

Домашнее задание 2

Выполнила Стогова Анастасия, БЭК213

In [147...]

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

Задача 1

a)

Пусть X - случайная величина, показывающая, на каком заказе к туристу в первый раз приедет таксист, который уже раньше приезжал к нему, тогда:

$$P(X = 2) = \frac{1}{n}$$

$$P(X = 3) = \frac{n-1}{n} \cdot \frac{2}{n}$$

...

$$P(X = 10) = \frac{n-1}{n} \cdot \frac{n-2}{n} \cdot \dots \cdot \frac{n-8}{n} \cdot \frac{9}{n} = \frac{9(n-1)\dots(n-8)}{n^9} - \text{искомая функция правдоподобия}$$

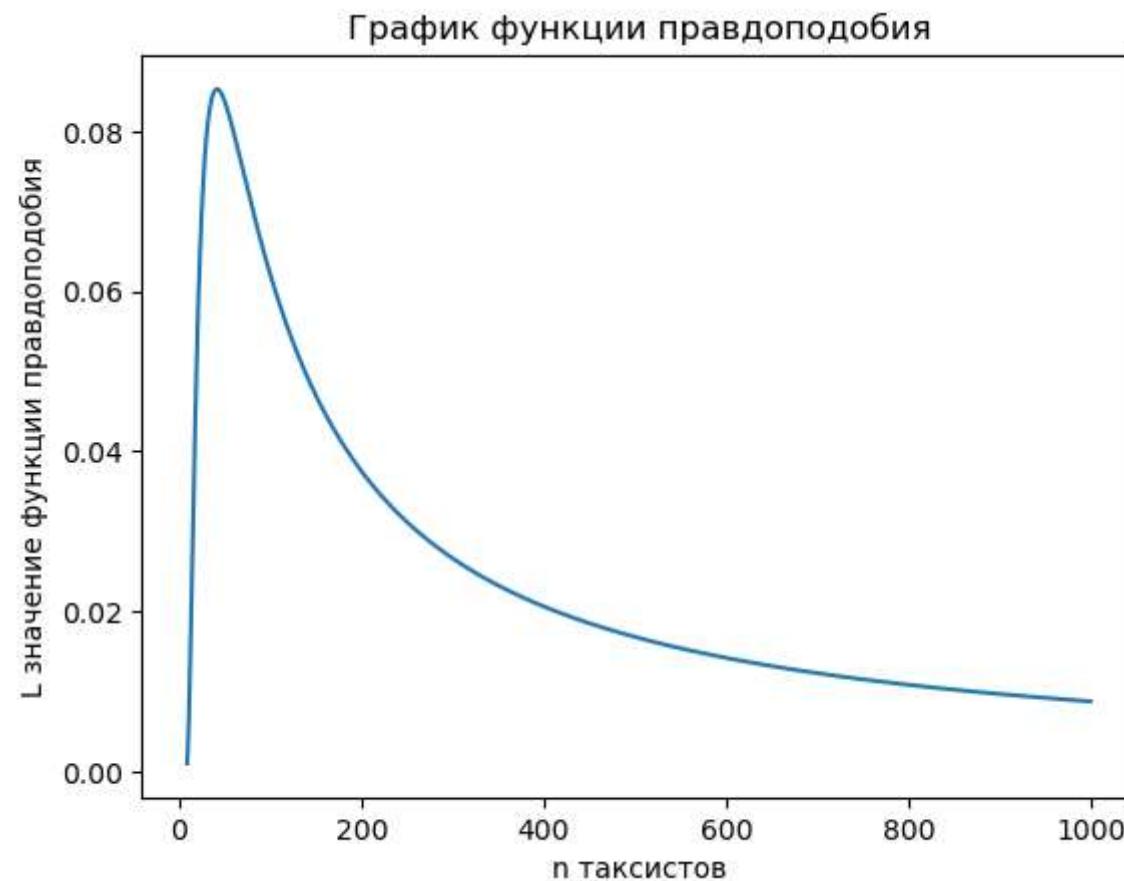
In [2]:

```
def P(x):
    P = 9 / x
    for i in range(1, 9):
        P = P*(x - i)/ x
    return P

ns = np.arange(9, 1000)
```

```
plt.plot(ns, P(ns))
plt.xlabel('n таксистов')
plt.ylabel('L значение функции правдоподобия')
plt.title('График функции правдоподобия')
```

Out[2]: Text(0.5, 1.0, 'График функции правдоподобия')



Найдем оценку методом максимального правдоподобия:

$$\ln L = \ln 9 + \sum_{i=1}^8 \ln(n-i) - 9 \ln n$$

$$\frac{\partial \ln L}{\partial n} = \sum_{i=1}^8 \frac{1}{\hat{n}_{ML} - i} - \frac{9}{\hat{n}_{ML}} = 0$$

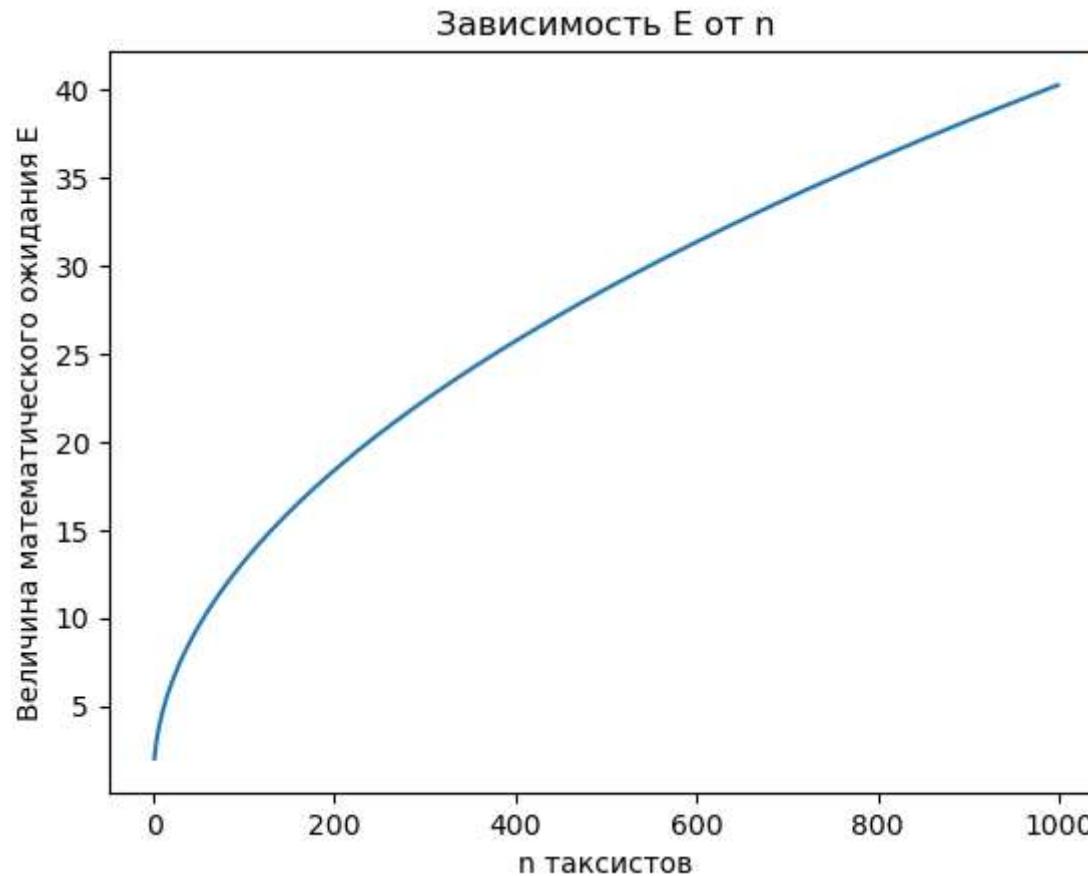
$$\frac{\partial^2 \ln L}{\partial n^2} = - \sum_{i=1}^8 \frac{1}{(n-i)^2} + \frac{9}{n^2} < 0$$

С помощью WolframAlpha решаем уравнение и получаем, что $\hat{n}_{ML} = 42$.

6)

```
In [3]: def expectation(n):
    expect = 0
    prob = 1
    for i in range(2, n + 2):
        expect += i * (i - 1) * prob / n
        prob = prob * (n - (i - 1)) / n
    return expect
ns = np.arange(1, 1000)
expectations = np.array([expectation(n) for n in ns])
```

```
In [4]: plt.plot(ns, expectations);
plt.xlabel('n таксистов')
plt.ylabel('Величина математического ожидания E')
plt.title('Зависимость E от n');
```



Найдем оценку n методом моментов:

```
In [5]: print('n =', ns[np.argmin(abs(np.array(expectations)-10))])  
n = 55
```

в)

```
In [6]: np.random.seed(42)  
  
ns = range(1, 201) # Диапазон общего количества таксистов  
  
ts = []  
for i in range(10000): # Генерируем 10000 вызовов такси до первого повторяющегося
```

```
repeated = False
while not repeated:
    taxi = np.random.choice(ns, 1)
    if taxi not in t:
        t.append(taxi)
    else:
        ts.append(len(t) + 1) # Сохраняем номер повторного
        repeated = True

def p(x, n): # Функция, которая выводит вероятность повторного таксиста на x-ый раз
    return ((x - 1) * math.factorial(n - 1) / math.factorial(n - x + 1)) / n ** (x - 1)

ps = []
ml = []
for x in ts:
    prob = [p(x, n) for n in range(x - 1, 200)]
    ps.append(max(prob)) # Находим 10000 вероятностей
    ml.append([i for i in range(x - 1, 200)][prob.index(max(prob))])

plt.hist(ml, bins=30)
plt.xlabel('n_ml - оценка методом максимального правдоподобия общего количества таксистов')
plt.ylabel('count')
plt.title('Гистограмма оценок методом максимального правдоподобия')
plt.show()
```



Задача 2

a)

In [7]:

```
import numpy as np
import itertools
import matplotlib.pyplot as plt

def calculate_likelihood(n, u, d):
    likelihood = 1
    for i in range(1, u):
        likelihood *= (n - i) / n

    combinations = itertools.combinations_with_replacement(range(1, u + 1), r=d - u)
```

```

sum_of_products = np.sum([np.prod(t) for t in list(combinations)])
likelihood *= sum_of_products / (n ** (d - u))

return likelihood

def find_maximum_likelihood_estimate(u, d):
    ns = range(u, 101)
    likelihoods = [calculate_likelihood(n, u, d) for n in ns]
    best_n = ns[np.argmax(likelihoods)]

    return best_n

def calculate_expected_value(n, d):
    expected_value = 0
    for u in range(d + 1):
        expected_value += calculate_likelihood(n, u, d) * u

    return expected_value

def find_method_of_moments_estimate(x):
    ns = range(1, 101)
    expected_values = [calculate_expected_value(n, 10) for n in ns]
    best_n = ns[np.argmin(np.abs(np.array(expected_values) - x))]

    return best_n

```

In [8]:

```

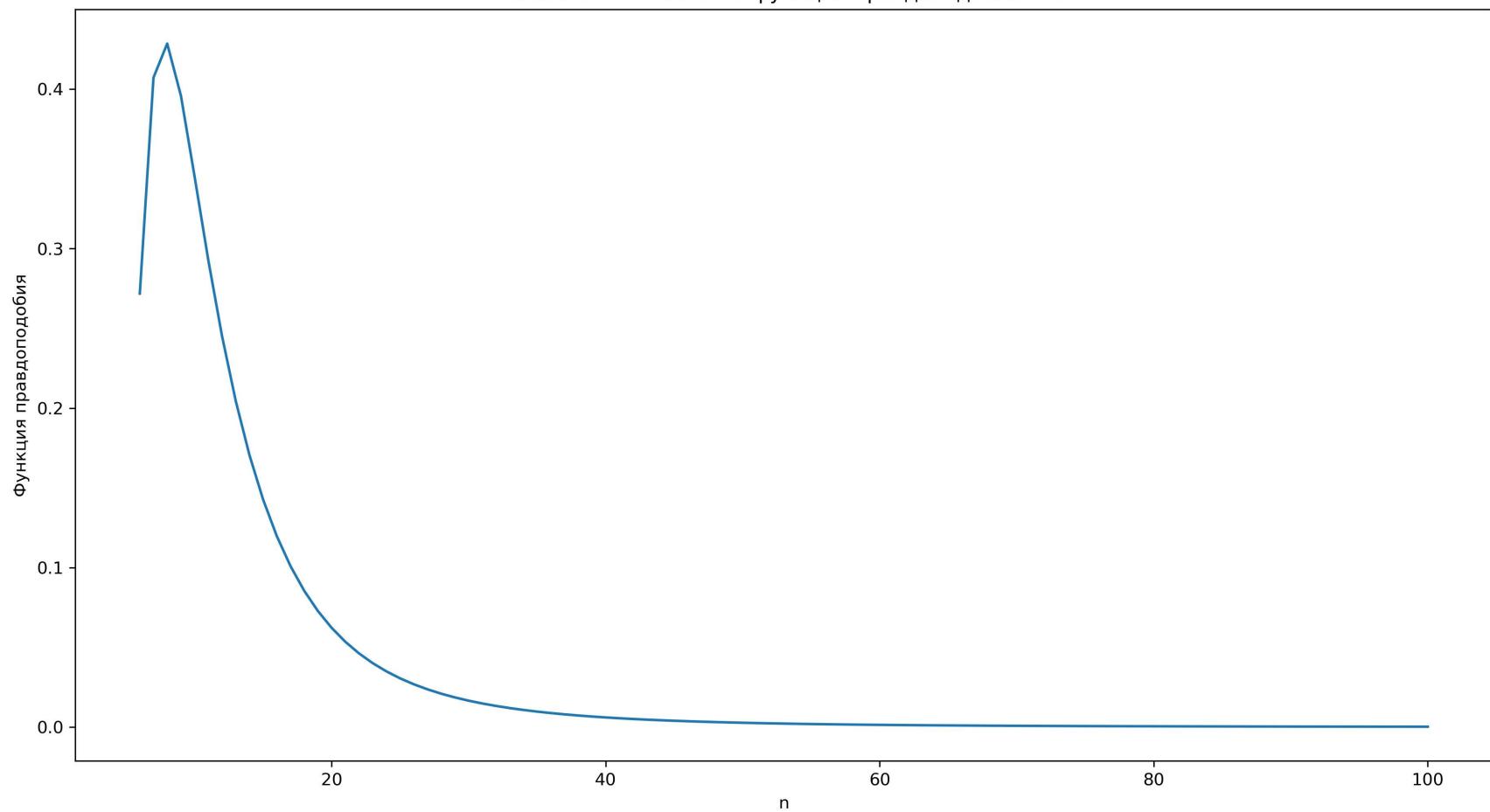
ns = range(6, 101)
likelihoods = [calculate_likelihood(n, 6, 10) for n in ns]

plt.figure(figsize=(15, 8), dpi=300)
plt.plot(ns, likelihoods)
plt.title('Зависимость величины функции правдоподобия от n')
plt.xlabel('n')
plt.ylabel('Функция правдоподобия')

ml_estimate = find_maximum_likelihood_estimate(6, 10)
print('Оценка числа n методом максимального правдоподобия:', ml_estimate)

```

Оценка числа n методом максимального правдоподобия: 8

Зависимость величины функции правдоподобия от n 

6)

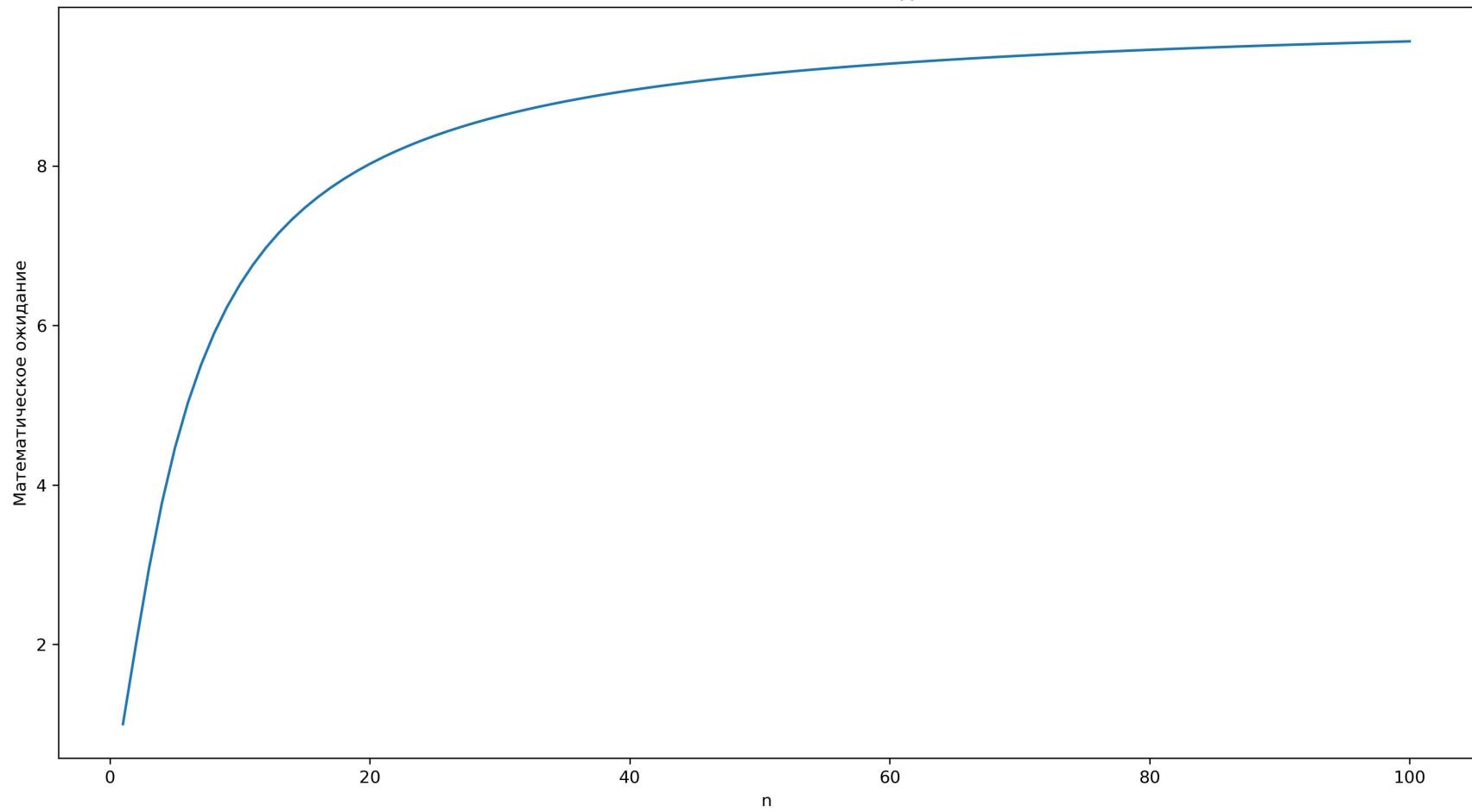
```
In [9]: ns = range(1, 101)
expected_values = [calculate_expected_value(n, 10) for n in ns]

plt.figure(figsize=(15, 8), dpi=300)
plt.plot(ns, expected_values)
plt.title('Зависимость математического ожидания от n')
plt.xlabel('n')
plt.ylabel('Математическое ожидание')

mm_estimate = find_method_of_moments_estimate(6)
print('Оценка числа n методом моментов:', mm_estimate)
```

Оценка числа n методом моментов: 8

Зависимость математического ожидания от n



В)

```
In [10]: from sklearn.metrics import mean_absolute_error, mean_squared_error

def generate_samples(n, d, num_samples):
    samples = []
    np.random.seed(3)
    taxi = np.random.choice(range(1, n + 1), size=(num_samples, d))
    for i in taxi:
        samples.append(len(set(i)))
```

```

    return samples

def calculate_expected_value(n, d):
    expected_value = 0
    for u in range(d + 1):
        expected_value += L(n, u, d) * u
    return expected_value

def find_ML(u, d):
    ns = range(u, 101)
    Ls = []
    for n in ns:
        Ls.append(L(n, u, d))
    best_n = ns[np.argmax(Ls)]
    return best_n

def find_MM(x):
    ns = range(1, 101)
    return ns[np.argmin(np.array(expected_values) - x)]]

def L(n, u, d):
    L = 1
    for i in range(1, u):
        L *= (n - i) / n
    combinations = itertools.combinations_with_replacement(range(1, u + 1), r=d - u)
    summa = np.sum([proizved(t) for t in list(combinations)])
    L *= summa / (n ** (d - u))
    return L

def proizved(tuple):
    proizv = 1
    for i in tuple:
        proizv *= i
    return proizv

# Генерация выборки
true_n = 20
num_samples = 10000
samples = generate_samples(true_n, 10, num_samples)

# Расчет матожидания
expected_values = [calculate_expected_value(n, 10) for n in range(1, 101)]

```

In [11]: # Метод максимального правдоподобия
`find_ML_vec = np.vectorize(find_ML, excluded=set([1]))`

```
ML_estimates = find_ML_vec(samples, 10)

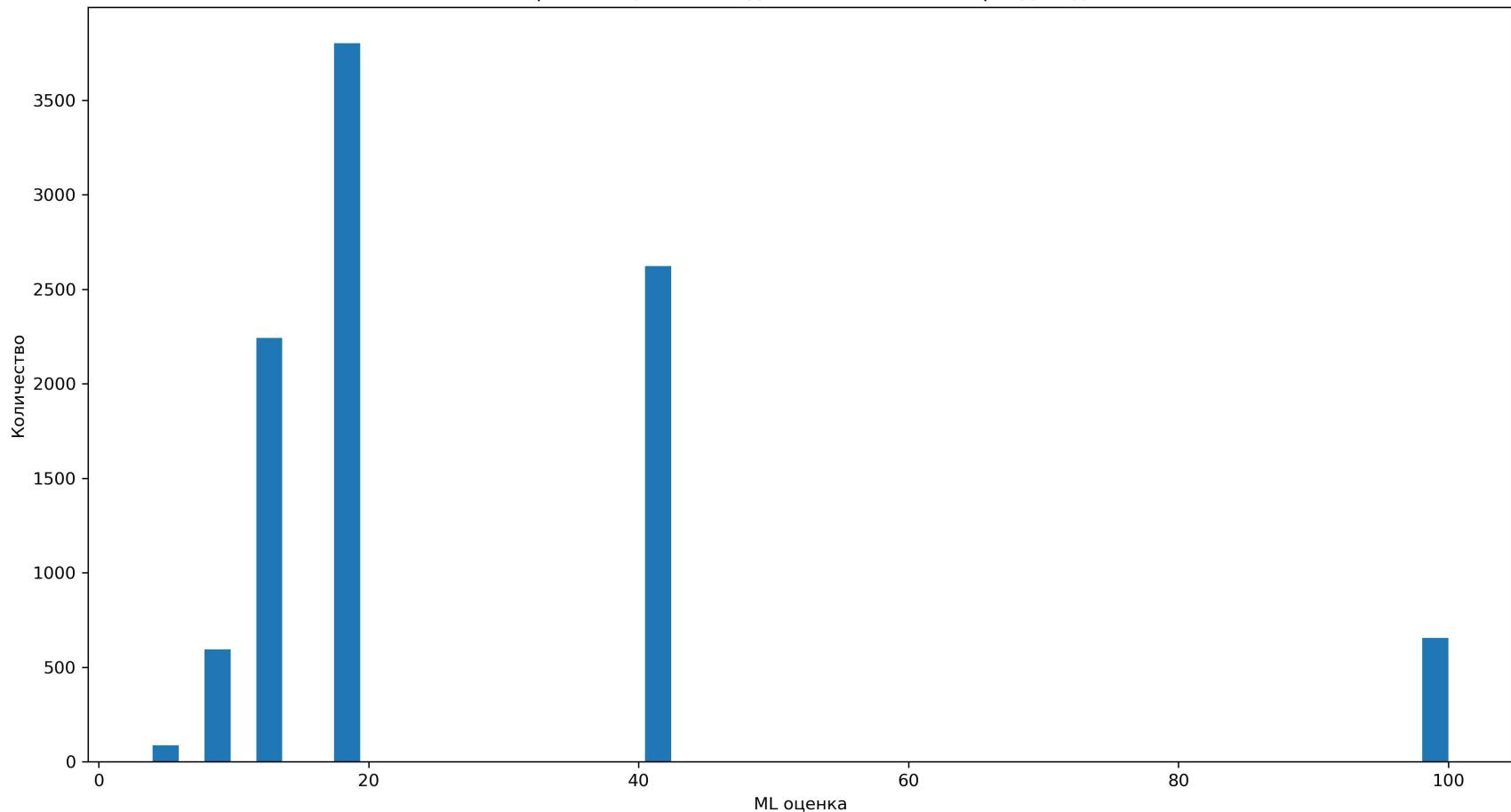
plt.figure(figsize=(15, 8), dpi=300)
plt.hist(ML_estimates, bins=50)
plt.title('Гистограмма оценок методом максимального правдоподобия')
plt.xlabel('ML оценка')
plt.ylabel('Количество')

bias = mean_absolute_error([true_n] * num_samples, ML_estimates)
variance = np.var(ML_estimates)
mse = mean_squared_error([true_n] * num_samples, ML_estimates)

print('Смещение =', bias)
print('Дисперсия =', variance)
print('Среднеквадратичная ошибка =', mse)
```

Смещение = 14.0178
Дисперсия = 506.9889109999999
Среднеквадратичная ошибка = 570.7172

Гистограмма оценок методом максимального правдоподобия



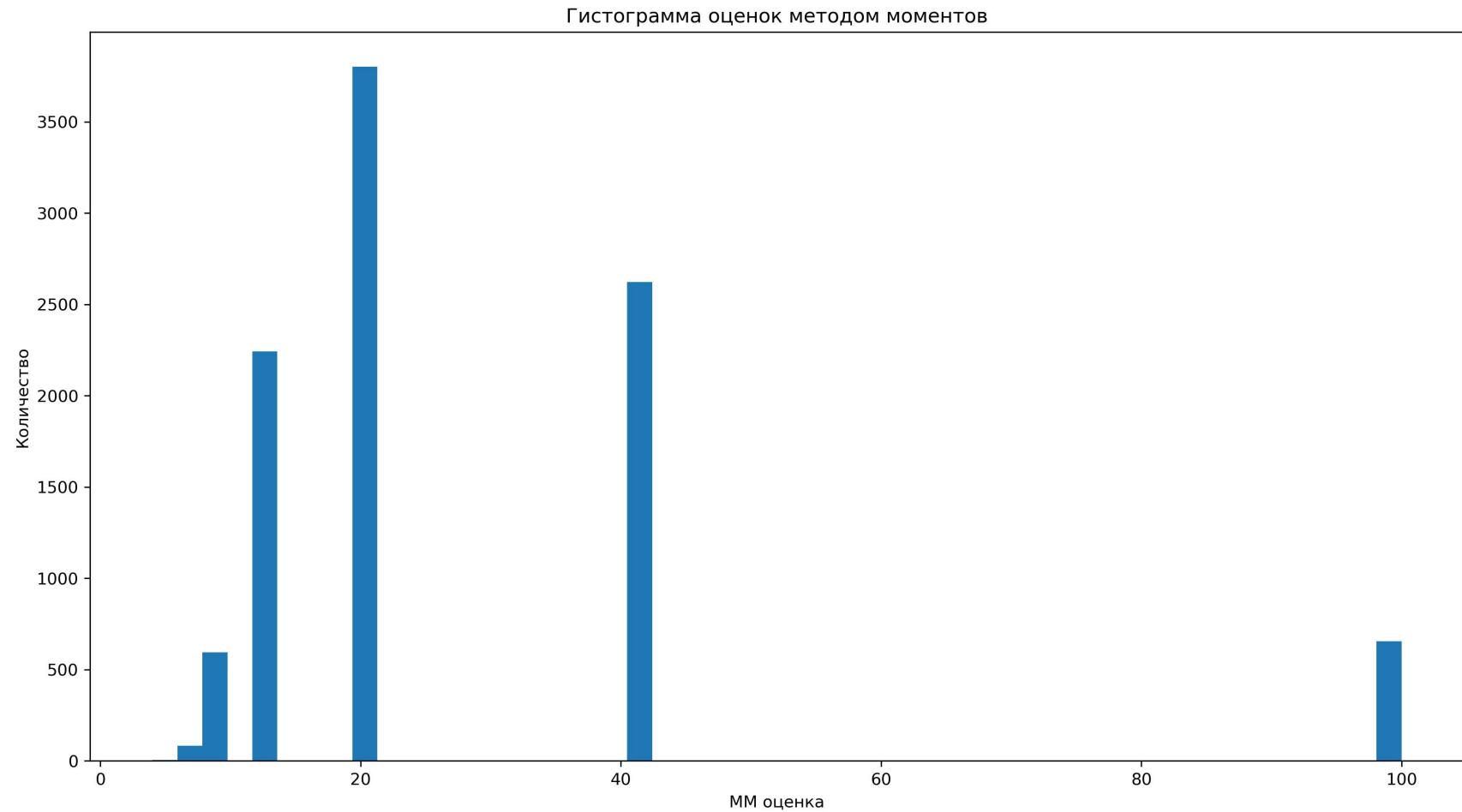
```
In [12]: # Метод моментов
find_MM_vec = np.vectorize(find_MM)
MM_estimates = find_MM_vec(samples)

plt.figure(figsize=(15, 8), dpi=300)
plt.hist(MM_estimates, bins=50)
plt.title('Гистограмма оценок методом моментов')
plt.xlabel('ММ оценка')
plt.ylabel('Количество')

bias = mean_absolute_error([true_n] * num_samples, MM_estimates)
variance = np.var(MM_estimates)
mse = mean_squared_error([true_n] * num_samples, MM_estimates)
```

```
print('Смещение =', bias)
print('Дисперсия =', variance)
print('Среднеквадратичная ошибка =', mse)
```

Смещение = 13.6294
Дисперсия = 500.01606204
Среднеквадратичная ошибка = 570.0964



Задача 3

a)

```
In [35]: np.random.seed(7645) # Задаем seed для воспроизводимости результатов

num_simulations = 10000
sample_size = 20
confidence_level = 0.95

sample_exp_3 = np.random.exponential(scale=1, size=(10000, 20))

coverage_z_test = 0

z = norm.isf(0.025)

for i in range(num_simulations):
    sample = sample_exp_3[i]
    sample_mean = np.mean(sample)
    sample_std = np.std(sample, ddof=1)
    margin_of_error = z * sample_std / np.sqrt(sample_size)
    lower_bound = sample_mean - margin_of_error
    upper_bound = sample_mean + margin_of_error
    if lower_bound <= 1 <= upper_bound:
        coverage_z_test += 1

probability_coverage = coverage_z_test / num_simulations

print("Вероятность покрытия математического ожидания с помощью классического асимптотического нормального интервала: {}")

Вероятность покрытия математического ожидания с помощью классического асимптотического нормального интервала: 0.9008
```

```
In [36]: coverage_bootstrap = 0

for i in range(num_simulations):
    sample = sample_exp_3[i]
    sample_mean = np.mean(sample)

    bootstrap_samples = np.random.choice(sample, size=(sample_size, num_simulations), replace=True)
    bootstrap_means = np.mean(bootstrap_samples, axis=0)

    lower_bound = np.percentile(bootstrap_means, (1 - confidence_level) / 2 * 100)
    upper_bound = np.percentile(bootstrap_means, (1 + confidence_level) / 2 * 100)

    if lower_bound <= 1 <= upper_bound:
        coverage_bootstrap += 1

probability_coverage = coverage_bootstrap / num_simulations
```

```
print("Вероятность покрытия математического ожидания с помощью наивного бутстрепа: {:.4f}".format(probability_coverage))
```

Вероятность покрытия математического ожидания с помощью наивного бутстрепа: 0.9010

```
In [47]: from scipy.stats import t
coverage_bootstrap = 0

for i in range(num_simulations):
    sample = sample_exp_3[i]
    sample_mean = np.mean(sample)
    bootstrap_samples = np.random.choice(sample, size=(sample_size, num_simulations), replace=True)
    bootstrap_means = np.mean(bootstrap_samples, axis=0)
    standard_error = np.std(bootstrap_means, ddof=1)
    t_critical = t.ppf(1 - (1 - confidence_level) / 2, df=sample_size - 1)
    margin_of_error = t_critical * standard_error
    lower_bound = sample_mean - margin_of_error
    upper_bound = sample_mean + margin_of_error
    if lower_bound <= 1 <= upper_bound:
        coverage_bootstrap += 1

probability_coverage = coverage_bootstrap / num_simulations

print("Вероятность покрытия математического ожидания с помощью бутстрепа t-статистики: {:.4f}".format(probability_coverage))
```

Вероятность покрытия математического ожидания с помощью бутстрепа t-статистики: 0.9104

```
In [49]: import numpy as np
from scipy.stats import t

np.random.seed(7645) # Задаем seed для воспроизводимости результатов

num_simulations = 10000
sample_size = 20
confidence_level = 0.95

sample_exp_3 = np.random.exponential(scale=1, size=(10000, 20))

coverage_t_test = 0

t_critical = t.ppf(0.975, df=3)

for i in range(num_simulations):
    sample = sample_exp_3[i]
    sample_mean = np.mean(sample)
    sample_std = np.std(sample, ddof=1)
```

```

margin_of_error = t_critical * sample_std / np.sqrt(sample_size)
lower_bound = sample_mean - margin_of_error
upper_bound = sample_mean + margin_of_error
if lower_bound <= 1 <= upper_bound:
    coverage_t_test += 1

probability_coverage = coverage_t_test / num_simulations

print("Вероятность покрытия математического ожидания с помощью t-статистики и распределения Стьюдента: {:.4f}".format(p))

```

Вероятность покрытия математического ожидания с помощью t-статистики и распределения Стьюдента: 0.9748

6)

С помощью классического асимптотического нормального интервала:

```

In [66]: import numpy as np
from scipy.stats import t

np.random.seed(42)

# Параметры симуляции
num_simulations = 10000
sample_size = 20
confidence_level = 0.95
degrees_of_freedom = 3

# Генерация выборок с распределением Стьюдента
samples = t.rvs(df=degrees_of_freedom, size=(num_simulations, sample_size))

# Вычисление выборочных средних и стандартных отклонений
sample_means = np.mean(samples, axis=1)
sample_std = np.std(samples, axis=1, ddof=1)

# Вычисление критических значений для классического асимптотического нормального интервала
t_critical = 1.96 # Для 95% доверительного уровня

# Вычисление границ доверительного интервала
lower_bound = sample_means - t_critical * (sample_std / np.sqrt(sample_size))
upper_bound = sample_means + t_critical * (sample_std / np.sqrt(sample_size))

# Определение, попадает ли истинное математическое ожидание в доверительный интервал
covered = np.logical_and(lower_bound <= 0, upper_bound >= 0)

```

```
# Вычисление доли выборок, где истинное математическое ожидание попадает в доверительный интервал
coverage_probability = np.mean(covered)

print("Оценка вероятности накрытия:", coverage_probability)
```

Оценка вероятности накрытия: 0.9438

С помощью наивного бутстрэпа:

In [289...]

```
import numpy as np
from scipy.stats import t

np.random.seed(42)

# Параметры моделирования
num_simulations = 10000
sample_size = 20
confidence_level = 0.95

# Генерация выборок из распределения Стьюдента с тремя степенями свободы
samples_student = t.rvs(df=3, size=(num_simulations, sample_size))

# Функция для расчета интервала бутстрэпа
def bootstrap_interval(data):
    sample_mean = np.mean(data)
    bootstrap_samples = np.random.choice(data, size=(sample_size, num_simulations), replace=True)
    bootstrap_sample_means = np.mean(bootstrap_samples, axis=0)
    lower_bound = np.percentile(bootstrap_sample_means, (1 - confidence_level) / 2 * 100)
    upper_bound = np.percentile(bootstrap_sample_means, (1 + confidence_level) / 2 * 100)
    return lower_bound, upper_bound

# Расчет вероятности покрытия для распределения Стьюдента с использованием наивного бутстрэпа
coverage_bootstrap = 0

for i in range(num_simulations):
    sample = samples_student[i]
    sample_mean = np.mean(sample)

    bootstrap_samples = np.random.choice(sample, size=(sample_size,), replace=True)
    bootstrap_means = np.mean(bootstrap_samples)

    lower_bound, upper_bound = bootstrap_interval(bootstrap_samples)

    if lower_bound <= sample_mean <= upper_bound:
        coverage_bootstrap += 1
```

```
print(coverage_bootstrap / num_simulations)
```

0.903

С помощью бутстрэпа t-статистики:

In [109...]

```
import numpy as np
from scipy.stats import t
import tqdm

np.random.seed(1)

num_simulations = 10000
sample_size = 20
coverage_t_test = 0

for i in tqdm.tqdm(range(num_simulations)):
    samples = t.rvs(df=3, size=sample_size)
    theta1 = samples.mean()
    bootstrap_samples = np.random.choice(samples, size=(num_simulations, sample_size), replace=True)
    bootstrap_thetas = bootstrap_samples.mean(axis=1)
    std = bootstrap_samples.std(axis=1, ddof=1)
    R = (bootstrap_thetas - theta1) / std
    left_side = theta1 - np.percentile(R, 2.5) * samples.std()
    right_side = theta1 - np.percentile(R, 97.5) * samples.std()
    if 0 < left_side and 0 > right_side:
        coverage_t_test += 1

print(coverage_t_test / num_simulations)
```

100%|██████████| 10000/10000 [01:28<00:00, 113.26it/s]

0.9192

В)

Ответ: таким образом, если случайные величины имеют экспоненциальное распределение, то лучше работает способ бутстрэпа t-статистики, а в случае с распределением Стьюдента - классический асимптотический нормальный интервал.

Задача 4

- Нулевая гипотеза (H_0): ожидаемые результаты экзамена одинаковы для групп с фамилиями, начинающимися на гласную и согласную букву.
- Альтернативная гипотеза (H_1): ожидаемые результаты экзамена различаются для групп с фамилиями, начинающимися на гласную и согласную букву.

```
In [221...]: df = pd.read_csv('дзстат.csv')
df.head(10)
```

Out[221]:

	num	Last name	Name	Mail	info_1	info_2	info_3	brip	prepod	Вариант	...	Unnamed: 71	Unnamed: 72
0	NaN	NaN	Желтый	NaN	NaN	NaN	NaN	NaN	NaN	1.0	...	NaN	
1	NaN	NaN	Голубой	NaN	NaN	NaN	NaN	NaN	NaN	2.0	...	NaN	
2	NaN	NaN	Зеленый	NaN	NaN	NaN	NaN	NaN	NaN	3.0	...	NaN	
3	NaN	NaN	Оранжевый	NaN	NaN	NaN	NaN	NaN	NaN	4.0	...	NaN	
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	
5	1	Репенкова	Полина Александровна	parepenkova@edu.hse.ru	351118027	3668903494	NaN	ип	Демешев	4.0	...	NaN	
6	2	Роддугина	Софья Александровна	saroldugina@edu.hse.ru	351118027	1705654528	NaN	ип	Демешев	NaN	...	NaN	
7	3	Сафина	Алия Линаровна	alsafina@edu.hse.ru	351118027	5096071996	NaN	ип	Демешев	3.0	...	NaN	
8	4	Сидоров	Иван Максимович	imsidorov@edu.hse.ru	351118027	4361493219	NaN	ип	Демешев	4.0	...	NaN	
9	5	Солоухин	Иван Владимирович	ivsoloukhin@edu.hse.ru	351118027	1705636488	NaN	ип	Демешев	4.0	...	NaN	

10 rows × 81 columns

```
In [222...]: df = df[['Last name', 'Unnamed: 72']][5: ].reset_index(drop=True).rename(columns={'Unnamed: 72': 'Scores'})
```

```
In [223...]: df
```

Out[223]:

	Last name	Scores
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	Сярова	0.0
330	Темиркулов	0.0
331	Эшмеев	16.0

332 rows × 2 columns

In [73]:

```
# Создаем две группы на основе критерия гласной и согласной буквы
vowel_group = df[df['Last name'].str[0].str.lower().isin(['а', 'е', 'ё', 'и', 'о', 'у', 'ы', 'э', 'ю', 'я'])]['Scores']
consonant_group = df[~df['Last name'].str[0].str.lower().isin(['а', 'е', 'ё', 'и', 'о', 'у', 'ы', 'э', 'ю', 'я'])]['Sco
```

a) Тест Уэлча:

In [200...]

```
from scipy.stats import ttest_ind
# Выполняем тест Уэлча
t_stat, p_value = ttest_ind(vowel_group, consonant_group, equal_var=False)

# Выводим результаты
print(f"p-value: {p_value}")
if p_value > 0.05:
    print('Стат. вывод: H_0 не отвергается')
else:
    print('Стат. вывод: H_0 отвергается')
```

p-value: 0.3974027153843839

Стат. вывод: H_0 не отвергается

б) Наивный бутстрэп:

```
In [219...]: df_vowel = df[df['Last name'].str.startswith(('А', 'У', 'О', 'Ы', 'И', 'Э', 'Я', 'Ю', 'Ё', 'Е'))]['Scores']
df_vowel
```

```
Out[219]: {17: 25.0, 18: 26.0, 32: 25.0, 45: 29.0, 46: 26.0, 47: 20.0, 48: 19.0, 49: 24.0, 50: 25.0, 51: 28.0, 67: 16.0, 68: 16.0, 85: 15.0, 113: 17.0, 114: 4.0, 126: 23.0, 143: 21.0, 144: 13.0, 158: 16.0, 166: 19.0, 167: 6.0, 168: 15.0, 169: 8.0, 170: 13.0, 171: 23.0, 172: 11.0, 173: 7.0, 174: 12.0, 184: 20.0, 188: 20.0, 189: 17.0, 220: 18.0, 221: 20.0, 227: 17.0, 228: 11.0, 229: 21.0, 230: 5.0, 240: 13.0, 270: 13.0, 271: 13.0, 272: 21.0, 273: 0.0, 274: 0.0, 275: 0.0, 276: 0.0}
```

```
277      0.0
299      0.0
300     22.0
331     16.0
Name: Scores, dtype: float64
```

In [220...]:

```
df_consonant = df[df['Last name'].str.startswith(['Б', 'В', 'Г', 'Д', 'Ж', 'З', 'Й', 'К', 'Л', 'М', 'Н', 'П', 'Р', 'С',
df_consonant
```

Out[220]:

0	16.0
1	0.0
2	19.0
3	26.0
4	21.0
	...
326	4.0
327	19.0
328	0.0
329	0.0
330	0.0

```
Name: Scores, Length: 283, dtype: float64
```

In [196...]:

```
vowel = np.random.choice(df_vowel, size = (10000, 49)).mean(axis = 1)
consonant = np.random.choice(df_consonant, size = (10000, 283)).mean(axis = 1)
delta = vowel - consonant
left1 = np.percentile(delta, 2.5)
right1 = np.percentile(delta, 97.5)
print(left1,right1)
```

-3.5089330785317667 1.285685440253838

In [197...]:

```
p_value = ((delta > 0).sum()/len(delta))*2
print(f'p-value = {p_value}')
if p_value > 0.05:
    print('Стат. вывод: H_0 не отвергается')
else:
    print('Стат. вывод: H_0 отвергается')
```

p-value = 0.3906
Стат. вывод: H_0 не отвергается

в) Бутстрэп t-статистики

In [164...]:

```
std = np.sqrt(np.random.choice(df_vowel, size = (10000, 49)).var(axis = 1)/49 + np.random.choice(df_consonant, size = (
gamma = (vowel - consonant - (df_vowel.mean() - df_consonant.mean()))/std
```

```
left2 = np.percentile(gamma, 2.5)
right2 = np.percentile(gamma, 97.5)
print(left2,right2)
```

-1.983753328726045 1.981332511411889

In [198]:

```
t = (df_vowel.mean() - df_consonant.mean()) / sqrt(df_vowel.var() / 49 + df_consonant.var() / 283)
p_value = 2 * (gamma < t).sum() / 10000
print(f'p-value = {p_value}')
if p_value > 0.05:
    print('Стат. вывод: H_0 не отвергается')
else:
    print('Стат. вывод: H_0 отвергается')
```

p-value = 0.3956

Стат. вывод: H_0 не отвергается

г) Перестановочный тест:

In [189]:

```
all_marks = np.append(vowel_group, consonant_group)
np.random.seed(1)
teta = []
for i in range(10000):
    sample = np.random.permutation(all_marks)
    X = sample[:len(vowel_group)]
    Y = sample[len(vowel_group):]
    delta = X.mean() - Y.mean()
    teta.append(delta)
right_side = np.percentile(np.array(teta), 97.5)
left_side = np.percentile(np.array(teta), 2.5)
right_side, left_side
```

Out[189]:

(2.345424388836806, -2.371096848633446)

In [217]:

```
p_value = 2 * (np.array(teta) < vowel_group.mean() - consonant_group.mean()).sum() / 10000
print(f'p-value = {p_value}')
if p_value > 0.05:
    print('Стат. вывод: H_0 не отвергается')
else:
    print('Стат. вывод: H_0 отвергается')
```

p-value = 0.3824

Стат. вывод: H_0 не отвергается

Задача 5

In [227]:

```
def vowel(word):
    vowels = ["а", "о", "у", "ы", "и", "е", "ё", "ю", "я", "э"]
    if word[0].lower() in vowels:
        return 1
    else:
        return 0
df['Vowel'] = df['Last name'].apply(vowel)

df['Greater than median'] = np.where(df['Scores'] >= df['Scores'].median(), 1, 0)
df
```

Out[227]:

	Last name	Scores	Greater than median	Vowel
0	Репенкова	16.0	0	0
1	Ролдугина	0.0	0	0
2	Сафина	19.0	1	0
3	Сидоров	26.0	1	0
4	Солоухин	21.0	1	0
...
327	Сенников	19.0	1	0
328	Ся	0.0	0	0
329	Сярова	0.0	0	0
330	Темиркулов	0.0	0	0
331	Эшмеев	16.0	0	1

332 rows × 4 columns

In [234]:

```
#Таблица сопряженности
contingency_table = pd.pivot_table(df, values='Scores', index='Vowel', columns='Greater than median', aggfunc='count')
contingency_table
```

Out[234]: Greater than median 0 1

Vowel	0	1
0	138	145
1	28	21

a)

In [249...]

```

import scipy.stats as stats

# Определение значений a, b, c, d из таблицы сопряженности
a = contingency_table[1][1]
b = contingency_table[1][0]
c = contingency_table[0][1]
d = contingency_table[0][0]

# Расчёт отношения шансов
odds_ratio = (a * d) / (b * c)

# Расчёт асимптотического интервала
log_odds = np.log(odds_ratio)
se = np.sqrt(1 / a + 1 / b + 1 / c + 1 / d)
lower_ci = np.exp(log_odds - 1.96 * se)
upper_ci = np.exp(log_odds + 1.96 * se)

# Вывод результатов
print(f"95% асимптотический интервал: [{lower_ci:.3f}, {upper_ci:.3f}]")

# Проверка гипотезы о равенстве отношения шансов 1
null_hypothesis = "Отношение шансов равно 1"
alternative_hypothesis = "Отношение шансов не равно 1"
p_value = 2 * (1 - stats.norm.cdf(np.abs(log_odds / se)))

print(f"Гипотеза: {null_hypothesis if p_value > 0.05 else alternative_hypothesis}")
print(f"P-значение: {p_value:.4f}")
if p_value > 0.05:
    print('Стат. вывод: гипотеза не отвергается')
else:
    print('Стат. вывод: гипотеза отвергается')

```

95% асимптотический интервал: [0.387, 1.316]

Гипотеза: Отношение шансов равно 1

Р-значение: 0.2802

Стат. вывод: гипотеза не отвергается

б)

In [260...]

```
import scipy.stats as stats

# Расчёт вероятностей хорошо написать экзамен в каждой группе
prob_good_vowel = 145/(145+148)
prob_good_consonant = 21/(21+28)

# Расчёт отношения вероятностей
probability_ratio = prob_good_vowel / prob_good_consonant

# Расчёт асимптотического интервала
log_ratio = np.log(probability_ratio)
se_ratio = np.sqrt(1 / 145 - 1/(145+148) + 1 / 21 - 1/(21+28))
lower_ci_ratio = np.exp(log_ratio - 1.96 * se_ratio)
upper_ci_ratio = np.exp(log_ratio + 1.96 * se_ratio)

# Вывод результатов
print(f"95% асимптотический интервал: [{lower_ci_ratio:.3f}, {upper_ci_ratio:.3f}]")

# Проверка гипотезы о равенстве отношения вероятностей 1
null_hypothesis = "Отношение вероятностей равно 1"
alternative_hypothesis = "Отношение вероятностей не равно 1"
p_value = 2 * (1 - stats.norm.cdf(np.abs(log_ratio / se_ratio)))

print(f"Гипотеза: {null_hypothesis if p_value > 0.05 else alternative_hypothesis}")
print(f"Р-значение: {p_value:.4f}")
if p_value > 0.05:
    print('Стат. вывод: гипотеза не отвергается')
else:
    print('Стат. вывод: гипотеза отвергается')
```

95% асимптотический интервал: [0.819, 1.628]

Гипотеза: Отношение вероятностей равно 1

Р-значение: 0.4116

Стат. вывод: гипотеза не отвергается

в)

In [274...]

```

contingency_table = pd.crosstab(df['Vowel'], df['Greater than median'])

# Расчет наблюдаемого отношения шансов (odds ratio)
observed_ratio = contingency_table.iloc[0, 0] * contingency_table.iloc[1, 1] / (contingency_table.iloc[0, 1] * contingency_table.iloc[1, 0])

# Наивный бутстрэп
np.random.seed(0) # Для воспроизводимости
n_iterations = 10000
bootstrapped_ratios = []
for _ in range(n_iterations):
    bootstrap_sample = contingency_table.sample(n=contingency_table.shape[0], replace=True)
    bootstrapped_ratio = bootstrap_sample.iloc[0, 0] * bootstrap_sample.iloc[1, 1] / (bootstrap_sample.iloc[0, 1] * bootstrap_sample.iloc[1, 0])
    bootstrapped_ratios.append(bootstrapped_ratio)

# Расчет 95% доверительного интервала
lower_ci = np.percentile(bootstrapped_ratios, 2.5)
upper_ci = np.percentile(bootstrapped_ratios, 97.5)

# Проверка гипотезы об отношении шансов равном 1
null_hypothesis = "Отношение шансов равно 1"
alternative_hypothesis = "Отношение шансов отличается от 1"
p_value = 2 * min(np.mean(bootstrapped_ratios >= observed_ratio), np.mean(bootstrapped_ratios <= observed_ratio))

# Вывод результатов
print(f"95% доверительный интервал для отношения шансов: [{lower_ci:.3f}, {upper_ci:.3f}]")
print(f"Гипотеза: {null_hypothesis if p_value > 0.05 else alternative_hypothesis}")
print(f"p-value: {p_value:.4f}")
if p_value > 0.05:
    print('Стат. вывод: гипотеза не отвергается')
else:
    print('Стат. вывод: гипотеза отвергается')

```

95% доверительный интервал для отношения шансов: [0.714, 1.401]
Гипотеза: Отношение шансов равно 1
p-value: 0.5064
Стат. вывод: гипотеза не отвергается

Задача 6

a)

In [287...]

```

import numpy as np
import pandas as pd
from scipy.stats import pearsonr
from itertools import permutations

# Преобразование значений Scores в числовой формат
df['Scores'] = pd.to_numeric(df['Scores'], errors='coerce')

# Удаление строк с непреобразуемыми значениями
df = df.dropna(subset=['Scores'])

# Оценка параметра в методом моментов
x = df['Last name'].str.len()
y = df['Scores']
beta_hat = np.mean(y) / np.mean(x)

print(f"Оценка параметра beta (\beta): {beta_hat:.4f}")

# Расчет выборочной корреляции
observed_corr, _ = pearsonr(x, y)
print(f"Выборочная корреляция: {observed_corr:.4f}")

```

Оценка параметра beta (β): 2.0613
 Выборочная корреляция: 0.0253

6)

In [288...]

```

# Перестановочный тест
np.random.seed(0) # Для воспроизводимости
n_permutations = 10000
permutation_corrs = []
for _ in range(n_permutations):
    permuted_y = np.random.permutation(y)
    permutation_corr, _ = pearsonr(x, permuted_y)
    permutation_corrs.append(permutation_corr)

# Расчет p-value
p_value = np.mean(np.abs(permutation_corrs) >= np.abs(observed_corr))

# Вывод результатов
print(f"p-value: {p_value:.4f}")
if p_value > 0.05:
    print('Стат. вывод: гипотеза не отвергается')

```

```
else:  
    print('Стат. вывод: гипотеза отвергается')
```

p-value: 0.6500
Стат. вывод: гипотеза не отвергается

Задача 7

Диалог с ChatGPT (задача 3.2 из задачника)

User:

Два охотника одновременно выстрелили в одну утку. Первый попадает с вероятностью 0.4, второй — с вероятностью 0.7 независимо от первого. а) Какова вероятность того, что в утку попала ровно одна пуля? б) Какова условная вероятность того, что утка была убита первым охотником, если в утку попала ровно одна пуля?

ChatGPT:

а) Чтобы найти вероятность того, что в утку попала ровно одна пуля, мы можем рассмотреть два случая:

Первый охотник попал, а второй промахнулся. Первый охотник промахнулся, а второй попал. Давайте вычислим вероятность каждого из этих случаев и сложим их, чтобы получить искомую вероятность.

Вероятность того, что первый охотник попадет и второй промахнется: $P(1 \text{ попал}, 2 \text{ промахнулся}) = P(1 \text{ попал}) P(2 \text{ промахнулся}) = 0.4 (1 - 0.7) = 0.4 * 0.3 = 0.12$

Вероятность того, что первый охотник промахнется, а второй попадет: $P(1 \text{ промахнулся}, 2 \text{ попал}) = P(1 \text{ промахнулся}) P(2 \text{ попал}) = (1 - 0.4) 0.7 = 0.6 * 0.7 = 0.42$

Теперь сложим эти вероятности: $P(\text{ровно одна пуля}) = P(1 \text{ попал}, 2 \text{ промахнулся}) + P(1 \text{ промахнулся}, 2 \text{ попал}) = 0.12 + 0.42 = 0.54$

Таким образом, вероятность того, что в утку попала ровно одна пуля, составляет 0.54 или 54%.

б) Чтобы найти условную вероятность того, что утка была убита первым охотником, если в нее попала ровно одна пуля, мы можем использовать формулу Байеса. Пусть А - утка была убита первым охотником, В - в утку попала ровно одна пуля.

Тогда условная вероятность $P(A|B)$, что утка была убита первым охотником при условии, что в нее попала ровно одна пуля, вычисляется по формуле: $P(A|B) = (P(A) * P(B|A)) / P(B)$

Вероятность $P(A)$ того, что утка была убита первым охотником, равна 0.5, так как оба охотника имели одинаковые шансы попасть.

Вероятность $P(B|A)$ того, что в утку попала ровно одна пуля при условии, что утку убил первый охотник, мы уже посчитали и она равна 0.12.

Вероятность $P(B)$ того, что в утку попала ровно одна пуля, мы также уже посчитали и она равна 0.54.

Подставим значения в формулу: $P(A|B) = (0.5 * 0.12) / 0.54 = 0.06 / 0.54 \approx 0.1111$

Таким образом, условная вероятность того, что утка была убита первым охотником, при условии, что в утку попала ровно одна пуля, составляет примерно 0.1111 или около 11.11%.

User:

Неправильно решен пункт 6)

ChatGPT:

Приншу извинения за путаницу. Давайте пересчитаем условную вероятность того, что утка была убита первым охотником, при условии, что в утку попала ровно одна пуля, используя формулу Байеса.

Пусть A - утка была убита первым охотником, B - в утку попала ровно одна пуля.

Вероятность $P(A)$ того, что утку убил первый охотник, равна 0.5, так как оба охотника имеют одинаковые шансы попасть.

Вероятность $P(B)$ того, что в утку попала ровно одна пуля, мы уже посчитали и она равна 0.54.

Теперь нам нужно найти вероятность $P(B|A)$, что в утку попала ровно одна пуля при условии, что утку убил первый охотник. В данной задаче охотники стреляют независимо друг от друга, поэтому вероятность попадания первого охотника не влияет на вероятность попадания второго охотника

User:

По условию, охотники имеют неравные шансы попасть, первый попадет в утку с вероятностью 0.4, а второй - с вероятностью 0.7

ChatGPT:

Приношу извинения за неверное понимание условия. Давайте пересчитаем условную вероятность того, что утка была убита первым охотником, при условии, что в утку попала ровно одна пуля, используя формулу Байеса.

Пусть A - утка была убита первым охотником, B - в утку попала ровно одна пуля.

Вероятность $P(A)$ того, что утку убил первый охотник, равна вероятности попадания первого охотника умноженной на вероятность промаха второго охотника. То есть, $P(A) = 0.4 \cdot (1 - 0.7) = 0.4 \cdot 0.3 = 0.12$.

Вероятность $P(B)$ того, что в утку попала ровно одна пуля, мы уже посчитали и она равна 0.54.

Теперь нам нужно найти вероятность $P(A|B)$, что утка была убита первым охотником при условии, что в утку попала ровно одна пуля. Используя формулу Байеса, $P(A|B) = (P(B|A) * P(A)) / P(B)$.

$P(B|A)$ - вероятность попадания ровно одной пули при условии, что утку убил первый охотник. В данном случае, $P(B|A) = 1$, так как если первый охотник убил утку, то в нее точно попала только одна пуля.

Тогда $P(A|B) = (1 * 0.12) / 0.54 = 0.12 / 0.54 \approx 0.2222$.

Таким образом, условная вероятность того, что утка была убита первым охотником, при условии, что в утку попала ровно одна пуля, составляет примерно 0.2222 или около 22.22%.

User:

Все верно, молодец!

ChatGPT:

Спасибо! Рад, что смог помочь. Если у вас есть еще вопросы, не стесняйтесь задавать. Я всегда готов помочь.

Задача 8

https://github.com/hse-econ-data-science/andan_2023 - очень помогают занятия по анализу данных

<https://www.youtube.com/@3blue1brown> - канал 3Blue1Brown, смотрела видео по ЦПТ

In []: