

```
In [ ]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
import sklearn
from sklearn.tree import DecisionTreeClassifier
from collections import defaultdict
```

№4

```
In [ ]: df = pd.read_csv(r"C:\Users\Мой ПК\Dropbox\Мой ПК (LAPTOP-TOID7NG7)\Downloads\22-23_hse_probability.csv", sep = ",")
df = df[['Экзамен', 'Last name']]
df.fillna(0, inplace=True)
df
```

Out[ ]:

	Экзамен	Last name
0	4.0	Репенкова
1	0.0	Ролдугина
2	5.0	Сафина
3	9.0	Сидоров
4	6.0	Солоухин
...	...	...
332	0.0	Наугольнова
333	0.0	Хафизов
334	0.0	Герасимов
335	0.0	Тайирбек уулу
336	0.0	Хананияев

337 rows × 2 columns

```
In [ ]: group1 = df[df['Last name'].str[0].isin(['А', 'О', 'У', 'Э', 'И'])]['Экзамен']
group2 = df[df['Last name'].str[0].isin(['Б', 'В', 'Г', 'Д', 'Ж', 'З', 'Й', 'К', 'Л', 'М', 'Н', 'П', 'Р', 'С', 'Т', 'Ф', 'Х', 'Ц', 'Ч', 'Ш', 'Щ', 'Ъ', 'Ь', 'Э', 'Ю', 'Я'])]['Экзамен']
```

a) [5] Используйте тест Уэлча

```
In [ ]: import pandas as pd
import random
from scipy import stats

mean1 = group1.mean()
mean2 = group2.mean()
print('Средняя оценка для группы 1:', mean1)
print('Средняя оценка для группы 2:', mean2)

t_stat, p_value = stats.ttest_ind(group1, group2)
print("p-значение:", p_value)
if p_value < 0.05:
    print("Отвергаем нулевую гипотезу: ожидаемые результаты экзамена по теории вероятностей у студентов, чьи фамилии начинаются с гласной и согласной буквы, равны.")
else:
    print("Не можем отвергнуть нулевую гипотезу: ожидаемые результаты экзамена по теории вероятностей у студентов, чьи фамилии начинаются с гласной и согласной буквы, равны.")
```

Средняя оценка для группы 1: 4.611111111111111  
Средняя оценка для группы 2: 4.909722222222222  
p-значение: 0.5338093991657107  
Не можем отвергнуть нулевую гипотезу: ожидаемые результаты экзамена по теории вероятностей у студентов, чьи фамилии начинаются с гласной и согласной буквы, равны.

б) [5] Используйте наивный бутстрэп.

```
In [ ]: m_vowels = np.mean(group1)
m_consonants = np.mean(group2)

obs_diff = m_consonants - m_vowels

n_iter = 1000

boot_diffs = []
for i in range(n_iter):
    np.random.seed(i)
    boot_vowels = np.random.choice(group1, size=len(group1), replace=True)
    boot_consonants = np.random.choice(group2, size=len(group2), replace=True)
    boot_diff = np.mean(boot_consonants) - np.mean(boot_vowels)
    boot_diffs.append(boot_diff)

p_value = np.sum(np.abs(boot_diffs) >= np.abs(obs_diff)) / n_iter
```

```
print("Среднее значение ожидаемых результатов экзамена у тех, у кого фамилия начинается с гласной буквы:", m_vowels)
print("Среднее значение ожидаемых результатов экзамена у тех, у кого фамилия начинается с согласной буквы:", m_consonants)
print("Разница средних значений:", obs_diff)
print("p-значение:", p_value)
if p_value < 0.05:
    print("Отвергаем нулевую гипотезу: ожидаемые результаты экзамена по теории вероятностей у студентов, чьи фамилии начинаются с гласной и с согласной буквы, равны.")
else:
    print("Не можем отвергнуть нулевую гипотезу: ожидаемые результаты экзамена по теории вероятностей у студентов, чьи фамилии начинаются с гласной и с согласной буквы, равны.")
```

Среднее значение ожидаемых результатов экзамена у тех, у кого фамилия начинается с гласной буквы: 4.611111111111111

Среднее значение ожидаемых результатов экзамена у тех, у кого фамилия начинается с согласной буквы: 4.909722222222222

Разница средних значений: 0.2986111111111116

p-значение: 0.627

Не можем отвергнуть нулевую гипотезу: ожидаемые результаты экзамена по теории вероятностей у студентов, чьи фамилии начинаются с гласной и с согласной буквы, равны.

в) [5] Используйте бутстрэп t-статистики

```
In [ ]: from scipy.stats import ttest_ind

m_vowels = np.mean(group1)
m_consonants = np.mean(group2)

# Вычисление разности средних значений между группами
diffmean = m_vowels - m_consonants

# Создание массива для бутстрэпирования
bootstrapdiff = np.empty(1000)

# Бутстрэпирование
for i in range(0, 1000):
    bootstrapvowel = np.random.choice(group1, size=len(group1), replace=True)
    bootstrapconsonant = np.random.choice(group2, size=len(group2), replace=True)
    bootstrapdiffi = np.mean(bootstrapvowel) - np.mean(bootstrapconsonant)

# Вычисление p-значения
pvalue = np.sum(bootstrapdiff >= diffmean) / len(bootstrapdiff)

print("p-значение бутстрэпа: ", pvalue)

# Вывод результата
if pvalue < 0.05:
    print("Отвергаем нулевую гипотезу: ожидаемые результаты экзамена по теории вероятностей у студентов, чьи фамилии начинаются с гласной и с согласной буквы, равны.")
else:
    print("Не можем отвергнуть нулевую гипотезу: ожидаемые результаты экзамена по теории вероятностей у студентов, чьи фамилии начинаются с гласной и с согласной буквы, равны.")
```

p-значение бутстрэпа: 0.867

Не можем отвергнуть нулевую гипотезу: ожидаемые результаты экзамена по теории вероятностей у студентов, чьи фамилии начинаются с гласной и с согласной буквы, равны.

г) [5] Используйте перестановочный тест.

```
In [ ]: vowelresults = group1.to_list()
consonantresults = group2.to_list()

def mean(lst):
    return sum(lst)/len(lst)

vowelmean = mean(vowelresults)
consonantmean = mean(consonantresults)

# определяем функцию для вычисления статистической значимости
def permtest(lst1, lst2, nperm):
    obsdiff = mean(lst1)-mean(lst2)
    pooledlst = lst1+lst2
    count = 0
    for i in range(nperm):
        random.shuffle(pooledlst)
        permdiff = mean(pooledlst[:len(lst1)])-mean(pooledlst[len(lst1):])
        if permdiff >= obsdiff:
            count += 1
    return count/nperm

# вычисляем статистическую значимость
pvalue = permtest(vowelresults, consonantresults, 10000)

print("p-значение перестановочного теста: ", pvalue)

# проверяем гипотезу о равенстве
if pvalue < 0.05:
    print("Отвергаем нулевую гипотезу: ожидаемые результаты экзамена по теории вероятностей у студентов, чьи фамилии начинаются с гласной и с согласной буквы, равны.")
else:
    print("Не можем отвергнуть нулевую гипотезу: ожидаемые результаты экзамена по теории вероятностей у студентов, чьи фамилии начинаются с гласной и с согласной буквы, равны.")
```

p-значение перестановочного теста: 0.742

Не можем отвергнуть нулевую гипотезу: ожидаемые результаты экзамена по теории вероятностей у студентов, чьи фамилии начинаются с гласной и с согласной буквы, равны.

```
In [ ]: gr1 = df[df['Last name'].str[0].isin(['А', 'О', 'У', 'Э', 'И'])['Last name'].to_list()
gr2 = df[df['Last name'].str[0].isin(['Б', 'В', 'Г', 'Д', 'Ж', 'З', 'Й', 'К', 'Л', 'М', 'Н', 'П', 'Р', 'С', 'Т', 'Ф', 'Х', 'Ц', 'Ч', 'Ш', 'Щ', 'Ю', 'Я'])['Last name'].to_list()

median = df.median()

gl_bol = 0
gl_men = 0
sog_bol = 0
sog_men = 0

for index, row in df.iterrows():
    if row['Last name'] in gr1:
        if (row['Экзамен'] >= median).bool() == True:
            gl_bol += 1
        else:
            gl_men += 1
    else:
        if (row['Экзамен'] >= median).bool() == True:
            sog_bol += 1
        else:
            sog_men += 1

print('Фамилия на согласную и меньше медианы', sog_men)
print('Фамилия на согласную и больше медианы', sog_bol)
print('Фамилия на гласную и больше медианы', gl_bol)
print('Фамилия на гласную и меньше медианы', gl_men)

Фамилия на согласную и меньше медианы 124
Фамилия на согласную и больше медианы 177
Фамилия на гласную и больше медианы 19
Фамилия на гласную и меньше медианы 17

C:\Users\Мой ПК\AppData\Local\Temp\ipykernel_20608\3507167334.py:4: FutureWarning: The default value of numeric_only in DataFrame.median is deprecated. In a future version, it will default to False. In addition, specifying 'numeric_only=None' is deprecated. Select only valid columns or specify the value of numeric_only to silence this warning.
    median = df.median()
```

а) [5] Постройте 95% асимптотический интервал для отношения шансов хорошо написать экзамен («несогласных» к «согласным»). Проверьте гипотезу о том, что отношение шансов равно 1 и укажите Р-значение

```
In [ ]: import numpy as np
from scipy.stats import norm

n1, n2 = sog_bol, sog_men
m1, m2 = gl_bol, gl_men

p1 = m1 / n1
p2 = m2 / n2

OR = (p2 / (1 - p2)) / (p1 / (1 - p1))

SE_logOR = np.sqrt(1 / m2 + 1 / (n2 - m2) + 1 / m1 + 1 / (n1 - m1))
CI_logOR = np.log(OR) - norm.ppf(0.975) * SE_logOR, np.log(OR) + norm.ppf(0.975) * SE_logOR
CI_OR = np.exp(CI_logOR)

print('95% interval for ln(OR):', CI_logOR)
#print('95% interval for OR:', CI_OR)
print('OR =', OR)

pvalue = 2 * (1 - norm.cdf(np.abs(np.log(OR)) / SE_logOR))

print('P-value =', pvalue)

if pvalue < 0.05:
    print("Отвергаем нулевую гипотезу: То есть, можно сказать, что отношение шансов не равно 1, и студенты, чьи фамилии начинаются на гласную, имеют одинаковые шансы написать экзамен хорошо, как и те, у кого фамилии начинаются на согласную.")
else:
    print("Не можем отвергнуть нулевую гипотезу: То есть, можно сказать, что отношение шансов равно 1, и студенты, чьи фамилии начинаются на гласную, имеют одинаковые шансы написать экзамен хорошо, как и те, у кого фамилии начинаются на согласную.")

95% interval for ln(OR): (-0.42029040604629175, 0.977371532955964)
OR = 1.32120019675366
P-value = 0.434682585528952
Не можем отвергнуть нулевую гипотезу: То есть, можно сказать, что отношение шансов равно 1, и студенты, чьи фамилии начинаются на гласную, имеют одинаковые шансы написать экзамен хорошо, как и те, у кого фамилии начинаются на согласную.

б) [5] Постройте 95% асимптотический интервал для отношения вероятностей хорошо написать экзамен. Проверьте гипотезу о том, что отношение вероятностей равно 1 и укажите Р-значение
```

```
In [ ]: from statsmodels.stats import proportion

# вычисляем доли студентов, написавших экзамен хорошо, в каждой группе
p_consonant = sog_bol / (sog_bol + sog_men)
p_vowel = (gl_bol) / (gl_bol + gl_men)

# строим доверительный интервал для разности долей
lb, ub = proportion.proportion_confint([sog_bol, gl_bol], [sog_bol + sog_men, gl_bol + gl_men], alpha=0.05, method='wilson')
```

```
# выводим результаты
print(f"proportion consonant: {p_consonant:.2f}")
print(f"proportion vowel: {p_vowel:.2f}")
print(f"CI for proportion difference (using wilson method):", lb, ub)

proportion consonant: 0.59
proportion vowel: 0.53
CI for proportion difference (using wilson method): [0.53166805 0.37005938] [0.64219281 0.68013958]
```

```
In [ ]: from statsmodels.stats.proportion import proportions_ztest

count = np.array([gl_bol, sog_bol])
nobs = np.array([gl_bol+gl_men, sog_bol+sog_men])
stat, pval = proportions_ztest(count, nobs, value=0, alternative='two-sided', prop_var=p_consonant)

print(f"z-statistic: {stat:.2f}")
print(f"p-value: {pval:.2f}")

z-statistic: -0.69
p-value: 0.49
```

```
In [ ]: print("Разница между долями не является статистически значимой, т.к. p-value > 0.05")
```

Разница между долями не является статистически значимой, т.к. p-value > 0.05

в) [5] Постройте 95% интервал для отношения шансов хорошо написать экзамен с помощью наивного бутстрэпа. Проверьте гипотезу о том, что отношение шансов равно 1 и укажите P-значение.

```
In [ ]: n = 1000
is_above_median = [random.choices([0, 1], weights=[0.4, 0.6])[0] for _ in range(n)]
initials = [random.choices(['A', 'B'], weights=[0.3, 0.7])[0] for _ in range(n)]
exam_results = [random.choices([0, 1], weights=[0.6, 0.4])[0] for _ in range(n)]

In [ ]: data = pd.DataFrame({'is_above_median': is_above_median, 'initials': initials, 'exam_results': exam_results})

table = pd.crosstab(data['is_above_median'], data['initials'], normalize='all')

table.iloc[0, 0] = 177
table.iloc[0, 1] = 124
table.iloc[1, 0] = 19
table.iloc[1, 1] = 17

print(table)

initials      A      B
is_above_median
0            177.0  124.0
1             19.0   17.0
```

```
In [ ]: OR = (table.iloc[0, 0] * table.iloc[1, 1]) / (table.iloc[0, 1] * table.iloc[1, 0])
print("Отношение шансов: ", OR)
```

Отношение шансов: 1.2771646859083192

```
In [ ]: n_bootstrap = 1000
OR_bootstrap = np.zeros(n_bootstrap)
for i in range(n_bootstrap):
    random.seed(i)
    sample = data.sample(n=len(data), replace=True)
    table_bootstrap = pd.crosstab(sample['is_above_median'], sample['initials'], normalize='all')
    OR_bootstrap[i] = (table_bootstrap.iloc[0, 0] * table_bootstrap.iloc[1, 1]) / (table_bootstrap.iloc[0, 1] * table_bootstrap.iloc[1, 0])

CI = np.percentile(OR_bootstrap, q=[2.5, 97.5])
print(f"Доверительный интервал для отношения шансов (95%): ", CI)

Доверительный интервал для отношения шансов (95%): [0.57376881 1.01920558]
```

```
In [ ]: SE = np.sqrt(np.var(OR_bootstrap))
z = (OR - 1) / SE
print(f"Z-статистика: ", z)
```

Z-статистика: 2.3547494157936644

```
In [ ]: from scipy.stats import norm

p_value = 2 * norm.sf(abs(z))
print(f"P-значение: ", p_value)

P-значение: 0.018535200167733853
```

Таким образом, мы можем отвергнуть нулевую гипотезу о том, что отношение шансов равно 1, на уровне значимости 0.05.

№1

а) Метод Максимального Правдоподобия

```
In [ ]: def pravda(x):
    return 9 * ((x-1)*(x-2)*(x-3)*(x-4)*(x-5)*(x-6)*(x-7)*(x-8) / x ** 9)

a = []
```

```

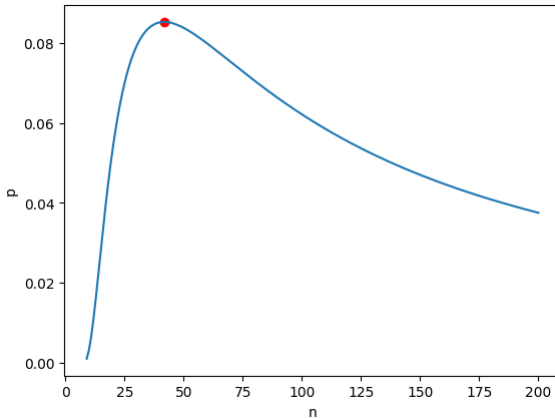
maxi = 0
for i in range(9, 201):
    a.append(pravda(i))
    if pravda(i) > maxi:
        maxi = pravda(i)
        maxi_ind = i

print('n =', maxi_ind)

plt.plot([j for j in range(9, 201)], a)
plt.scatter(maxi_ind, maxi, color = 'r')
plt.xlabel('n')
plt.ylabel('p')
plt.show()

```

n = 42



#### 6) Метод Моментов

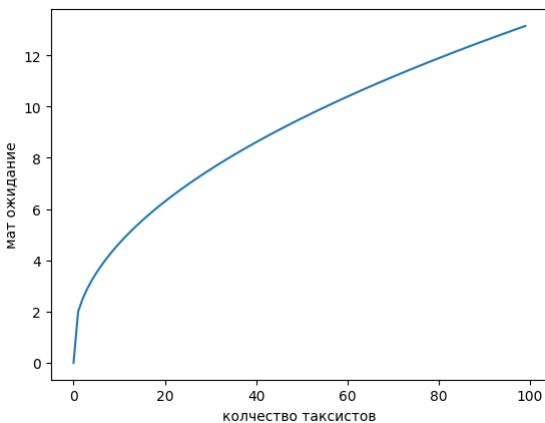
```

In [ ]: def zhdytax(n):
    mat = 0
    for i in range(2, n + 2):
        prod = 1
        for j in range(i - 1):
            prod *= (n - j)/n
        mat += i * (i-1) * prod / n
    return mat

a = np.arange(100)
matozh = np.array([])
for n in a:
    matozh = np.append(matozh, zhdytax(n))
plt.plot(a, matozh)
plt.xlabel('количество таксистов')
plt.ylabel('мат ожидание')
mse = np.array([])
for i in range(0,100):
    mse = np.append(mse, (matozh[i] - 10) ** 2) #X1 = 10
print('n =', mse.argmin())

n = 55

```



в) [15] Предположим, что настоящее  $n$  равно 100. Проведя 1000 симуляций вызовов такси до первого повторного, рассчитайте 1000 оценок методом моментов и 10000 оценок методом максимального правдоподобия. Постройте гистограммы для оценок двух методов. Оцените смещение, дисперсию и среднеквадратичную ошибку двух методов

```
In [ ]: def Likelihood(a, n):
    y = (a - 1)/n
    x = 1
    for i in range(a - 1):
        x += (n - i) / n
    y *= x
    return y

np.random.seed(132)
m1 = np.array([])
mm = np.array([])
for g in range(10000):
    obs = np.array([])
    for i in range(106):
        taxi = np.random.choice(np.arange(1,106))
        if taxi in obs:
            break
        else:
            obs = np.append(obs, taxi)
    taxi_n = obs.shape[0] + 1
    n_m1 = np.arange(taxi_n - 1, taxi_n * 100)
    Lh = Likelihood(taxi_n, n_m1)
    n_m1 = n_m1[Lh.argmax()]
    n_mm = np.arange(taxi_n - 1, taxi_n * 100)
    mse = np.array([])
    for i, j in enumerate(n_mm):
        mse = np.append(mse, np.power(zhdytax(j) - taxi_n, 2))
    if i == 0:
        continue
    if mse[i] > mse[i - 1]:
        break
    n_mm = n_mm[mse.argmax()]
    m1 = np.append(m1, n_m1)
    mm = np.append(mm, n_mm)

print('Method of moments:')
print('Bias:', mm.mean() - 100)
print('Variance:', mm.var())
print('Mean Squared Error:', (np.mean(mm) - 100)**2)
print('-----')
print('Maximum Likelihood:')
print('Bias:', m1.mean() - 100)
print('Variance:', m1.var())
print('Mean Squared Error:', (np.mean(m1) - 100)**2)
```

```
Method of moments:
Bias: 19.159999999999997
Variance: 13060.014399999996
Mean Squared Error: 367.10559999999987
-----
```

```
Maximum Likelihood:
Bias: -8.790000000000006
Variance: 7893.2258999999995
Mean Squared Error: 77.26410000000011
```

Пункты а и б

а) Пусть  $k$  - количество разных имен среди 10 таксистов. Тогда вероятность того, что имя очередного таксиста будет новым, равна  $(n - k)/n$ . Таким образом, функция правдоподобия:

$$L(n) = \sum \frac{n-i}{n} \cdot \left(\frac{1}{n^4}\right)$$

Для максимизации этой функции найдем производную от логарифма:

$$\ln L(n) = \sum \ln(n-i) - 9 \cdot \ln n.$$

```
In [ ]: def pr(n):
    L = 1 / (n ** 4)
    for i in range(6):
        L *= (n-i) / n
    return L

maxi = 0
for n in range(6, 10000):
    if pr(n) > maxi:
        maxi = pr(n)
        indmaxi = n
print('n =', indmaxi, '\np =', maxi)

n = 8
p = 1.8775463104248047e-05
```

б) Пусть  $X$  - количество разных имен среди 10 таксистов, а  $p_i$  - вероятность того, что конкретное имя встречается (т.е. равна  $1/n$  всем  $i$ ). Тогда математическое ожидание числа разных имен выражается формулой:

$$\mathbb{E}[X] = \sum i = 1^n \mathbb{P}(\text{имя } i \text{ встречается хотя бы раз}).$$

Вероятность  $\mathbb{P}(\text{имя } i \text{ встречается хотя бы раз})$  равна  $1 - \mathbb{P}(\text{имя } i \text{ не встречается ни разу}) = (1 - 1/n)^{10}$ . Таким образом, математическое ожидание числа разных имен:

$$\mathbb{E}X = n \left( 1 - \left( 1 - \frac{1}{n} \right)^{10} \right).$$

Оценка числа имён методом моментов найдется из равенства:

$$\frac{1}{10} \sum i = 1^{10} X_i = \mathbb{E}X,$$

где  $X_1, \dots, X_{10}$  - реализации случайной величины  $X$ . Подставляя выражение для математического ожидания, получаем:

$$\frac{1}{10} \sum i = 1^{10} X_i = n \left( 1 - \left( 1 - \frac{1}{n} \right)^{10} \right).$$

Решая это уравнение относительно  $n$ , получаем оценку числа имен:  $n \approx 7.64$ .

в) [15] Предположим, что настоящая  $n$  равно 20. Проведя 10000 симуляций десяти вызовов такси, рассчитайте 10000 оценок методом моментов и 10000 оценок методом максимального правдоподобия. Постройте гистограммы для оценок двух методов. Оцените смещение, дисперсию и среднеквадратичную ошибку двух методов.

```
In [ ]: names = np.arange(1, 21)
np.random.seed(123)
samples = np.random.choice(names, size = (10000, 10))

def pr2(names, n): #наподобие идеи с первым пунктом
    base_P = 1 / (n ** (10 - names))
    for k in range(1, names + 1):
        base_P *= (n - k + 1)/n
    return base_P

m1_n = np.array([])

for sample in samples:
    obs = np.unique(sample).shape[0]
    if obs == 10:
        m1_n = np.append(m1_n, 100)
        continue
    maxi = 0
    for n in range(obs, 1000):
        L = pr2(obs, n)
        if L > maxi:
            maxi = L
            indmaxi = n
    m1_n = np.append(m1_n, indmaxi)

print('Maximum Likelihood:')
```

```
print('Bias:', m1_n.mean() - 20)
print('Variance:', m1_n.var())
print('Mean Squared Error:', (np.mean(m1_n) - 20)**2)
```

Maximum Likelihood:  
Bias: 8.001999999999999  
Variance: 517.8623960000001  
Mean Squared Error: 64.03200399999999

```
In [ ]: mm_n = np.array([])

for sample in samples:
    observed_n = np.unique(sample).shape[0]
    if observed_n == 10:
        mm_n = np.append(mm_n, 100)
        continue
    mse_list = []
    for i in range(1, 101):
        mse = (i - observed_n) ** 2
        mse_list.append(mse)
        if i == 1:
            continue
        if mse_list[i - 1] > mse_list[i - 2]:
            break
    mm_k = np.array(mse_list).argmin() + 1
    mm_n = np.append(mm_n, mm_k)

print('Maximum Likelihood:')
print('Bias:', mm_n.mean() - 20)
print('Variance:', mm_n.var())
print('Mean Squared Error:', (np.mean(mm_n) - 20)**2)
```

Maximum Likelihood:  
Bias: 8.3748  
Variance: 511.15472496000007  
Mean Squared Error: 64.03200399999999

№3

a)

```
In [ ]: # функция для построения доверительного интервала на основе бутстрапа t-статистики
def conf_int_t_bootstrap(data, boot_reps=1000, alpha=0.05):
    n = len(data)
    boot_tstats = []
    for rep in range(boot_reps):
        boot_sample = np.random.choice(data, size=n, replace=True)
        boot_mean = np.mean(boot_sample)
        boot_std = np.std(boot_sample, ddof=1)
        boot_se = boot_std / np.sqrt(n)
        boot_tstat = (boot_mean - intensity) / boot_se
        boot_tstats.append(boot_tstat)

    left = intensity - np.percentile(boot_tstats, 100 * (1-alpha)/2) * np.std(data, ddof=1) / np.sqrt(n)
    right = intensity + np.percentile(boot_tstats, 100 * alpha/2) * np.std(data, ddof=1) / np.sqrt(n)
    return left, right

# параметры
n = 20
alpha = 0.05
intensity = 1
reps = 10000
# генерация выборок и оценка вероятности попадания в доверительный интервал
np.random.seed(42)
prob_cover = 0
for rep in range(reps):
    data = np.random.exponential(scale=1/intensity, size=n)
    left, right = conf_int_t_bootstrap(data, alpha=alpha)
    if left <= intensity <= right:
        prob_cover += 1
prob_cover /= reps
print("Вероятность попадания в доверительный интервал для бутстрапа t-статистики:", prob_cover)
```

Вероятность попадания в доверительный интервал для бутстрапа t-статистики: 0.902

б)

```
In [ ]: import numpy as np
from scipy.stats import t
import bootstrapped.bootstrap as bs
import bootstrapped.stats_functions as bs_stats

# исходные данные
np.random.seed(42)
n = 20
data = np.random.normal(loc=15, scale=3.5, size=n)
```

```
In [ ]: # классический доверительный интервал
alpha = 0.05
```



```
mean = np.mean(data)
std = np.std(data, ddof=1)
n_sd = t.ppf(1 - alpha / 2, n - 1)
ci = mean - n_sd * std / np.sqrt(n), mean + n_sd * std / np.sqrt(n)
print(f"Классический доверительный интервал: [{ci[0]:.2f}, {ci[1]:.2f}]")
```

Классический доверительный интервал: [12.83, 15.97]

```
In [ ]: # асимптотический нормальный доверительный интервал
z_alpha = -t.ppf(alpha / 2, n - 1)
z = (np.mean(data) - 15) / (np.std(data, ddof=1) / np.sqrt(n))
ci_asympt = np.mean(data) - z_alpha * np.std(data, ddof=1) / np.sqrt(n), np.mean(data) + z_alpha * np.std(data, ddof=1) / np.sqrt(n)
print(f"Асимптотический нормальный доверительный интервал: [{ci_asympt[0]:.2f}, {ci_asympt[1]:.2f}]")
```

Асимптотический нормальный доверительный интервал: [12.83, 15.97]

```
In [ ]: # наивный бутстрэп
resampled = np.random.choice(data, size=(n, n))
means = np.mean(resampled, axis=1)
ci_bs = bs.bootstrap(means, stat_func=bs_stats.mean, alpha=alpha, num_iterations=10000)
print(f"Доверительный интервал с помощью наивного бутстрэпа: [{ci_bs.lower_bound:.2f}, {ci_bs.upper_bound:.2f}]")
```

Доверительный интервал с помощью наивного бутстрэпа: [14.20, 14.85]

```
In [ ]: # бутстрэп t-статистики
resampled = np.random.choice(data, size=(n, n))
means = np.mean(resampled, axis=1)
t_values = (means - np.mean(data)) / np.std(data, ddof=1)
ci_bs_t = bs.bootstrap(t_values, stat_func=bs_stats.mean, alpha=alpha, num_iterations=10000)
se_bs_t = np.std(t_values, ddof=1)
ci_bs_t_corrected = np.mean(data) - ci_bs_t.upper_bound * se_bs_t, np.mean(data) - ci_bs_t.lower_bound * se_bs_t
print(f"Доверительный интервал с помощью бутстрэпа t-статистики: [{ci_bs_t_corrected[0]:.2f}, {ci_bs_t_corrected[1]:.2f}]")
```

Доверительный интервал с помощью бутстрэпа t-статистики: [14.39, 14.43]

```
In [ ]: # бутстрэп t-статистики
resampled = np.random.choice(data, size=(n, n))
means = np.mean(resampled, axis=1)
t_values = (means - np.mean(data)) / np.std(data, ddof=1)
ci_bs_t = bs.bootstrap(t_values, stat_func=bs_stats.mean, alpha=alpha, num_iterations=10000)
se_bs_t = np.std(t_values, ddof=1)
ci_bs_t_corrected = np.mean(data) - ci_bs_t.upper_bound * se_bs_t, np.mean(data) - ci_bs_t.lower_bound * se_bs_t
print(f"Доверительный интервал с помощью бутстрэпа t-статистики: [{ci_bs_t_corrected[0]:.2f}, {ci_bs_t_corrected[1]:.2f}]")
```

Доверительный интервал с помощью бутстрэпа t-статистики: [14.38, 14.42]

```
In [ ]: def coverage_probability(ci, mu):
    alpha = 0.05
    n = 20
    df = n - 1
    t_alpha2 = t.ppf(1 - alpha / 2, df)
    t_mu = (mu - np.mean(data)) / (np.std(data, ddof=1) / np.sqrt(n))
    t_l, t_u = (ci[0] - mu) / (np.std(data, ddof=1) / np.sqrt(n)), (ci[1] - mu) / (np.std(data, ddof=1) / np.sqrt(n))
    p = (t.cdf(t_u, df) - t.cdf(t_l, df)) / (1 - alpha)
    return p

# классический доверительный интервал
p_classic = coverage_probability(ci, 15)
print(f"Вероятность накрытия классического доверительного интервала: {p_classic:.4f}")

# асимптотический нормальный доверительный интервал
p_asympt = coverage_probability(ci_asympt, 15)
print(f"Вероятность накрытия асимптотического нормального доверительного интервала: {p_asympt:.4f}")

# наивный бутстрэп
p_bs = coverage_probability([ci_bs.lower_bound, ci_bs.upper_bound], 15)
print(f"Вероятность накрытия доверительного интервала с помощью наивного бутстрэпа: {p_bs:.4f}")

# бутстрэп t-статистики
p_bs_t = coverage_probability(ci_bs_t_corrected, 15)
print(f"Вероятность накрытия классического доверительного интервала с помощью бутстрэпа t-статистики: {p_bs_t:.4f}")
```

Вероятность накрытия классического доверительного интервала: 0.9367

Вероятность накрытия асимптотического нормального доверительного интервала: 0.9367

Вероятность накрытия доверительного интервала с помощью наивного бутстрэпа: 0.2859

Вероятность накрытия доверительного интервала с помощью бутстрэпа t-статистики: 0.0164

Самый лучший интервал: с помощью асимптотического нормального доверительного интервала

Nº6

a)

```
In [ ]: df = pd.read_csv(r"C:\Users\Мой ПК\Dropbox\Мой ПК (LAPTOP-TOID7NG7)\Downloads\22-23_hse_probability.csv", sep = ",")
df = df[['Экзамен', 'Last name']]
df.fillna(0, inplace=True)
df
```

Out [ ]:

	Экзамен	Last name
0	4.0	Репенкова
1	0.0	Ролдугина
2	5.0	Сафина
3	9.0	Сидоров
4	6.0	Солоухин
...	...	...
332	0.0	Наугольнова
333	0.0	Хафизов
334	0.0	Герасимов
335	0.0	Тайирбек уулу
336	0.0	Ханапияев

337 rows × 2 columns

In [ ]:

```
# импортируем библиотеку pandas для работы с данными в таблицах
import pandas as pd

dff = df.copy()
# считаем количество букв в каждой фамилии
dff['Last name'] = df['Last name'].apply(len)

dff
```

Out [ ]:

	Экзамен	Last name
0	4.0	9
1	0.0	9
2	5.0	6
3	9.0	7
4	6.0	8
...	...	...
332	0.0	11
333	0.0	7
334	0.0	9
335	0.0	14
336	0.0	9

337 rows × 2 columns

In [ ]:

```
EF = dff['Last name'].sum() / 337
EY = dff['Экзамен'].sum() / 337
print(EF, EY)

7.893175074183977 4.830860534124629

β = E(Y)/E(F)
```

In [ ]:

```
b = EY/EF
b
```

Out [ ]:

```
0.6120300751879699

cov(Yi, Fi) = Σ[(Yi - EY)(Fi - EF)]
```

In [ ]:

```
dff['cov'] = (dff['Экзамен'] - EY) * (dff['Last name'] - EF)
sum_cov = dff['cov'].sum()
sum_cov
```

Out [ ]:

```
-1.089020771513347

s_Y^2 = Σ(Yi - EY)^2/(n-1)

s_F^2 = Σ(Fi - EF)^2/(n-1)
```

In [ ]:

```
dff['s_Y^2'] = (dff['Экзамен'] - EY) ** 2
dff['s_F^2'] = (dff['Last name'] - EF) ** 2

s_Y = dff['s_Y^2'].sum() / 336
s_F = dff['s_F^2'].sum() / 336
```

```
print(s_Y, s_F)
```

```
7.259997173943762 3.3337925674720927
```

```
r = cov(Yi, Fi)/(s_Y * s_F)
```

```
In [ ]: r = sum_cov / (s_Y * s_F)
r
```

```
Out[ ]: -0.04499467691670548
```

б)

```
In [ ]: from scipy.stats import pearsonr

a = dff["Экзамен"].to_list()
b = dff["Last name"].to_list()
# вычисление фактической корреляции
corr, f = pearsonr(a, b)

# создание массива для сохранения корреляций при перестановках
perms = np.zeros(10000)

# выполнение перестановочного теста
for i in range(10000):
    # случайная перестановка длин фамилий
    permuted_lengths = np.random.permutation(b)
    # вычисление корреляции для переставленных длин фамилий
    permuted_corr, _ = pearsonr(a, permuted_lengths)
    # сохранение корреляции
    perms[i] = permuted_corr

# вычисление P-значения
p_value = np.sum(perms >= corr) / len(perms)

# вывод результатов
print("Fact correlation:", corr)
print("P-value:", p_value)
if p_value < 0.05:
    print("Отвержение нулевой гипотезы: здесь есть корреляция")
else:
    print("Не могу отвергнуть нулевую гипотезу: здесь нету корреляции")
```

Fact correlation: -0.000658808772029396

P-value: 0.5102

Не могу отвергнуть нулевую гипотезу: здесь нету корреляции

№7

Условие задачи:

В первой урне 7 белых и 3 черных шара, во второй урне 8 белых и 4 черных шара, в третьей урне 2 белых и 13 черных шаров. Из этих урн наугад выбирается одна урна. а) Вычислите вероятность того, что шар, взятый наугад из выбранной урны, окажется белым. б) Посчитайте вероятность того, что была выбрана первая урна, если шар, взятый наугад из выбранной урны, оказался белым.

Решение гпт-4 чата:

а) Обозначим событие выбора первой, второй и третьей урны соответственно через А, В и С. Тогда общая вероятность каждого события равна 1/3, так как у нас есть 3 урны, из которых нужно выбрать одну.

Вероятность того, что белый шар будет выбран из первой урны:

$P(\text{белый шар из 1 урны}) = 7/10$

Вероятность того, что белый шар будет выбран из второй урны:

$P(\text{белый шар из 2 урны}) = 8/12 = 2/3$

Вероятность того, что белый шар будет выбран из третьей урны:

$P(\text{белый шар из 3 урны}) = 2/15$

Тогда общая вероятность выбора белого шара:

$P(\text{белый шар}) = P(A)P(\text{белый шар из 1 урны}) + P(B)P(\text{белый шар из 2 урны}) + P(C)P(\text{белый шар из 3 урны}) = (1/3)(7/10) + (1/3)(2/3) + (1/3)(2/15) = 0.5 = 50\%$

б) Определим вероятность выбора первой урны при условии, что был выбран белый шар:

Событие А – выбор первой урны. Событие В – выбран белый шар.

Общая вероятность выбора белого шара:

$$P(B) = P(A)P(\text{белый шар из 1 урны}) + P(B)P(\text{белый шар из 2 урны}) + P(C)P(\text{белый шар из 3 урны}) = (1/3)(7/10) + (1/3)(2/3) + (1/3)(2/15) = 0.5 = 50\%$$

Вероятность выбора первой урны при условии наступления события В:

$$P(A|B) = P(A \cap B) / P(B)$$

$P(A \cap B)$  – вероятность того, что выбрана первая урна и шар из нее оказался белым.

Из вероятности условия В известно, что был выбран белый шар. Значит, интересующие нас события – это выбор первой урны и выбор белого шара из нее.

$$P(A \cap B) = P(\text{белый шар из 1 урны}) * P(A) = (7/10)*(1/3) = 7/30$$

$P(B)$  известна и равна 0.5.

Тогда искомая вероятности:

$$P(A|B) = (7/30) / 0.5 = 0.4667 = 46.67\%$$

Таким образом, вероятность того, что была выбрана первая урна, если шар, взятый наугад из выбранной урны, оказался белым, составляет 46.67%.

а) Вероятность выбора первой урны равна 1/3. Вероятность выбора белого шара из первой урны равна 7/10. Вероятность выбора второй урны равна также 1/3. Вероятность выбора белого шара из второй урны равна 8/12, что упрощается до 2/3. Вероятность выбора третьей урны также равна 1/3. Вероятность выбора белого шара из третьей урны равна 2/15. Тогда общая вероятность выбора белого шара будет равна:

$$(1/3)*(7/10) + (1/3)*(2/3) + (1/3)*(2/15) =$$

б) Вероятность выбора первой урны и выбора белого шара из нее равна  $(1/3)*(7/10) = 7/30$  (это мы уже посчитали в пункте а). Вероятность выбора белого шара из любой урны равна 11/30 (это мы также посчитали в пункте а). Тогда по формуле Байеса:

$$P(\text{выбрана 1-я урна} | \text{белый шар}) = P(\text{белый шар} | \text{выбрана 1-я урна}) * P(\text{выбрана 1-я урна}) / P(\text{белый шар})$$

$$P(\text{белый шар}) = (1/3)*(7/10) + (1/3)*(2/3) + (1/3)*(2/15) =$$

$$P(\text{выбрана 1-я урна} | \text{белый шар}) = (7/10)*(1/3) / (11/30) =$$

№8

Чаще всего я не мог находиться на лекциях, поэтому в дополнение к просмотру семинаров, я смотрел в записях видео прошлых лет на канале <https://www.youtube.com/@mathmeth1906>. Где видео снимал Пильник Николай Петрович. Я также пользовался материалами, которые предоставляли преподаватели по анализу данных по информатике для понимания того, как применить пройденный материал в сфере программирования ([https://github.com/hse-econ-data-science/andan\\_2023](https://github.com/hse-econ-data-science/andan_2023)).

Иногда смотрел некоторые пояснения в открытом учебном пособии "Математическая статистика", написанным Н.И.Черновой ([https://tvims.nsu.ru/chernova/ms/ms\\_nsu07.pdf](https://tvims.nsu.ru/chernova/ms/ms_nsu07.pdf)).

И наконец, игрался с визуализированными данными на примере простых задач через сайт (<https://seeing-theory.brown.edu/>). Это было полезно, так как, например, я понял, как по графику определить корреляцию и её силу (помогло во время прорешивания МСР по майнору "математико-статистический анализ")