

ФИО: Шагина Екатерина Владимировна БЭК211

Задача 1

In [6]:

```
import pandas as pd
import numpy as np
import scipy.stats as stats
from math import sqrt
import matplotlib as plt
from scipy.stats import expon
```

пункт а)

Вероятность встречи нового таксиста на 1-й день:

$$P_1 = 1$$

\ На 2-й день:

$$P_2 = 1 * \frac{n-1}{n}$$

\ На 3-й день:

$$P_3 = 1 * \frac{n-1}{n} * \frac{n-2}{n}$$

\ ... \ На 10-й день:

$$P_n = \prod_{k=0}^8 \frac{n-k}{n} * \frac{9}{n}$$

Функцией правдоподобия для данной задачи будет вероятность того, что на первых 9 заказах приедут только уже приезжавшие таксисты, а на 10-ом заказе приедет новый таксист, т.е.:

$$L(n) = P(10\text{-ый заказ - новый таксист}) = \prod_{k=0}^8 \frac{n-k}{n} \cdot \frac{9}{n}$$

In [78]:

```
def Likelihood(n):

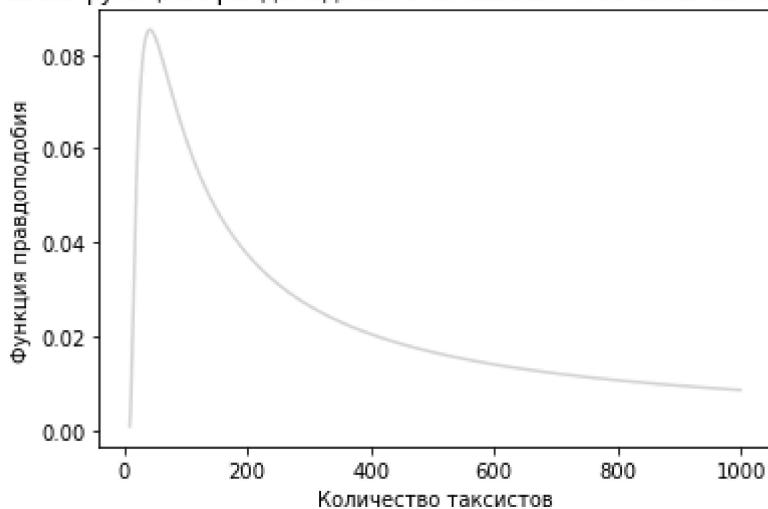
    p = 1
    for k in range(2,11):
        p = p*(n-k+2)/n
    return p*(9)/n

likelihood_values = [Likelihood(n) for n in range(9, 1000)]
n_values = np.arange(9, 1000)

plt.plot(n_values, likelihood_values, color = 'pink')
plt.xlabel("Количество таксистов")
```

```
plt.ylabel("Функция правдоподобия")
plt.title('Значение функции правдоподобия в зависимости от количества таксистов')
plt.show()
```

Значение функции правдоподобия в зависимости от количества таксистов



In [73]:

```
max_index = np.argmax(likelihood_values)
maximum_likelihood = likelihood_values[max_index]
estimated_n = n_values[max_index]
print(f'Оценка числа n методом максимального правдоподобия: {estimated_n}')
```

Оценка числа n методом максимального правдоподобия: 42

пункт 6)

In [80]:

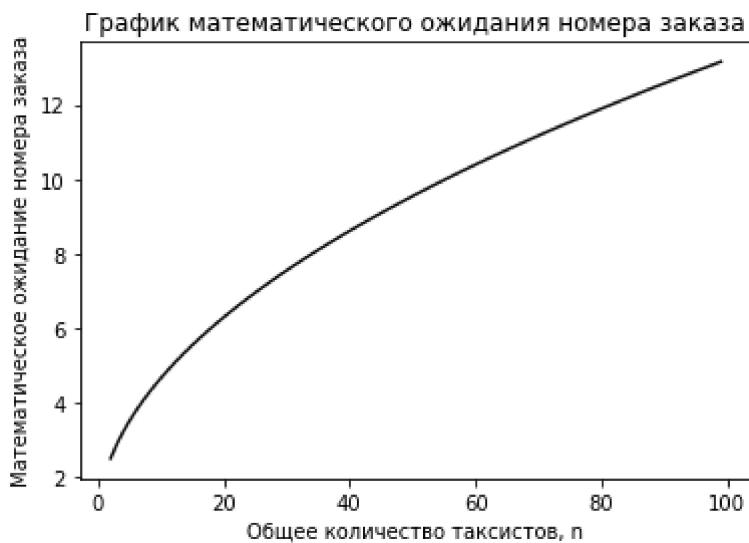
```
def prob(k, n):
    p = 1
    for i in range(2, k+1):
        p = p * (n - i + 2) / n
    return p * (k - 1) / n

def expected(n):
    c = 0
    for k in range(2, n+2):
        c = c + k * prob(k, n)
    return c

n_values = np.arange(2, 100)
E = [expected(n) for n in n_values]

plt.plot(n_values, E, color='navy')

plt.xlabel('Общее количество таксистов, n')
plt.ylabel('Математическое ожидание номера заказа')
plt.title('График математического ожидания номера заказа')
plt.show()
```



В этом коде функция `prob(k, n)` вычисляет вероятность того, что повторный приезд произойдет на номере заказа k , при условии, что общее количество таксистов равно n . Функция `expected(n)` вычисляет математическое ожидание номера заказа, на котором происходит первый повторный приезд, при общем количестве таксистов n . Затем мы используем массив `n_values` для генерации значений общего количества таксистов и вычисляем соответствующие значения математического ожидания E .

пункт в)

Задача 2

пункт а)

In [94]:

```
import itertools

def calculate_likelihood(k, n, l: int = 10):

    probability = 1
    a = 0

    for i in range(1, k):
        probability = probability * ((n-i)/n)
        # Генерация комбинаций с повторениями для l - k элементов
    values = itertools.combinations_with_replacement(np.arange(1, k + 1), l - k)

    # Вычисление вероятности повторений имен
    for j in values:
        prob1 = 1

        for i in range(l-k):
            prob1 = prob1 * j[i]
        a = a + prob1

    # Общая вероятность с учетом повторений имен
```

```
probability = probability * (a/(n**(1 - k)))
return probability
```

In [103...]

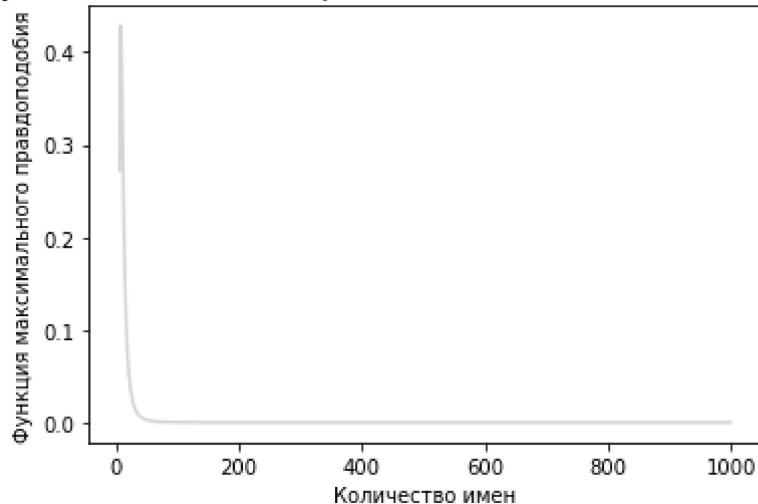
```
n_values1 = np.arange(6,1000)
ps = [calculate_likelihood(6, n) for n in range(6, 1000)]

plt.plot(n_values1, ps, color = 'pink')
plt.xlabel('Количество имен')
plt.ylabel('Функция максимального правдоподобия')
plt.title('Функция максимального правдоподобия от общего количества имен')

print(f' ML-оценка: {n_values1[ps == np.max(ps)][0]}')
```

ML-оценка: 8

Функция максимального правдоподобия от общего количества имен



пункт б)

Напишем функцию, которая бы вычисляла математическое ожидание числа разных имён у 10 таксистов. Для этого мы будем для каждого k вычислять вероятность $\text{calculate_likelihood}(k,n)$ того, что среди 10 таксистов будет ровно k разных имён.

In [97]:

```
def expected_names(n):
    sum = 0

    for k in range(1, 11):
        sum = sum + k * calculate_likelihood(k,n)

    return sum
```

In [106...]

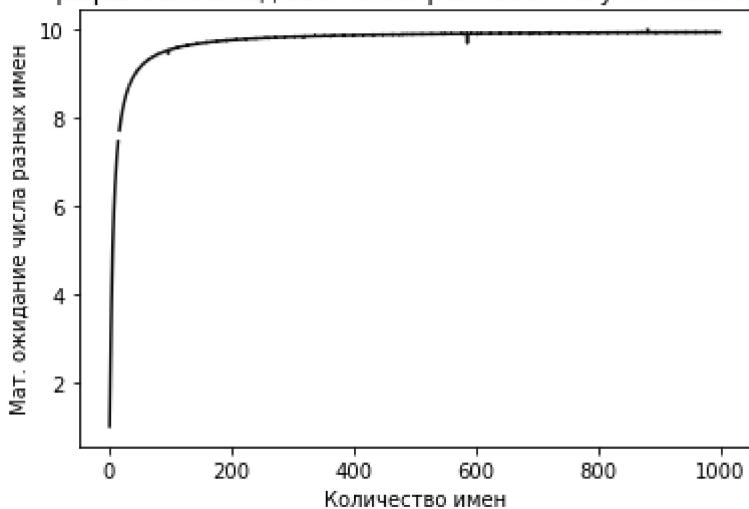
```
n_values2 = np.arange(1, 1000)
E2 = np.array([expected_names(n) for n in n_values2])

plt.plot(np.arange(1,1000), E2, color = 'navy')
plt.xlabel('Количество имен')
plt.ylabel('Мат. ожидание числа разных имен')
plt.title('График мат. ожидания числа разных имен у 10 таксистов')
```

```
print(f'Оценка общего числа таксистов при помощи метода моментов: {n_values2[abs(E2 - 6)]}
```

```
C:\Users\shagi\AppData\Local\Temp\ipykernel_20248\2876163027.py:22: RuntimeWarning: divide by zero encountered in long_scalars
    probability = probability * (a/(n**(1 - k)))
Оценка общего числа таксистов при помощи метода моментов: 8.
```

График мат. ожидания числа разных имен у 10 таксистов



Задача 3

пункт а)

Сначала надо нагенерировать наблюдения:

```
In [7]: data = expon.rvs (scale = 1, size = 20)
data
```

```
Out[7]: array([0.19785289, 0.07250695, 0.188242 , 0.0079492 , 1.24170027,
   2.17716115, 0.3294011 , 1.99978961, 6.08102981, 0.64775446,
   2.58709133, 2.49073117, 0.16990167, 0.11931748, 1.18732482,
   0.60980863, 0.64466777, 0.63561087, 1.0417917 , 0.6726165 ])
```

классический асимптотический нормальный интервал:

```
In [8]: mean = data.mean()
mean
```

```
Out[8]: 1.155112468071503
```

```
In [9]: std = data.std()
std
```

```
Out[9]: 1.3831316288236621
```

```
In [10]: z_obs_left = mean - 1.96 * std/sqrt(20)
z_obs_left
```

```
Out[10]: 0.5489283046710788
```

```
In [11]: z_obs_r = mean + 1.96 * std/sqrt(20)
z_obs_r
```

```
Out[11]: 1.7612966314719274
```

```
In [12]: #np.sum((mean >= z_obs_left - 0,5) & (mean <= z_obs_r + 0,5))
```

```
In [13]: counter = 0
for i in range (0, 10000):
    data = expon.rvs (scale = 1, size = 20)
    mean = data.mean()
    std = data.std()
    z_obs_left = mean - 1.96 * std/sqrt(20)
    z_obs_r = mean + 1.96 * std/sqrt(20)
    if z_obs_left < 1 and z_obs_r > 1:
        counter+=1
counter
```

```
Out[13]: 9019
```

```
In [14]: p = counter/10000
p
```

```
Out[14]: 0.9019
```

наивный бутстрэп:

```
In [15]: nboot = np.random.choice(data, size = (10000, 20))
```

```
In [16]: nboot
```

```
Out[16]: array([[0.07122824, 1.76295746, 0.88284824, ... , 2.38852343, 0.26314794,
   2.37437784],
   [1.59342768, 2.1070414 , 2.38852343, ... , 0.5471707 , 0.26314794,
   1.76295746],
   [2.37437784, 0.33098424, 1.52175595, ... , 0.1721006 , 0.1721006 ,
   1.09535368],
   ... ,
   [1.52175595, 2.38852343, 1.59342768, ... , 0.33098424, 1.76295746,
   1.52175595],
   [0.88284824, 1.09535368, 0.51504022, ... , 0.5471707 , 0.5471707 ,
```

```
0.13244251],
[1.76295746, 1.59342768, 0.50629081, ..., 2.38852343, 1.59342768,
2.38852343]])
```

In [17]:

```
counter1 = 0
for i in range (0, 10000):
    data = expon.rvs (scale = 1, size = 20)
    nboot = np.random.choice(data, size = (10000, 20))
    boot_mean = nboot.mean(axis = 1)
    z_l = np.percentile (boot_mean, 2.5)
    z_r = np.percentile (boot_mean, 97.5)
    if z_l < 1 and z_r > 1:
        counter1+=1
```

In [18]:

```
counter1
```

Out[18]:

```
9069
```

In [19]:

```
p1 = counter1/10000
p1
```

Out[19]:

```
0.9069
```

бутстрэп t-статистики

In [20]:

```
counter2 = 0
for i in range (0, 10000):
    data = expon.rvs (scale = 1, size = 20)
    nboot = np.random.choice(data, size = (10000, 20))
    final_boot = (nboot.mean(axis = 1) - data.mean())/nboot.std(axis = 1)
    q_l = np.percentile (final_boot, 2.5)
    q_r = np.percentile (final_boot, 97.5)
    left = data.mean() - q_r*data.std()
    right = data.mean() - q_l*data.std()
    if left < 1 and right > 1:
        counter2+=1
counter2
```

Out[20]:

```
9440
```

In [21]:

```
p2 = counter2/10000
p2
```

Out[21]:

```
0.944
```

пункт б)

классический асимптотический нормальный интервал:

In [22]:

```
from scipy.stats import t
```

In [23]:

```
counter01 = 0
for i in range (0, 10000):
    data = t.rvs (df = 3, size = 20)
    mean = data.mean()
    std = data.std()
    z_obs_left = mean - 3.182 * std/sqrt(20)
    z_obs_r = mean + 3.182 * std/sqrt(20)
    if z_obs_left < 0 and z_obs_r > 0:
        counter01+=1
counter01
```

Out[23]:

9968

In [24]:

```
p = counter01/10000
p
```

Out[24]:

0.9968

наивный бутстрэп:

In [25]:

```
counter02 = 0
for i in range (0, 10000):
    data = t.rvs (df = 3, size = 20)
    nboot = np.random.choice(data, size = (10000, 20))
    boot_mean = nboot.mean(axis = 1)
    z_l = np.percentile (boot_mean, 2.5)
    z_r = np.percentile (boot_mean, 97.5)
    if z_l < 0 and z_r > 0:
        counter02+=1
```

In [26]:

```
p = counter02/10000
p
```

Out[26]:

0.9214

бутстрэп t-статистики

In [27]:

```
counter03 = 0
for i in range (0, 10000):
    data = t.rvs (df = 3, size = 20)
    nboot = np.random.choice(data, size = (10000, 20))
```

```

final_boot = (nboot.mean(axis = 1) - data.mean())/nboot.std(axis = 1)
q_l = np.percentile (final_boot, 2.5)
q_r = np.percentile (final_boot, 97.5)
left = data.mean() - q_r*data.std()
right = data.mean() - q_l*data.std()
if left < 0 and right > 0:
    counter03+=1
counter03

```

Out[27]: 9204

In [28]:

```

p = counter03/10000
p

```

Out[28]: 0.9204

пункт в)

Лучше оказался второй способ, когда наблюдения имеют распределение Стьюдента, так как там вероятность покрытия больше, чем в первом способе.

Задача 4

In [7]:

```

df = pd.read_csv('Задачи 4.csv', sep = ';')
df

```

Out[7]:

	Фамилия	Балл
0	Репенкова	16
1	Ролдугина	0
2	Сафина	19
3	Сидоров	26
4	Солоухин	21
...
327	Сенников	19
328	Ся	0
329	Сятова	0
330	Темиркулов	0
331	Эшмееев	16

332 rows × 2 columns

пункт а)

In [8]:

```
df_glas = df[df['Фамилия'].str.startswith(('А', 'У', 'О', 'Ы', 'И', 'Э', 'Я', 'Ю', 'Ё', 'Г'))]
df_glas
```

Out[8]:

17	25
18	26
32	25
45	29
46	26
47	20
48	19
49	24
50	25
51	28
67	16
68	16
85	15
113	17
114	4
126	23
143	21
144	13
158	16
166	19
167	6
168	15
169	8
170	13
171	23
172	11
173	7
174	12
184	20
188	20
189	17
220	18
221	20
227	17
228	11
229	21
230	5
240	13
270	13
271	13
272	21
273	0
274	0
275	0
276	0
277	0
299	0
300	22
331	16

Name: Балл, dtype: int64

In [9]:

```
df_soglas = df[df['Фамилия'].str.startswith(('Б', 'В', 'Г', 'Д', 'Ж', 'З', 'Й', 'К', 'Л'))]
df_soglas
```

```
Out[9]: 0      16
        1      0
        2      19
        3      26
        4      21
        ..
       326     4
       327    19
       328     0
       329     0
       330     0
Name: Балл, Length: 283, dtype: int64
```

```
In [10]: std1 = sqrt(df_glas.var()/49 + df_soglas.var()/283)
std1
```

```
Out[10]: 1.2655940198521967
```

```
In [11]: r_1 = df_glas.mean() - df_soglas.mean()
```

```
In [12]: stats.ttest_ind(df_glas, df_soglas, equal_var = False)
```

```
Out[12]: Ttest_indResult(statistic=-0.8519661870595602, pvalue=0.3974027153843839)
```

Гипотеза не отвергается, так как р-значение больше уровня значимости.

пункт 6)

```
In [13]: glas1 = np.random.choice(df_glas, size = (10000, 49))
```

```
In [14]: np.random.choice(df_glas, size = (10000, 49)).mean(axis = 1)
```

```
Out[14]: array([18.18367347, 15.69387755, 14.59183673, ..., 16.02040816,
   15.30612245, 15.42857143])
```

```
In [15]: glas = np.random.choice(df_glas, size = (10000, 49)).mean(axis = 1)
```

```
In [16]: soglas1 = np.random.choice(df_soglas, size = (10000, 283))
```

```
In [17]: np.random.choice(df_soglas, size = (10000, 283)).mean(axis = 1)
```

```
Out[17]: array([16.42402827, 15.6254417 , 15.65724382, ..., 16.26855124,
   16.02826855, 16.68551237])
```

```
In [18]: soglas = np.random.choice(df_soglas, size = (10000, 283)).mean(axis = 1)
```

```
In [19]: r = glas - soglas
r
```

```
Out[19]: array([-1.11819427,  0.57647653, -1.11862696, ..., -0.93581885,
   0.81683133, -3.38761087])
```

```
In [20]: np.percentile (r, 2.5)
```

```
Out[20]: -3.6009176462104273
```

```
In [21]: np.percentile (r, 97.5)
```

```
Out[21]: 1.345085454676572
```

```
In [22]: r_1
```

```
Out[22]: -1.0782433114588574
```

```
In [23]: np.sum(r > 0)/10000
```

```
Out[23]: 0.194
```

```
In [24]: np.sum(r < 0)/10000
```

```
Out[24]: 0.806
```

Минимум из этих долей: 0,194, поэтому р-значение равно $0,194^2 = 0,388$. Значит, гипотеза не отвергается.

ПУНКТ В)

```
In [25]: glas.std()
```

```
Out[25]: 1.1716247342956396
```

```
In [26]: soglas.std()
```

```
Out[26]: 0.4654331782582396
```

```
In [27]: glas.mean()
```

```
Out[27]: 15.265785714285713
```

```
In [28]: soglas.mean()
```

```
Out[28]: 16.3593445229682
```

```
In [29]: glas1.var(axis = 1)/49
```

```
Out[29]: array([1.2789569 , 1.16519477, 1.24482146, ..., 1.48210355, 1.28169385,
   1.10453978])
```

```
In [30]: soglas1
```

```
Out[30]: array([[16, 17, 16, ..., 17, 0, 0],
   [16, 7, 25, ..., 18, 18, 17],
   [26, 28, 21, ..., 26, 8, 30],
   ...,
   [22, 27, 30, ..., 15, 10, 25],
   [7, 12, 19, ..., 24, 20, 10],
   [0, 18, 18, ..., 18, 25, 18]], dtype=int64)
```

```
In [31]: std = np.sqrt(glas1.var(axis = 1)/49 + soglas1.var(axis = 1)/283)
std
```

```
Out[31]: array([1.23164481, 1.1799323 , 1.20278291, ..., 1.32435134, 1.21526112,
   1.16381276])
```

```
In [32]: glas2 = (glas1.mean(axis = 1) - soglas1.mean(axis = 1) - r_1)/std
glas2
```

```
Out[32]: array([-0.70653115, 1.29793684, -1.33827043, ..., -1.35362362,
   -0.42695163, -0.28280039])
```

```
In [33]: np.percentile (glas2, 2.5)
```

```
Out[33]: -1.9113414781533886
```

```
In [34]: np.percentile (glas2, 97.5)
```

```
Out[34]: 2.095133528091791
```

```
In [35]: r_1/std1
```

```
Out[35]: -0.8519661870595603
```

Значит, гипотеза не отвергается.

```
In [36]: np.sum(glas2 < r_1/std1)/10000
```

```
Out[36]: 0.1917
```

```
In [37]: np.sum(glas2 > r_1/std1)/10000
```

Out[37]: 0.8083

Минимум из этих долей: 0,1917, поэтому р-значение равно $0,1917^2 = 0,3834$. Значит, гипотеза не отвергается.

ПУНКТ Г)

In [38]:

```
df_per = pd.concat([df_glas, df_soglas])
df_per
```

Out[38]:

17	25
18	26
32	25
45	29
46	26
..	
326	4
327	19
328	0
329	0
330	0

Name: Балл, Length: 332, dtype: int64

In [39]:

```
np.random.permutation(df_per)
```

Out[39]:

```
array([23, 17, 15, 17, 19, 15, 12, 0, 21, 11, 16, 25, 23, 13, 8, 28, 29,
       16, 19, 10, 17, 13, 15, 13, 11, 22, 25, 0, 17, 13, 14, 17, 25, 13,
       0, 19, 27, 19, 23, 16, 4, 12, 19, 24, 21, 0, 0, 16, 14, 20, 25,
       0, 18, 5, 11, 0, 21, 14, 10, 26, 0, 19, 20, 14, 0, 23, 15, 17,
       22, 20, 25, 18, 20, 6, 23, 23, 13, 17, 15, 18, 13, 18, 12, 10, 25,
       18, 29, 25, 26, 10, 21, 27, 23, 9, 22, 0, 24, 16, 27, 16, 0, 24,
       0, 24, 16, 18, 27, 14, 13, 0, 24, 20, 11, 7, 24, 6, 15, 16, 18,
       19, 21, 16, 5, 23, 17, 16, 0, 11, 9, 17, 14, 7, 24, 13, 19, 22,
       23, 5, 21, 0, 8, 21, 18, 0, 22, 0, 0, 18, 24, 7, 24, 18, 19,
       21, 0, 23, 18, 13, 23, 22, 12, 9, 17, 21, 13, 17, 20, 22, 14, 12,
       8, 16, 10, 13, 12, 17, 23, 8, 18, 18, 22, 0, 0, 22, 26, 17, 20,
       20, 24, 15, 14, 18, 18, 25, 23, 23, 0, 22, 22, 18, 25, 8, 20, 20,
       15, 4, 4, 0, 25, 16, 24, 26, 11, 26, 22, 22, 22, 27, 16, 18, 15,
       7, 10, 17, 10, 23, 18, 24, 9, 17, 9, 15, 28, 0, 0, 27, 7, 19,
       12, 15, 12, 9, 18, 26, 24, 17, 28, 0, 15, 12, 23, 0, 16, 19, 18,
       28, 16, 19, 5, 13, 23, 20, 23, 24, 0, 21, 0, 17, 15, 26, 19, 21,
       26, 22, 19, 29, 19, 30, 11, 17, 15, 20, 0, 17, 18, 25, 10, 17, 25,
       26, 22, 28, 0, 12, 13, 23, 0, 20, 23, 25, 21, 26, 22, 16, 15, 17,
       22, 27, 27, 28, 23, 0, 0, 18, 24, 19, 18, 17, 9, 10, 0, 20, 11,
       27, 22, 23, 14, 12, 24, 0, 26, 28], dtype=int64)
```

In [40]:

```
lst = []
```

In [41]:

```
for i in range (0, 9999):
    test = np.random.permutation(df_per)
    delta = test[:49].mean() - test[49:].mean()
    lst.append(delta)
```

In [42]: `np.percentile (lst, 2.5)`

Out[42]: -2.466863777313046

In [43]: `np.percentile (lst, 97.5)`

Out[43]: 2.3933078531766068

In [44]: `r_1`

Out[44]: -1.0782433114588574

In [45]: `lst1 = np.array(lst)`

In [46]: `np.sum(lst1 < r_1)/10000`

Out[46]: 0.1898

In [47]: `np.sum(lst1 > r_1)/10000`

Out[47]: 0.8057

Минимум из этих долей: 0,1898, поэтому р-значение равно $0,1898^2 = 0,3796$. Значит, гипотеза не отвергается.

Задача 5

пункт а)

In [48]: `median = df['Балл'].median()
median`

Out[48]: 17.5

Медиана балла за экзамен равна 17,5.

Создадим новую переменную, которая будет принимать значение True, если первая буква фамилии - гласная, и значение False - если согласная.

In [49]: `df['lettervowel'] = df['Фамилия'].str.startswith(('А', 'У', 'О', 'Ы', 'И', 'Э', 'Я', 'Ю'))
df['lettervowel'] = df['lettervowel'].astype(int)
df`

Out[49]:

	Фамилия	Балл	lettervowel
0	Репенкова	16	0
1	Ролдугина	0	0
2	Сафина	19	0
3	Сидоров	26	0
4	Солоухин	21	0
...
327	Сенников	19	0
328	Ся	0	0
329	Сятова	0	0
330	Темиркулов	0	0
331	Эшмееев	16	1

332 rows × 3 columns

In [50]:

```
n1 = 0
n2 = 0
n3 = 0
n4 = 0
for i in range (0, 332):
    if (df['Балл'][i] < median) & (df['lettervowel'][i] == 1):
        n1+=1
    if (df['Балл'][i] > median) & (df['lettervowel'][i] == 1):
        n2+=1
    if (df['Балл'][i] < median) & (df['lettervowel'][i] == 0):
        n3+=1
    if (df['Балл'][i] > median) & (df['lettervowel'][i] == 0):
        n4+=1
```

In [51]:

n1, n2, n3, n4

Out[51]:

(28, 21, 138, 145)

In [52]:

```
data = {'Меньше медианы': [28, 138], 'Больше медианы': [21, 145]}

df_sopr = pd.DataFrame(data, index = ['Гласная', 'Согласная'])
```

Out[52]:

	Меньше медианы	Больше медианы
Гласная	28	21
Согласная	138	145

Далее задачу я решала на листочке, так как так мне было удобнее строить Доверительный Интервал и выполнять все вычисления.



В данном пункте гипотеза отвергаться не будет, так как значение 1 входит в наш интервал.

пункт б)

Этот пункт я тоже сделала на листочке.



Получается, что значение 1 снова входит в интервал, значит, гипотеза не отвергается. P-value будет равно 0,199.

пункт в)

```
In [62]: df_glass5 = df[df.Балл > median][df.lettervowel == 1]
```

```
C:\Users\shagi\AppData\Local\Temp\ipykernel_23520/3470118115.py:1: UserWarning: Boolean Series key will be reindexed to match DataFrame index.  
df_glass5 = df[df.Балл > median][df.lettervowel == 1]
```

```
In [63]: df_glass5
```

```
Out[63]:
```

	Фамилия	Балл	lettervowel
17	Адилхан	25	1
18	Алексанян	26	1
32	Охотин	25	1
45	Аврамчук	29	1
46	Авсеенко	26	1
47	Адамокова	20	1
48	Адамцева	19	1
49	Азаров	24	1
50	Алексеева	25	1
51	Афанасьев	28	1
126	Ермишова	23	1
143	Овчарова	21	1
166	Ягжов	19	1
171	Акимов	23	1
184	Есауленко	20	1
188	Иванов	20	1

Фамилия	Балл	lettervowel
---------	------	-------------

220	Уначева	18	1
221	Ушатова	20	1
229	Адмайкин	21	1
272	Абдулаева	21	1
300	Ермаков	22	1

In [64]:

```
df_soglas5 = df[df.Балл > median][df.lettervowel == 0]
df_soglas5
```

C:\Users\shagi\AppData\Local\Temp\ipykernel_23520/2858113806.py:1: UserWarning: Boolean Series key will be reindexed to match DataFrame index.

```
df_soglas5 = df[df.Балл > median][df.lettervowel == 0]
```

Out[64]:

	Фамилия	Балл	lettervowel
2	Сафина	19	0
3	Сидоров	26	0
4	Солоухин	21	0
5	Старощук	22	0
6	Стогова	20	0
...
315	Мацко	23	0
316	Менаджиев	25	0
317	Москвина	19	0
324	Полищук	25	0
327	Сенников	19	0

145 rows × 3 columns

In [65]:

```
glas_g = np.random.choice(df_glas5['Балл'], size = (10000, 21))
```

In [66]:

```
soglas_g = np.random.choice(df_soglas5['Балл'], size = (10000, 145))
```

In [67]:

```
gen_boot_sample = np.append(glas_g, soglas_g, axis = 1)
```

In [68]:

```
meds = np.median(gen_boot_sample, axis = 1)
meds
```

Out[68]:

```
array([23., 23., 22., ..., 23., 22., 22.])
```

```
In [69]: glas_gb = (glas_g > meds.reshape(-1, 1)).sum(axis = 1)
```

```
In [70]: glas_gl = (glas_g <= meds.reshape(-1, 1)).sum(axis = 1)
```

```
In [71]: glass5 = glas_gb/glas_gl
```

```
In [72]: soglas_gb = (soglas_g > meds.reshape(-1, 1)).sum(axis = 1)
```

```
In [73]: soglas_gl = (soglas_g <= meds.reshape(-1, 1)).sum(axis = 1)
```

```
In [74]: soglas5 = soglas_gb/soglas_gl
```

```
In [75]: ch_obs = glass5/soglas5  
ch_obs
```

```
Out[75]: array([0.81818182, 2.38461538, 0.82608696, ..., 0.66935484, 0.44057971,  
2.99242424])
```

```
In [76]: np.percentile (ch_obs, 2.5)
```

```
Out[76]: 0.36325851393188857
```

```
In [77]: np.percentile (ch_obs, 97.5)
```

```
Out[77]: 2.612244897959184
```

Значение 1 входит в наш ДИ, поэтому гипотеза не отвергается. P-value = 0.9898.

Задача 6

пункт а)

```
In [75]: df['Количество букв в фамилии'] = df['Фамилия'].apply(len)
```

```
In [76]: df
```

```
Out[76]:   Фамилия  Балл  lettervowel  Количество букв в фамилии
```

	Фамилия	Балл	lettervowel	Количество букв в фамилии
0	Репенкова	16	0	9
1	Ролдугина	0	0	9
2	Сафина	19	0	6

	Фамилия	Балл	lettervowel	Количество букв в фамилии
3	Сидоров	26	0	7
4	Солоухин	21	0	8
...
327	Сенников	19	0	8
328	Ся	0	0	2
329	Сятова	0	0	6
330	Темиркулов	0	0	10
331	Эшмееев	16	1	6

332 rows × 4 columns

In [77]:

```
score_mean = df['Балл'].mean()
score_mean
```

Out[77]:

```
16.204819277108435
```

In [78]:

```
f_mean = df['Количество букв в фамилии'].mean()
f_mean
```

Out[78]:

```
7.86144578313253
```

In [79]:

```
beta = score_mean/f_mean
beta
```

Out[79]:

```
2.0613026819923372
```

Оцененная методом моментов Бета: 2,06.

In [80]:

```
df['Балл'].corr(df['Количество букв в фамилии'])
```

Out[80]:

```
0.025328052669147665
```

пункт б)

In [81]:

```
table = df['Балл'].copy()
table
```

Out[81]:

0	16
1	0
2	19
3	26
4	21
	..
327	19

```
328      0
329      0
330      0
331     16
Name: Балл, Length: 332, dtype: int64
```

In [82]:

```
df_1 = pd.DataFrame(np.random.permutation(table))
df_1
```

Out[82]:

	0
0	16
1	18
2	23
3	16
4	18
...	...
327	0
328	4
329	16
330	26
331	25

332 rows × 1 columns

In [83]:

```
corr_coef = []
```

In [84]:

```
for i in range (0, 9999):
    test = np.random.permutation(table)
    df_test = pd.DataFrame(test)
    df_test.columns = ['Балл']
    coefs = df_test['Балл'].corr(df['Количество букв в фамилии'])
    corr_coef.append(coefs)
```

In [85]:

```
corr_coef
```

Out[85]:

```
[-0.050599888917924066,
 0.027873123336870736,
 -0.049751532028683,
 0.006452111883534858,
 0.0333874431169374,
 -0.034056929577724035,
 0.09383287147536043,
 -0.08135282615291119,
 0.03169072933845536,
 -0.01793814868214457,
 -0.038934981690859925,
```

-0.027906342130726618,
0.010057628662809232,
-0.10022876693852395,
0.02702476644762972,
0.06053486357265018,
0.0696547001319912,
-0.12992125806195987,
0.03317535389462718,
-0.04784272902789066,
0.07050305702123226,
0.06350411268499374,
-0.013908453458249719,
0.03169072933845536,
-0.08665505671066759,
0.0019982382150195038,
-0.09344191182459578,
0.06074695279496043,
0.036780870673901504,
-0.02090739779448815,
0.04187101200934767,
0.006664201105845147,
0.0032707735488810685,
0.07898662591364247,
-0.12101351072492907,
0.04611279645555278,
0.02087417900063231,
0.08110751813674505,
-0.032148126576931745,
0.055868900681824554,
-0.09386609026921637,
-0.05378122725257783,
0.018329108332909204,
0.0075125579950861545,
-0.05887136858802403,
0.010481807107429764,
0.07622946602360915,
0.09362078225305012,
0.07198768157740404,
-0.04232840924782401,
0.012602699330532268,
0.04950622401251687,
0.007088379550465667,
0.01705657299904771,
-0.12398275983727267,
-0.010939204345906122,
0.04865786712327589,
-0.09026057348994197,
-0.045085569137857405,
0.0022103274373297377,
0.03720504911852202,
-0.05420540569719843,
-0.026633806796865087,
0.0030586843265707833,
0.131584753046586,
-0.023452468462211225,
-0.024725003796072814,
0.00390704121581181,
0.0425072796762784,
0.004755398105052834,
-0.015605167236731762,

-0.03320857268848303,
-0.023876646906831733,
0.003058684326570771,
-0.04741855058327017,
0.00920927177356819,
-0.014332631902870218,
0.04356772578782968,
-0.033208572688483,
0.0658370941304066,
-0.04996362125099328,
0.022570892779114334,
-0.04190423080320351,
0.09107571158532704,
-0.006273241455080502,
0.1023164403677707,
-0.009454579789734355,
0.05968650668340917,
-0.08813968126683934,
-0.029390966686898445,
-0.011363382790526615,
0.06859425402043992,
0.006876290328155376,
-0.0860187890437368,
0.07262394924433486,
0.0639282911296143,
0.15491456750071414,
-0.01602934568135226,
-0.1524027156268471,
-0.09959249927159322,
0.01663239455442719,
-0.003728170787357415,
0.044416082677070715,
0.08534930258295012,
0.04590070723324252,
-0.08771550282221884,
-0.019846951682936917,
0.03444788922848864,
-0.027482163686106155,
0.09171197925225784,
0.06371620190730402,
0.12310118415417574,
0.07941080435826298,
-0.05865927936571376,
-0.07986820159673942,
-0.01242382890207791,
-0.03151185891000092,
-0.06290106381191891,
-0.03808662480161889,
-0.04084378469165227,
-0.05166033502947529,
-0.05759883325416249,
0.0238434281129759,
-0.04190423080320352,
-0.006485330677390719,
-0.01348427501362922,
-0.042964676914754786,
-0.001395189341944608,
-0.016877702570593307,
-0.05208451347409582,
0.04547652878862203,

-0.009878758234354827,
-0.1006529453831445,
-0.06777911592505473,
-0.05929554703264454,
-0.08517043215449582,
-0.07605059559515483,
0.0017861489927092555,
-0.010939204345906102,
-0.07223298959357012,
-0.01751397023752405,
0.048657867123275814,
-0.023452468462211228,
0.022146714334493812,
-0.04550974758247788,
-0.03257230502155224,
0.06986678935430145,
0.06435246957423478,
-0.04190423080320356,
-0.011151293568216373,
-0.002031457008875362,
-0.08092864770829065,
0.03508415689541945,
0.025328052669147644,
0.02363133889066562,
-0.0016072785642548698,
-0.025361271463003588,
0.044840261121691205,
0.06265575579575275,
-0.03999542780241121,
0.001786148992709258,
-0.05653838714261124,
0.022782982001424597,
-0.010939204345906131,
-0.020907397794488154,
-0.038298714023929176,
-0.09110893037918297,
0.04208310123165793,
0.011542253218981011,
-0.11189367416558801,
0.028509391003801503,
0.045052350344001485,
-0.04869108591713178,
-0.07923193392980862,
-0.0041523492319779295,
0.09574167447615278,
0.040386387453175875,
-0.09386609026921634,
-0.005424884565839482,
0.08450094569370917,
0.02829730178149121,
0.03720504911852198,
0.056505168348755325,
-0.008182044455872763,
0.02108626822294257,
0.050142491679447655,
0.004119130438122072,
0.0329632646723169,
-0.025573360685313812,
0.08238005347060656,
-0.10489472982934964,

0.009845539440498987,
0.029145658670732266,
0.010481807107429757,
0.0851372133606399,
-0.021119487016798444,
-0.14476750362367785,
0.011966431663601554,
0.05396009768103221,
0.045688618010932276,
0.008997182551257938,
0.056080989904134765,
0.08131960735905522,
-0.027694252908416362,
0.010269717885119488,
0.045264439566311765,
0.03932594134162458,
-0.08813968126683937,
0.07347230613357578,
0.015147769998255374,
-0.02536127146300354,
0.017056572999047694,
0.06456455879654509,
-0.03575364335620609,
0.01981373288908103,
0.07665364446822963,
-0.13225423950737258,
-0.07477806026129322,
-0.05123615658485479,
0.006240022661224608,
0.10528568948011427,
-0.014332631902870215,
-0.01136338279052665,
-0.05017571047330347,
0.09065153314070663,
-0.045509747582477895,
-0.017089791792903552,
-0.027906342130726614,
0.017056572999047694,
-0.05526585180874968,
-0.058023011698783,
-0.039783338580100955,
-0.07774730937363684,
0.0696547001319912,
0.050778759346378426,
-0.028118431353036894,
-0.02854260979765738,
0.03487206767310921,
0.08598557024988089,
-0.03469319724465481,
-0.00033474323039329724,
0.07813826902440145,
-0.05526585180874969,
0.06944261090968093,
-0.1161354586117932,
0.018541197555219484,
-0.0016072785642548626,
-0.020907397794488147,
-0.01009084745666511,
0.03275117545000666,
0.045264439566311765,

0.088530640917604,
-0.11634754783410345,
-0.007333687566631788,
0.029357747893042525,
0.011754342441291274,
-0.05356913803026761,
-0.01030293667897537,
-0.022816200795280475,
-0.013696364235939486,
0.027873123336870757,
-0.010090847456665103,
0.006664201105845131,
-0.08580669982142658,
-0.014120542680559966,
0.09086362236301683,
0.06223157735113221,
0.03402371078386817,
0.12034402426414244,
0.062867845018063,
0.09722629903232448,
-0.07711104170670603,
-0.04805481825020095,
0.06265575579575274,
0.01790492988828868,
0.0401742982308656,
0.028297301781491285,
-0.05059988891792401,
-0.13161797184044183,
0.06456455879654506,
-0.015605167236731759,
0.03444788922848868,
-0.025149182240693294,
0.013451056219773334,
0.04314354734320914,
0.006240022661224619,
-0.017726059459834312,
0.052687562347170716,
0.12585834404420912,
0.018117019110598973,
0.02108626822294256,
0.059474417461098945,
-0.005000706121218978,
-0.01899859479369587,
-0.11019696038710604,
0.000725702881157935,
0.034447889228488696,
0.02320716044604513,
0.038477584452383556,
0.010269717885119498,
-0.07499014948360345,
0.051627116235619414,
0.06732171868657838,
-0.048478996694821475,
-0.02811843135303687,
0.0022103274373297278,
-0.01963486246062662,
-0.0016072785642548613,
0.04674906412248355,
0.029357747893042546,
-0.14243452217826486,

-0.05059988891792406,
-0.06290106381191886,
-0.03320857268848301,
0.09934719125542711,
0.07707782291285013,
-0.09747160704849064,
0.0482336886786553,
-0.05738674403185223,
-0.1318300610627522,
-0.016453524125972775,
-0.03660200024544714,
0.04738533178941431,
0.052475473124860415,
-0.05272078114102662,
0.03253908622769635,
0.00899718255125792,
-0.031723948132311254,
-0.01708979179290356,
-0.007757866011252285,
-0.0626889745896086,
0.08683392713912191,
-0.0028798138981163926,
0.05883814979416819,
0.06329202346268349,
-0.005212795343529195,
0.005179576549673357,
-0.04317676613706508,
-0.04254049847013429,
0.03953803056393483,
0.0032707735488809488,
0.04059847667548608,
0.024267606557596393,
-0.07371761414974198,
0.06265575579575272,
-0.053144959585647134,
0.04568861801093227,
-0.03638991102313685,
0.05226338390255019,
0.018753286777529726,
-0.022179933128349698,
-0.0308755912430702,
-0.08686714593297783,
0.07898662591364244,
-0.08092864770829064,
-0.10659144360783164,
0.021086268222942545,
-0.09768369627080085,
-0.014968899569801044,
0.046112796455552746,
0.07029096779892198,
-0.03935916013548044,
0.024479695779906635,
0.06604918335271685,
0.009209271773568216,
0.041234744342416856,
-0.0007589216750138156,
0.00030586843265707534,
0.06456455879654506,
0.0556568114595143,
0.004331219660432325,

0.04356772578782971,
0.016208216109806665,
0.10995165237093987,
-0.034056929577724056,
-0.06587031292426251,
0.03508415689541944,
-0.06671866981350351,
0.03211490778307588,
-0.054841673364129165,
0.023207160446045105,
0.0344478892284887,
0.05459636534796302,
-0.013484275013629194,
-0.030239323576139412,
-0.02854260979765738,
-0.03214812657693172,
0.03593251378466048,
-0.020059040905247134,
0.03508415689541945,
-0.09238146571304447,
0.0779261798020912,
-0.10489472982934957,
-0.06438568836809068,
0.034872067673109214,
0.07071514624354248,
0.17018499150705269,
-0.05632629792030093,
-0.07859566626287784,
0.07983498280288347,
0.00199823821501949,
0.023843428112975892,
0.12967595004579371,
-0.060992260811126575,
-0.058023011698782975,
-0.07159672192663936,
0.062231577351132224,
0.05883814979416814,
0.01493568077594511,
-0.025785449907624054,
0.0011498813257784443,
0.008997182551257938,
-0.03893498169085997,
-0.0173018810152138,
0.054596365347963034,
-0.02493709301838304,
0.11673850748486807,
-0.017938148682144564,
-0.015393078014421526,
0.055232633014893805,
0.009845539440499006,
-0.04699437213864966,
-0.05356913803026762,
0.02914565867073228,
-0.021119487016798416,
-0.08029238004135994,
0.024691785002216887,
0.03699295989621176,
0.0344478892284887,
0.07241186002202456,
-0.03893498169085994,

-0.054629584141818885,
-0.03999542780241124,
0.07983498280288347,
0.0238434281129759,
0.060959042017270745,
-0.01199965045745741,
0.009633450218188731,
0.0026345058819502733,
-0.030663502020759938,
-0.01793814868214458,
-0.018574416349075407,
-0.018786505571385583,
-0.015393078014421533,
-0.06820329436967526,
0.03699295989621172,
-0.021543665461418927,
0.024691785002216863,
-0.04402512302630607,
0.0898031762514655,
0.03529624611772972,
-0.03702617869006766,
0.009845539440498973,
-0.004576527676598451,
0.022358803556804085,
-0.07074836503739836,
0.024479695779906666,
-0.0018193677865650845,
-0.035965732578516346,
0.019601643666770738,
0.0484457779009656,
-0.01051502590128562,
-0.06671866981350354,
0.033175353894627155,
-0.03342066191079328,
0.06774589713119893,
-0.07901984470749833,
-0.053569138030267645,
0.021086268222942565,
0.01493568077594514,
-0.03893498169085993,
0.003270773548881039,
0.033387443116937414,
0.05502054379258352,
-0.021119487016798413,
-0.03087559124307023,
0.013238966997463084,
-0.0206953085721779,
-0.046358104471718925,
-0.058659279365713746,
0.012178520885911759,
0.02299507122373483,
0.11101209848249113,
-0.10616726516321115,
0.07198768157740403,
-0.07202090037125987,
-0.013908453458249715,
0.0923482469191886,
-0.0054248845658394496,
0.06668545101964765,
0.04102265512010664,

-0.05038779969561385,
0.07050305702123222,
0.03444788922848867,
0.04971831323482713,
-0.016453524125972775,
0.001998238215019518,
-0.01072711512359588,
-0.0584471901434035,
0.0838646780267784,
0.0077246472173964275,
-0.010727115123595853,
-0.07668686326208557,
-0.037662446356998405,
-0.0501757104733035,
-0.006485330677390762,
0.0017861489927092579,
0.0043312196604323,
0.05014249167944765,
0.030206104782283534,
0.004755398105052815,
-0.002879813898116407,
-0.025361271463003543,
-0.037874535579308664,
0.048869956345586094,
0.027448944892250176,
0.01684448377673743,
0.06817007557581942,
0.026600588003009244,
0.0747448414674373,
0.03275117545000663,
0.05396009768103222,
0.07326021691126558,
0.013238966997463034,
-0.06926374048122655,
-0.06650658059119322,
-0.01921068401600611,
-0.045297658360167636,
-0.034056929577724035,
0.03190281856076565,
-0.03151185891000098,
0.10380106492394245,
0.11567806137331675,
0.05162711623561942,
-0.05929554703264452,
-0.005424884565839479,
0.06477664801885531,
0.047385331789414314,
-0.05632629792030093,
-0.048903175139441944,
-0.0985320531600419,
-0.011999650457457404,
0.002210327437329758,
0.11292090148328343,
-0.025361271463003557,
0.05374800845872196,
-0.034905286466965064,
0.024479695779906736,
0.03656878145159127,
-0.028966788242277903,
0.014935680775945126,

0.021722535889873305,
0.09595376369846297,
0.01769284066597843,
0.0955295852538424,
0.059686506683409156,
0.148127712386786,
0.011754342441291254,
-0.01793814868214459,
-0.08029238004135987,
-0.0889880381560803,
-0.026633806796865053,
0.003482862771191301,
0.026600588003009275,
-0.0682032943696753,
0.07283603846664505,
-0.0011831001196343522,
0.054172186903342516,
-0.06777911592505473,
0.024691785002216894,
0.01769284066597847,
0.06541291568578603,
0.04229519045396815,
0.04993040245713736,
0.021086268222942534,
-0.01963486246062662,
-0.019422773238316395,
0.16785201006163972,
0.03847758445238353,
-0.05929554703264454,
-0.008818312122803582,
0.018117019110598927,
-0.049751532028682974,
-0.015180988792111258,
-0.05017571047330349,
0.1485518908314065,
-0.046570193694029156,
-0.032572305021552284,
0.035296246117729725,
0.06244366657344246,
-0.021967843906039463,
0.07474484146743735,
0.0017861489927092334,
0.09022735469608607,
0.08513721336063988,
-0.009242490567424063,
0.10740658170321682,
-0.0018193677865651567,
-0.012211739679767655,
0.0073004687727759076,
0.02235880355680418,
-0.025573360685313767,
-0.0726571680381907,
-0.062264796144988095,
0.06838216479812968,
0.044628171899381,
-0.007969955233562502,
-0.01602934568135228,
0.11652641826255784,
-0.08135282615291113,
-0.029178877464588162,

-0.031299769687690736,
-0.01793814868214463,
-0.028966788242277882,
0.07071514624354248,
0.0009377921034682275,
0.09489331758691165,
-0.01115129356821637,
0.027448944892250225,
0.03190281856076559,
0.026388498780698936,
-0.00012265400808304723,
-0.05653838714261121,
0.04802159945634507,
0.07601737680129891,
0.08238005347060656,
0.08768228402836298,
0.0482336886786553,
0.041871012009347626,
-0.009030401345113872,
0.062019488128821985,
-0.12122559994723932,
-0.0637494207011599,
0.00390704121581182,
0.019813732889081035,
0.011754342441291298,
-0.014968899569801053,
-0.02430082535145223,
0.15449038905609358,
-0.03448110802234458,
0.0005136136588476369,
0.09637794214308346,
0.05862606057185788,
0.01472359155363484,
-0.03214812657693174,
0.04908204556789637,
-0.09174519804611371,
0.0310544616715246,
0.05586890068182457,
-0.02005904090524716,
0.021722535889873378,
-0.032996483466172774,
-0.03151185891000097,
0.04590070723324256,
0.002634505881950247,
0.07135141391047327,
-0.01687770257059329,
0.04144683356472714,
-0.03935916013548047,
-0.00012265400808304423,
0.020237911333701532,
0.06944261090968094,
0.027661034114560508,
-0.06693075903581377,
0.006452111883534869,
0.032751175450006664,
0.007088379550465604,
-0.07180881114894963,
0.025328052669147647,
0.007300468772775846,
-0.07986820159673942,

-0.031087680465380466,
-0.099380410049283,
0.055656811459514295,
-0.09747160704849064,
0.002422416659639985,
0.021298357445252797,
0.07071514624354248,
0.023207160446045164,
-0.0274821636861061,
0.11673850748486808,
-0.03384484035541381,
-0.09874414238235217,
-0.020483219349867614,
0.02532805266914763,
0.007088379550465615,
0.07750200135747068,
-0.06374942070115991,
0.12331327337648593,
0.02257089277911432,
-0.018362327126765107,
-0.012848007346698458,
-0.056326297920301006,
-0.10998487116479577,
0.09616585292077327,
-0.06862747281429579,
0.03614460300697072,
-0.10107712382776503,
-0.004364438454288185,
-0.033844840355413804,
0.05247547312486042,
-0.01730188101521381,
-0.015817256459041994,
-0.05441749491950864,
-0.04953944280637276,
0.004331219660432283,
0.060110685128029646,
-0.014544721125180488,
0.09510540680922193,
-0.05017571047330351,
-0.03087559124307024,
0.02744894489225021,
-0.03999542780241122,
-0.03172394813231125,
0.018753286777529757,
0.014299413109014354,
-0.006697419899700989,
-0.029390966686898407,
-0.011999650457457425,
0.05989859590571943,
-0.013484275013629239,
-0.0009710108973240869,
-0.003091903120426676,
-0.062264796144988144,
-0.002243546231185657,
0.011542253218981008,
-0.008606222900493298,
0.05989859590571943,
-0.028754699019967665,
0.001786148992709252,
-0.08665505671066759,

-0.01624143490366254,
0.08089542891443476,
-0.0930177333799753,
0.03487206767310919,
-0.08941221660070094,
-0.13225423950737264,
0.09510540680922192,
0.03656878145159123,
-0.06671866981350355,
-0.033844840355413804,
-0.02387664690683173,
0.01663239455442716,
0.025752231113768252,
-0.05441749491950865,
-0.010302936678975347,
-0.10171339149469577,
-0.10383428371779832,
0.03678087067390148,
-0.060355993144195826,
-0.032360215799242004,
-0.0007589216750138399,
0.07283603846664509,
0.02129835744525282,
0.043355636565519454,
-0.054205405697198374,
0.050990848568688664,
0.08598557024988093,
-0.02939096668689843,
-0.08156491537522137,
0.02193462511218359,
0.06944261090968094,
-0.043176766137065066,
0.03805340600776303,
0.012178520885911785,
-0.1254673843934444,
-0.06650658059119322,
-0.1006529453831445,
0.07941080435826298,
-0.028330520575347132,
-0.05738674403185223,
-0.023876646906831767,
-0.008606222900493338,
-0.012635918124388211,
0.16488276094929616,
-0.09874414238235223,
0.06011068512802972,
-0.06480986681271121,
-0.05250869191871635,
-0.039995427802411214,
0.049294134790206647,
0.05141502701330914,
-0.021755754683729162,
0.02956983711535275,
0.04971831323482713,
0.0845009456937092,
-0.067142848258124,
0.01832910833290923,
-0.05102406736254456,
0.012178520885911785,
0.1114362769271117,

-0.0419042308032035,
0.0442039934547605,
0.022570892779114345,
0.06965470013199118,
0.03699295989621177,
-0.0238766469068318,
-0.030875591243070225,
0.03805340600776306,
0.04823368867865535,
-0.04763063980558043,
0.029357747893042515,
0.06965470013199117,
-0.09047266271225221,
-0.029815145131518905,
-0.0007589216750138408,
0.0738964845781963,
0.05756561446030662,
0.13985623271668599,
0.006452111883534827,
-0.007333687566631781,
0.04505235034400148,
0.03338744311693738,
-0.008818312122803566,
-0.03236021579924199,
0.07453275224512707,
-0.011787561235147171,
0.013875234664393873,
0.035720424562350236,
0.007300468772775856,
0.036992959896211756,
0.052475473124860464,
0.016632394554427166,
0.060322774350339954,
0.07241186002202454,
-0.03935916013548045,
0.12140447037569364,
0.007512557995086174,
-0.0028798138981164134,
-0.030027234353829216,
0.006452111883534883,
-0.03617782180082659,
-0.0134842750136292,
-0.07605059559515479,
0.0005136136588476609,
0.019813732889081028,
-0.07965611237442913,
0.0584139713495476,
0.12543416559958862,
-0.006697419899701103,
0.08747019480605275,
0.06795798635350911,
0.07856244746902197,
-0.06608240214657272,
-0.005000706121218963,
0.0032707735488810468,
0.01642030533211692,
0.024267606557596418,
0.05247547312486046,
-0.0173018810152138,
0.03868967367469381,

0.027448944892250235,
-0.043176766137065066,
-0.03129976968769074,
0.033811621561557856,
0.013663145442083581,
0.007936736439706651,
0.05396009768103224,
0.02638849878069893,
-0.01327218579131895,
0.031690729338455334,
0.018753286777529726,
-0.12716409817192645,
-0.13416304250816488,
0.0005136136588476608,
0.031266550893834844,
-0.009242490567424067,
0.01366314544208359,
0.07135141391047327,
0.10867911703707832,
0.04886995634558612,
-0.01921068401600613,
-0.052508691918716266,
0.0030586843265707655,
-0.050599888917924024,
-0.016029345681352343,
-0.07392970337205215,
-0.03935916013548047,
-0.05314495958564709,
-0.03087559124307027,
0.022995071223734856,
0.04081056589779638,
-0.07032418659277788,
-0.09153310882380346,
-0.05950763625495478,
0.027448944892250204,
-0.021331576239108672,
0.03805340600776305,
0.03635669222928095,
0.0768657336905399,
0.009633450218188722,
0.011330163996670764,
0.013663145442083604,
-0.02387664690683175,
0.0898031762514656,
0.0830163211375373,
-0.04508556913785736,
-0.06990000814815733,
-0.05865927936571379,
0.06011068512802966,
0.04271936889858867,
-0.009666669012044606,
0.06753380790888862,
0.03444788922848869,
-0.05356913803026762,
0.031690729338455376,
-0.08453416448756502,
-0.043388855359375346,
0.02363133889066564,
0.09892301281080644,
-0.048903175139441965,

-0.023028290017590724,
0.04420399345476046,
0.0779261798020912,
-0.08220118304215222,
0.028509391003801524,
0.01620821610980664,
-0.003091903120426642,
0.046961153344793796,
-0.04975153202868302,
0.018965375999839985,
-0.05293287036333686,
0.09234824691918857,
-0.02154366546141897,
-0.015817256459042,
-0.02196784390603944,
-0.024088736129142002,
-0.028118431353036908,
-0.039995427802411214,
0.021086268222942576,
-0.028754699019967637,
-0.012211739679767655,
0.06986678935430146,
-0.03872289246854968,
0.011330163996670752,
0.004119130438122063,
0.004543308882742576,
-0.09747160704849062,
-0.039783338580100976,
0.02935774789304255,
0.08386467802677836,
-0.01815023790445481,
-0.009878758234354832,
-0.0043644384542881965,
-0.02705798524148561,
-0.04211632002551379,
0.06562500490809636,
0.009421360995878453,
0.030630283226904052,
-0.06268897458960865,
0.0121785208859118,
0.12988803926810397,
-0.10044085616083422,
-0.0024556354534959004,
-0.08177700459753166,
0.05247547312486046,
0.10549777870242455,
-0.04275258769244457,
-0.12419484905958295,
-0.028330520575347122,
0.06329202346268349,
0.0034828627711913137,
0.02787312333687074,
-0.010727115123595846,
-0.03575364335620609,
0.03317535389462714,
-0.012423828902077952,
-0.026633806796865053,
0.06244366657344251,
-0.0425404984701343,
-0.09556280404769832,

```
-0.08177700459753166,  
0.030206104782283583,  
-0.02493709301838305,  
-0.037026178690067614,  
-0.017089791792903555,  
-0.006273241455080482,  
0.037417138340832275,  
0.09022735469608605,  
-0.09047266271225221,  
-0.009878758234354865,  
0.019813732889081007,  
0.020237911333701526,  
0.009209271773568223,  
0.004543308882742614,  
-0.029603055909208715,  
0.051415027013309196,  
0.022782982001424607,  
0.05714143601568614,  
-0.05590211947568045,  
0.004331219660432314,  
-0.03384484035541378,  
-0.002879813898116406,  
0.008785093328947698,  
-0.005424884565839432,  
-0.015393078014421557,  
0.03169072933845538,  
0.08492512413832967,  
0.09743838825463473,  
-0.053357048807957365,  
...]
```

In [86]: `np.percentile (corr_coef, 2.5)`

Out[86]: -0.10680353283014189

In [87]: `np.percentile (corr_coef, 97.5)`

Out[87]: 0.10846702781476802

Значение 0 входит в данный интервал, значит, гипотеза о том, что корреляция равна 0, не отвергается.

In [88]: `coefs1 = np.array(corr_coef)`

In [89]: `np.sum(coefs1 < 0)/10000`

Out[89]: 0.5002

In [90]: `np.sum(coefs1 >= 0)/10000`

Out[90]: 0.4997

Получается, что p-value равно 0,9994.

Задача 7

Я взяла задачу из Теории Вероятностей из КР-1 2015-2016 года.

К

Имеются два игральных кубика: красный со смещенным центром тяжести, так что вероятность выпадения «6» равняется $1/3$, а оставшиеся грани имеют равные шансы на появление и правильный белый кубик. Петя случайным образом выбирает кубик и подбрасывает его.

а) Вероятность того, что выпадет «6», равна... ?

б) Вероятность того, что Петя взял красный кубик, если известно, что выпала шестерка, равна... ?

в) Если бы в эксперименте Петя подбрасывал бы кубик не один раз, а 60 раз, то безусловное математическое ожидание количества выпавших шестёрок равнялось бы... ?



а) Вероятность того, что выпадет "6" можно найти, учитывая вероятности выпадения "6" для каждого кубика и вероятность выбора каждого кубика Петей. Пусть R обозначает красный кубик, а W - белый кубик.



Вероятность выпадения "6" равна сумме вероятности выпадения "6" на красном кубике, умноженной на вероятность выбора красного кубика, и вероятности выпадения "6" на белом кубике, умноженной на вероятность выбора белого кубика:

$P(\text{выпадет "6"}) = P(\text{выбор красного кубика}) * P(\text{выпадет "6" на красном кубике}) + P(\text{выбор белого кубика}) * P(\text{выпадет "6" на белом кубике})$

Regenerate response

Send a message.

Вероятность выпадения "6" на белом кубике равна 1/6:

$$P(\text{выпадет "6" на белом кубике}) = 1/6$$

Подставляя значения в формулу, получаем:

$$P(\text{выпадет "6"}) = (1/2) * (1/3) + (1/2) * (1/6) = 1/6 + 1/12 = 2/12 + 1/12 = 3/12$$

Упрощая дробь, получаем:

$$P(\text{выпадет "6"}) = 1/4$$

Таким образом, вероятность того, что выпадет "6", равна 1/4.

Опять же, приношу извинения за предыдущую ошибку и благодарю за указание на нее.

К

Теперь пересчитай пункт б) и в), исходя из нового правильного решения пункта а)



Исходя из нового правильного решения пункта а), где вероятность выпадения "6" равна 1/4, пересчитаем пункты б) и в):



б) Для вычисления вероятности

Regenerate response

ый кубик, если известно, что

Send a message.



$P(\text{выпадет "6"}) = P(\text{выбор красного кубика}) * P(\text{выпадет "6" на красном кубике}) + P(\text{выбор белого кубика}) * P(\text{выпадет "6" на белом кубике})$

Поскольку Петя случайным образом выбирает кубик, вероятность выбора каждого кубика равна 1/2:

$P(\text{выбор красного кубика}) = P(\text{выбор белого кубика}) = 1/2$

Вероятность выпадения "6" на красном кубике равна 1/3:

$P(\text{выпадет "6" на красном кубике}) = 1/3$

Вероятность выпадения "6" на белом кубике равна 1/6:

$P(\text{выпадет "6" на белом кубике}) = 1/6$

Подставляя значения в формулу, получаем:

$$P(\text{выпадет "6"}) = (1/2) * (1/3) + (1/2) * (1/6) = 1/6 + 1/12 = 3/12 + 2/12 = 5/12$$

Таким образом, вероятность того, что выпадет "6", равна 5/12.

- 6) Для нахождения вероятности того, что Петя взял красный кубик, если известно, что выпала шестерка, можно воспользоваться формулой условной вероятности.

⟳ Regenerate response

Пусть А обозначает событие "выпала шестерка" и Б обозначает событие "взят красный".

Send a message.



Пусть А обозначает событие "выпала шестерка" и В обозначает событие "взят красный кубик". Тогда вероятность того, что Петя взял красный кубик, при условии, что выпала шестерка, можно выразить следующим образом:

$$P(B|A) = P(A \text{ и } B) / P(A)$$

$P(A \text{ и } B)$ - вероятность события "выпала шестерка" и "взят красный кубик"

$P(A)$ - вероятность события "выпала шестерка"

Вероятность события "выпала шестерка" равна $5/12$ (по результатам предыдущего пункта).

Вероятность события "выпала шестерка" и "взят красный кубик" равна:

$$P(A \text{ и } B) = P(\text{выбор красного кубика}) * P(\text{выпадет "6" на красном кубике}) = (1/2) * (1/3) = 1/6$$

Подставляя значения в формулу, получаем:

$$P(B|A) = (1/6) / (5/12) = (1/6) * (12/5) = 2/5$$

Таким образом, вероятность того, что Петя взял красный кубик, если известно, что выпала шестерка, равна $2/5$.

 Regenerate response

в) Для нахождения безусловного математического ожидания количества выпавших

Send a message.



в) Для нахождения безусловного математического ожидания количества выпавших шестерок в 60 подбрасываниях кубика, мы можем использовать свойство линейности математического ожидания.

Пусть X обозначает случайную величину, представляющую количество выпавших шестерок в одном подбрасывании. Так как каждое подбрасывание независимо, X следует распределению Бернулли с вероятностью успеха $p = 5/12$ (вероятность выпадения "6" из предыдущего пункта).

Математическое ожидание одного подбрасывания равно $E(X) = p = 5/12$.

Поскольку каждое подбрасывание независимо, безусловное математическое ожидание количества выпавших шестерок в 60 подбрасываниях будет равно:

$$E(60X) = 60 * E(X) = 60 * (5/12) = 25.$$

Таким образом, безусловное математическое ожидание количества выпавших шестерок в 60 подбрасываниях равно 25.

К У тебя ошибка в пункте а), при вычислении вероятности выпадения "6" получается $1/4$, а не $5/12$

 Regenerate response

Send a message. 



Приношу извинения за ошибку в предыдущем ответе. Давайте пересчитаем вероятность выпадения "6" с учетом правильной вероятности для каждого кубика.



а) Для вычисления вероятности выпадения "6" нужно учесть вероятности для каждого кубика и вероятность выбора каждого кубика Петей. Пусть R обозначает красный кубик, а W - белый кубик.

Вероятность выпадения "6" будет равна сумме вероятности выпадения "6" на красном кубике, умноженной на вероятность выбора красного кубика, и вероятности выпадения "6" на белом кубике, умноженной на вероятность выбора белого кубика:

$$P(\text{выпадет "6"}) = P(\text{выбор красного кубика}) * P(\text{выпадет "6" на красном кубике}) + P(\text{выбор белого кубика}) * P(\text{выпадет "6" на белом кубике})$$

Поскольку Петя случайным образом выбирает кубик, вероятность выбора каждого кубика равна 1/2:

$$P(\text{выбор красного кубика}) = P(\text{выбор белого кубика}) = 1/2$$

Вероятность выпадения "6" на красном кубике равна 1/3:

$$P(\text{выпадет "6" на красном кубике}) = 1/3$$

Вероятность выпадения "6" на белом кубике

Regenerate response

Send a message.



б) Для вычисления вероятности того, что Петя взял красный кубик, если известно, что выпала шестерка, можно использовать формулу условной вероятности.

Пусть А обозначает событие "выпала шестерка", а В обозначает событие "взят красный кубик". Тогда вероятность того, что Петя взял красный кубик, при условии, что выпала шестерка, можно выразить следующим образом:

$$P(B|A) = P(A \text{ и } B) / P(A)$$

$P(A \text{ и } B)$ - вероятность события "выпала шестерка" и "взят красный кубик"

$P(A)$ - вероятность события "выпала шестерка"

Вероятность события "выпала шестерка" равна $1/4$ (по результатам пункта а).

Вероятность события "выпала шестерка" и "взят красный кубик" равна:

$$P(A \text{ и } B) = P(\text{выбор красного кубика}) * P(\text{выпадет "6" на красном кубике}) = (1/2) * (1/4) = 1/8$$

Подставляя значения в формулу, получаем:

$$P(B|A) = (1/8) / (1/4) = (1/8) * (4/1) = 1/2$$

Таким образом, вероятность того, что Петя взял красный кубик, если известно, что выпала шестерка, равна $1/2$.

 Regenerate response

Send a message. 



Таким образом, с помощью подсказок и исправления ошибок удалось добиться правильных ответов и решений.

Задача 8

Мне был полезен курс от Яндекс.Практикума, особенно в начале года, когда мы только начали изучение Теории Вероятностей. В нем есть хорошие блоки по Теории вероятностей и Комбинаторике, которые как раз и помогли лучше понимать, что конкретно мы будем делать в курсе Теории Вероятностей, и некоторые темы стали проще восприниматься. Вот ссылка на Яндекс.Практикум: <https://practicum.yandex.ru/profile/math-foundations/>.