7, 14, 10,  2,  0,  7,  2,  2,  0, 10,  4])
```

12 Поменяйте его форму так, чтобы получилась квадратная матрица C. Получите матрицу D, поэлементно прибавив матрицу B из предыдущего вопроса к матрице C, умноженной на 10. Вычислите определитель, ранг и обратную матрицу D_inv для D.

In [177]:

```
C = c.reshape(4, 4)
C
```

Out[177]:

```
array([[ 8,  7,  0, 11],
       [ 7,  7, 14, 10],
       [ 2,  0,  7,  2],
       [ 2,  0, 10,  4]])
```

In [181]:

```
D = C*10 + B
D
```

Out[181]:

```
array([[ 85,  84,  23, 142],
       [ 84, 120, 226, 222],
       [ 43,  86, 219, 232],
       [ 52, 122, 312, 342]])
```

In [182]:

```
# Определитель матрицы D
np.linalg.det(D)
```

Out[182]:

```
8266999.999999998
```

In [183]:

```
# Ранг матрицы D
np.linalg.matrix_rank(D)
```

Out[183]:

```
4
```

In [213]:

```
# Обратная матрица для матрицы D
D_inv = np.linalg.inv(D)
D_inv
```

Out[213]:

```
array([[ 0.00926334, -0.02441998,  0.21224386, -0.1319729 ],
       [-0.00777791,  0.05333253, -0.28304101,  0.16061449],
       [-0.006307  ,  0.00215798,  0.03773316, -0.02437886],
       [ 0.00711987, -0.01728075,  0.03427362, -0.01206484]])
```

13 Приравняйте к нулю отрицательные числа в матрице D_inv, а положительные - к единице. Убедитесь, что в матрице D_inv остались только нули и единицы. С помощью функции numpy.where, используя матрицу D_inv в качестве маски, а матрицы B и C - в качестве источников данных, получите матрицу E размером 4x4. Элементы матрицы E, для которых соответствующий элемент матрицы D_inv равен 1, должны быть равны соответствующему элементу матрицы B, а элементы матрицы E, для которых соответствующий элемент матрицы D_inv равен 0, должны быть равны соответствующему элементу матрицы C.

In [214]:

```
D_inv = np.where(D_inv < 0 , 0, 1)
D_inv
```

Out[214]:

```
array([[1, 0, 1, 0],
       [0, 1, 0, 1],
       [0, 1, 1, 0],
       [1, 0, 1, 0]])
```

In [220]:

```
E = np.where(D_inv == 1 , B, C)
E
```

Out[220]:

```
array([[  5,   7,  23,  11],
       [  7,  50,  14, 122],
       [  2,  86, 149,   2],
       [ 32,   0, 212,   4]])
```

In [ ]:

```

```

In [ ]:

```

```

In [215]:

```
1 B
```

Out[215]:

```
array([[  5,  14,  23,  32],
       [ 14,  50,  86, 122],
       [ 23,  86, 149, 212],
       [ 32, 122, 212, 302]])
```

In [219]:

```
1 C
```

Out[219]:

```
array([[ 8,  7,  0, 11],
       [ 7,  7, 14, 10],
       [ 2,  0,  7,  2],
       [ 2,  0, 10,  4]])
```

In [ ]:

```
1
```