Домашнее задание по математической статистике

Мамонтова Дарья БЭК211

In [1]:
```
import scipy.stats as sts
import numpy as np
import pandas as pd
import math
import copy
import random
import matplotlib.pyplot as plt
import tqdm
```

Задача 1

пункт а

Из-за того, что таксисты независимы друг друг от друга, то пусть k-номер такси по счёту, на котором мы встретим одного и того же таксиста, тогда:

$P(X = 2) = \frac{1}{n}$ и так далее.

В общем виде:

$P(X = k) = \frac{n-1}{n} * \frac{n-2}{n} * \ldots * \frac{k}{n}$

Пусть $\frac{n-1}{n} * \ldots * \frac{n-k-1}{n} = N$

Так как вероятности независимы, то функция максимального правдоподобия будет выглядеть как произведение вероятностей, а именно:

$L = P(n, 1) * \ldots * P(n, k)$

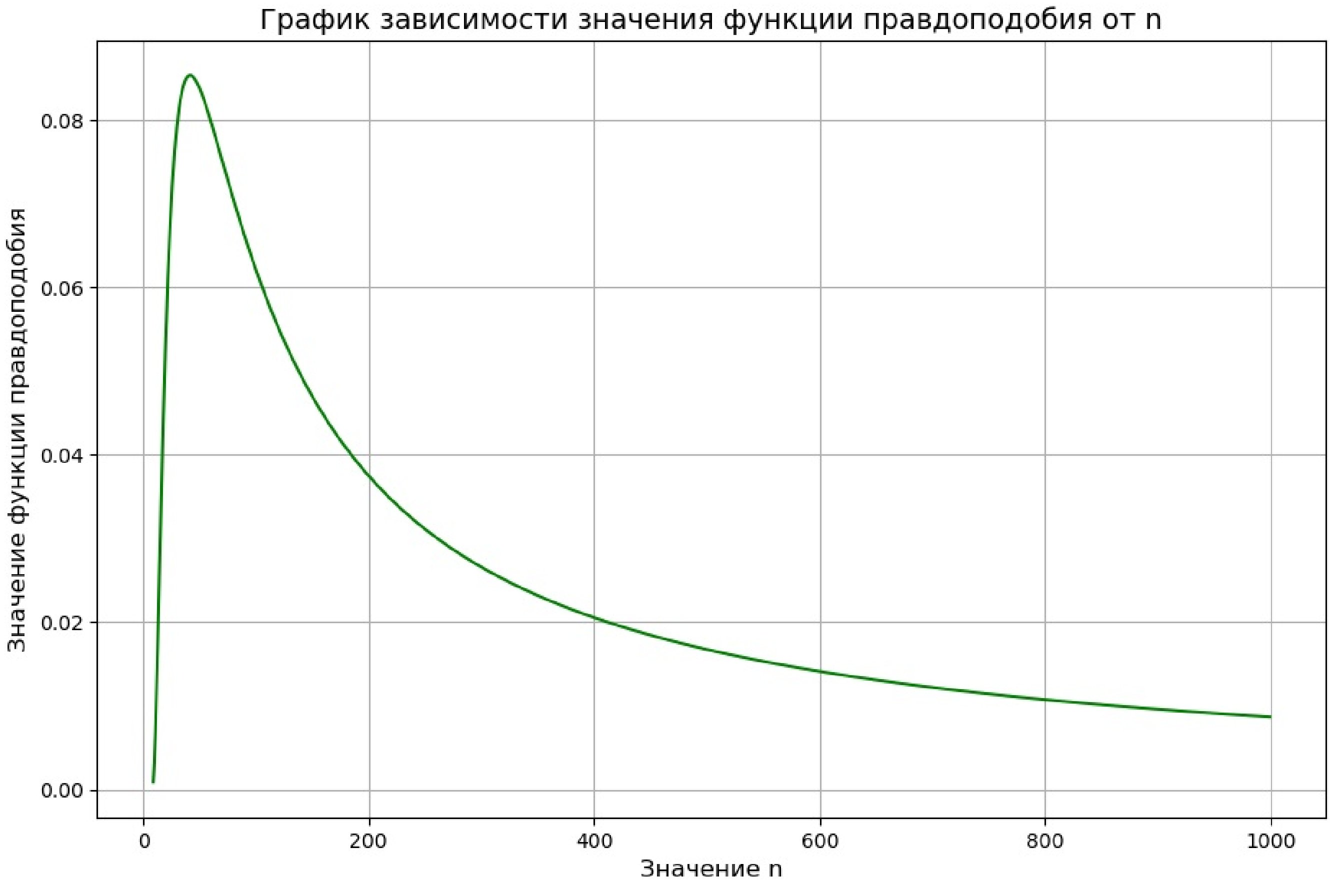
In [2]:
```
def function(j):
    k = 1
    for i in range(2,11):
        k = k*(j-i+2)/j
    return (k*9)/j

prob = [function(j) for j in range(9, 1000)]
itog = np.arange(9, 1000)

a = list(itog[prob == np.max(prob)])
print(f'Оценка числа n методом максимального правдоподобия:',*a)
```

```
Оценка числа n методом максимального правдоподобия: 42
```

Теперь график:

In [3]:
```
plt.figure(figsize=(11,7))
plt.grid(True)
plt.xlabel('Значение n',fontsize=12)
plt.ylabel('Значение функции правдоподобия',fontsize=12)
plt.title('График зависимости значения функции правдоподобия от n',fontsize=14)
plt.plot(np.arange(9, 1000),prob,color = 'green')
```

Out[3]:
```
[<matplotlib.lines.Line2D at 0x15f1ffe8ca0>]
```



пункт б

In [4]:

```
def function_2(n):
    taxers = np.arange(1,n+1)
    real = np.random.choice(taxers, len(taxers)+1)
    seen_elements = set()
    for element in real:
        if element in seen_elements:
            duplicate = element
            break
        seen_elements.add(element)
    return np.where(real == duplicate)[0][0]
```

In [5]:

```
l = []
for n in tqdm.tqdm(range(1,100)):
    l.append(np.mean([function_2(n) for i in range(10**4)]))
```

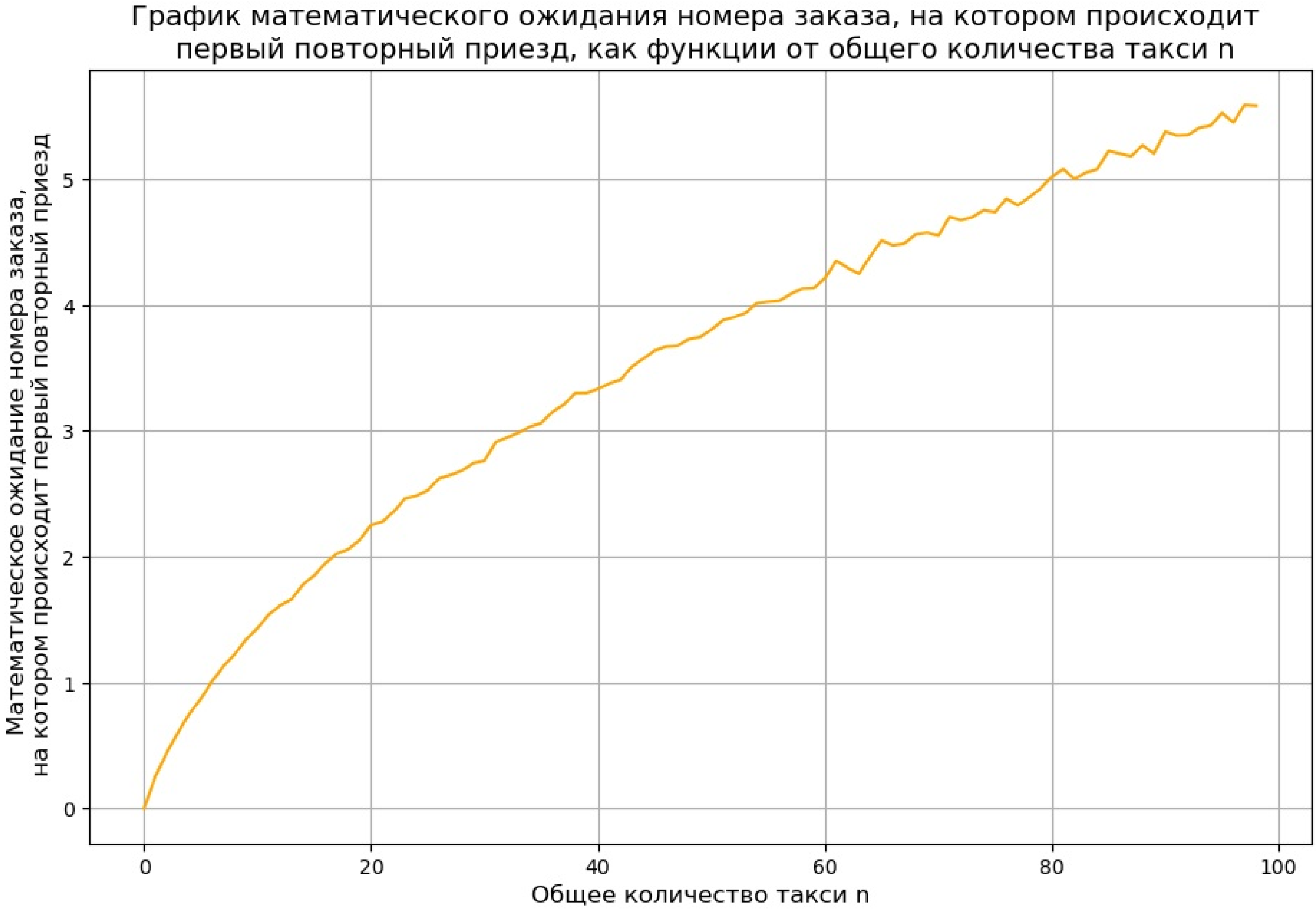
```
100%|██████████████████████████████████████████████████████████████████| 99/99 [00:29<00:00,  3
.41it/s]
```

In [6]:

```
l = []
for n in tqdm.tqdm(range(1,100)):
    l.append(np.mean([function_2(n) for i in range(10**4)]))
plt.figure(figsize=(11,7))
plt.plot(l,color='orange')
plt.grid(True)
plt.xlabel('Общее количество такси n',fontsize=12)
plt.ylabel('Математическое ожидание номера заказа, \n на котором происходит первый повторный приезд',fontsize=1
plt.title('График математического ожидания номера заказа, на котором происходит \n первый повторный приезд, как
```

```
100%|██████████████████████████████████████████████████████████████████| 99/99 [00:29<00:00,  3
.40it/s]
```

Out[6]: Text(0.5, 1.0, 'График математического ожидания номера заказа, на котором происходит \n первый повторный приезд, как функции от общего количества такси n')

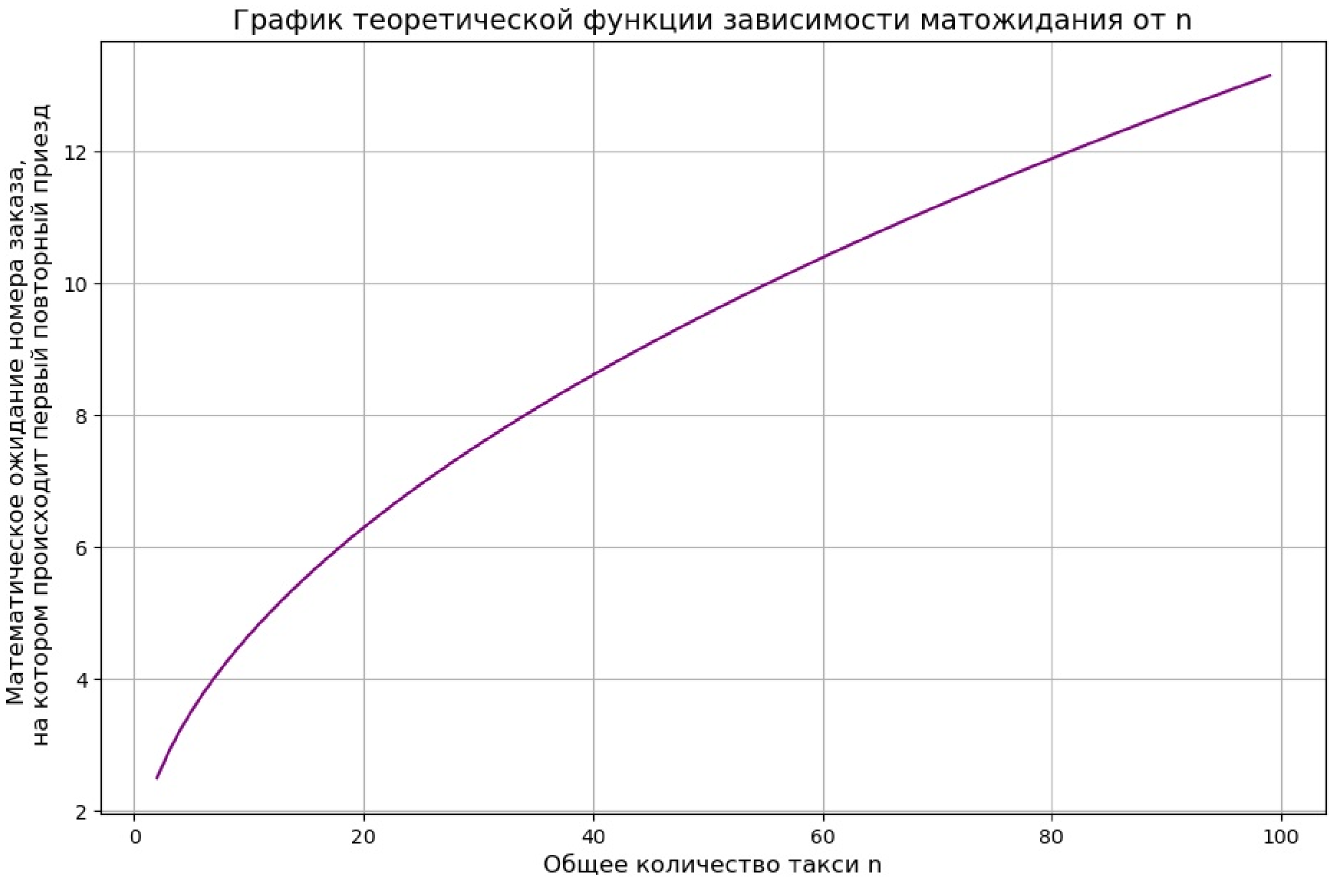


In [7]:
```
def P(k, n):
    a = 1
    for x in range(2,k+1):
        a = a*(n-x+2)/n
    return a*(k-1)/n

def E(n):
    s = 0
    for k in range(2, n+2):
        s += k*P(k, n)
    return(s)
```

Нарисуем график теоретической функции зависимости матожидания от n

In [8]:
```
n_gen = np.arange(2, 100)
E_n = [E(n) for n in n_gen]
plt.figure(figsize=(11,7))
plt.plot(n_gen,E_n, color = 'purple')
plt.grid(True)
plt.xlabel('Общее количество такси n',fontsize=12)
plt.ylabel('Математическое ожидание номера заказа, \n на котором происходит первый повторный приезд',fontsize=1
plt.title('График теоретической функции зависимости матожидания от n',fontsize=14)
```

Out[8]: Text(0.5, 1.0, 'График теоретической функции зависимости матожидания от n')



пункт в

In [9]:
```
np.random.seed(3)
n = 100
taxis = np.arange(1, n+1)
otv = []
for i in range(1, 10001):
    one = np.random.choice(taxis)
    biv = []
    while np.isin(one, biv) == False:
        biv.append(one)
        one = np.random.choice(taxis)

    k = len(biv)+1
    otv.append(k)
```

Функция для максимизации функции правдоподобия. На каждой итерации будет сдвигаться n

In [10]:
```
def L_max(k):
    n = k-1
    L_f1 = P(n, k)
    L_f2 = P(n, k)
    while L_f1 <= L_f2:
        L_f1 = P(n, k)
        n += 1
```

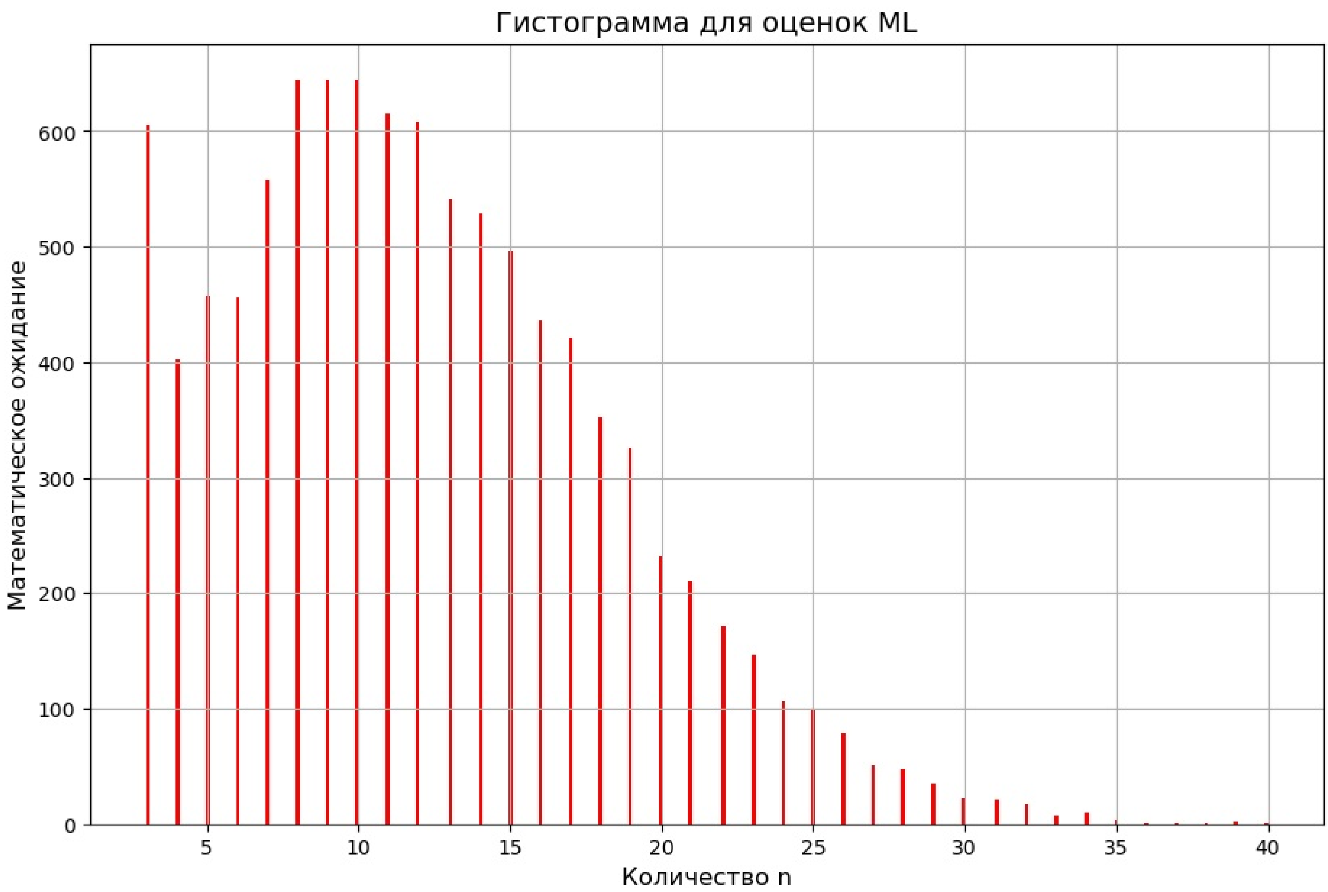
```
        L_f2 = P(n, k)
    return n-1
```

Теперь применим функцию к данным из 10000 симуляций

In [11]:
```
L_v = np.vectorize(L_max)
all_L = L_v(otv)
```

In [12]:
```
plt.figure(figsize = (11,7))
plt.hist(all_L, bins = 300,color='red')
plt.grid(True)
plt.title('Гистограмма для оценок ML',fontsize=14)
plt.xlabel('Количество n',fontsize=12)
plt.ylabel('Математическое ожидание',fontsize=12)
```

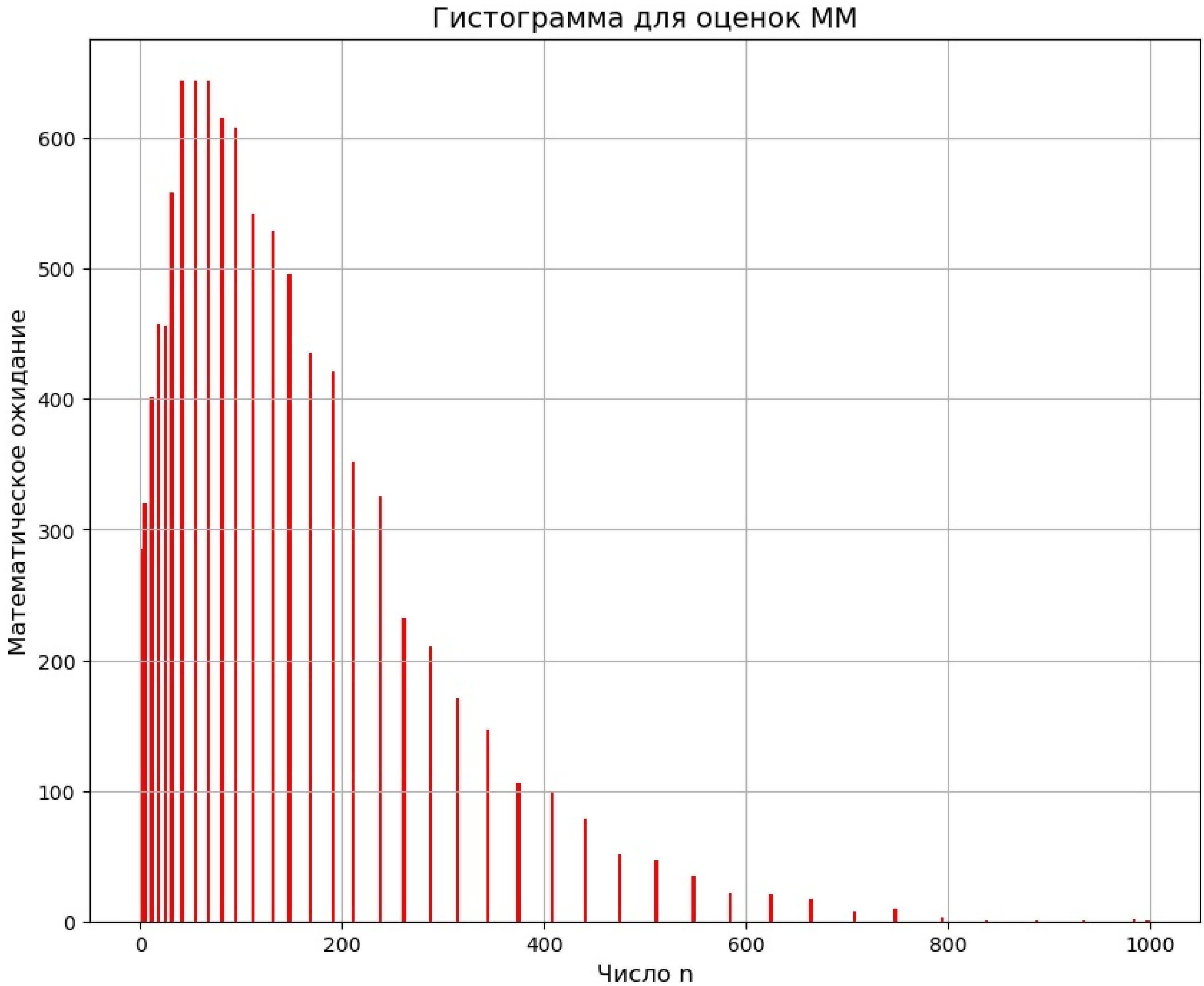
Out[12]:
```
Text(0, 0.5, 'Математическое ожидание')
```



Аналогично, гистограмма для метода моментов

In [13]:
```
otv =np.array(otv)
nn = np.arange(1, 1001)
E_vect = np.vectorize(E)
E_res = E_vect(nn)
E_res2 = E_res[:, np.newaxis]
DN_res2 = otv[np.newaxis, :]
E_result = np.absolute(E_res2 - DN_res2)
all_M = np.argmin(E_result, axis = 0)

plt.figure(figsize = (10,8))
plt.grid(True)
plt.hist(all_M, bins = 300,color='red')
plt.title('Гистограмма для оценок MM',fontsize=14)
plt.xlabel('Число n',fontsize=12)
plt.ylabel('Математическое ожидание',fontsize=12)
```

Out[13]:
```
Text(0, 0.5, 'Математическое ожидание')
```



Посчитаем дисперсию, смещение и среднквадратичную ошибку для каждого из способов

In [14]:
```
s_L = abs(100 - np.mean(all_L))
s_M = abs(100 - np.mean(all_M))
var_L = np.var(all_L)
var_M = np.var(all_M)
sig_L = np.std(all_L)
sig_M = np.std(all_M)
```

Для более удобного восприятия занесём полученные данные в таблицу

In [15]:
```
result = pd.DataFrame({'Оценка': ('ML', 'MM'),'Смещение': (s_L, s_M), 'Дисперсия': (var_L, var_M), 'Среднеквадр
result.set_index('Оценка')
```

Out[15]:

Оценка	Смещение	Дисперсия	Среднеквадратичная ошибка
ML	87.7876	38.275886	6.186751
MM	23.3565	14170.663208	119.040595

Вывод: Как мы можем заметить, метод максимального правдоподобия оказался лучше

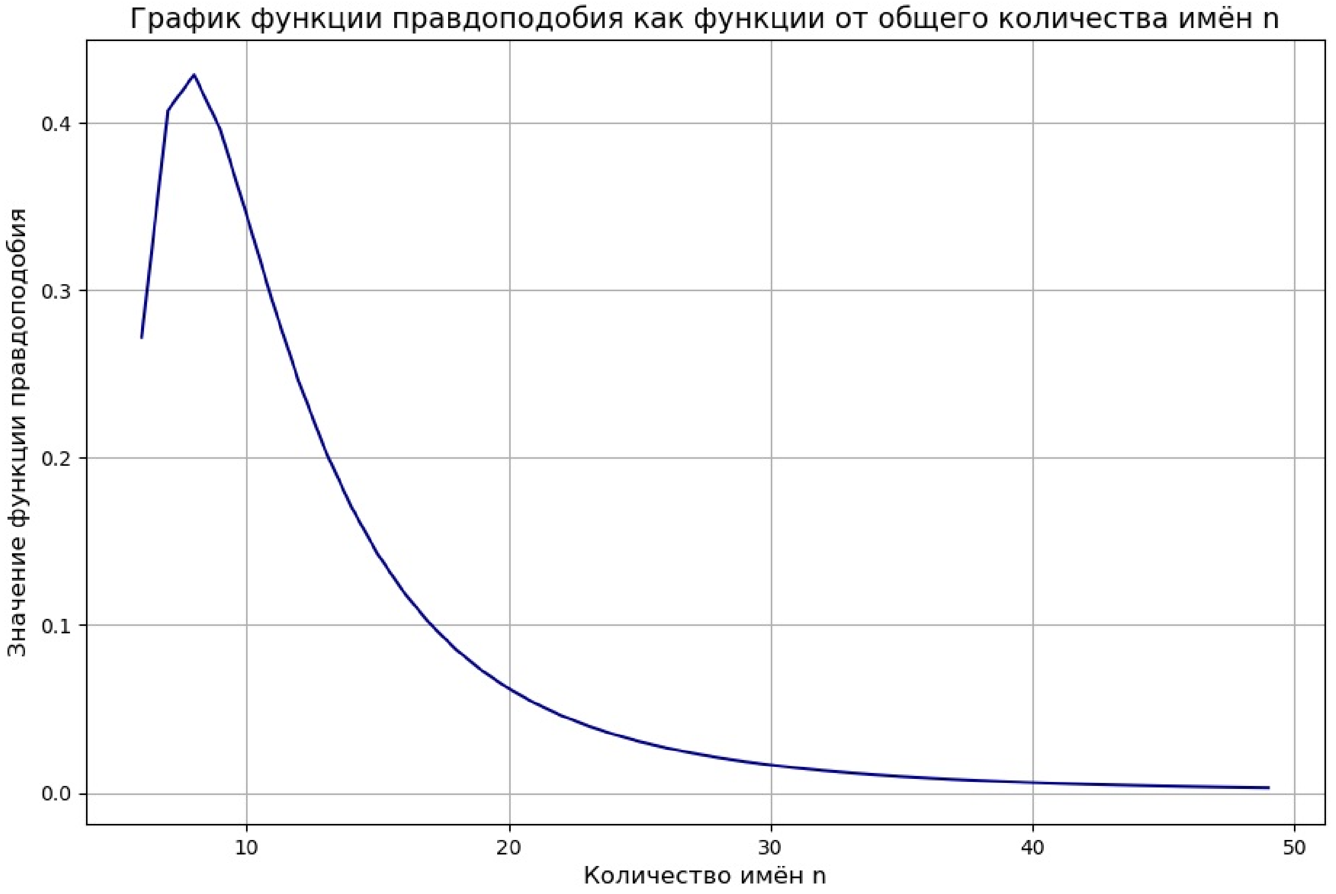
Задача 2

пункт а

In [16]:
```
import itertools
def P_2(k, n, N):
    p=1
    for i in range(1, k):
        p*= ((n-i)/n)
    combinations = itertools.combinations_with_replacement(np.arange(1, k+1), N - k)
    cnt = 0
    for combination in combinations:
        mult = 1
        for i in range(N - k):
            mult *= combination[i]
        cnt += mult
    p *= (cnt/(n**(N - k)))
    return p
```

In [17]:
```
n_vec = np.arange(6, 50)
k = 6
L_list = []
for n in n_vec:
    L_list.append(P_2(k, n, 10))
plt.figure(figsize=(11,7))
plt.plot(n_vec, L_list,color='#00008B')
plt.grid(True)
plt.title('График функции правдоподобия как функции от общего количества имён n',fontsize=14)
plt.ylabel('Значение функции правдоподобия',fontsize=12)
plt.xlabel('Количество имён n', fontsize=12)
```

Out[17]:
```
Text(0.5, 0, 'Количество имён n')
```



In [18]:
```
ans = []
for n in range(6, 15):
    L = P_2(k, n, 10)
    el = [n, L]
    ans.append(el)
ans = sorted(ans, key = lambda x : x[1],reverse=True)
print('Оценка числа n методом максимального правдоподобия:',ans[0][0])
```

```
Оценка числа n методом максимального правдоподобия: 8
```

пункт б

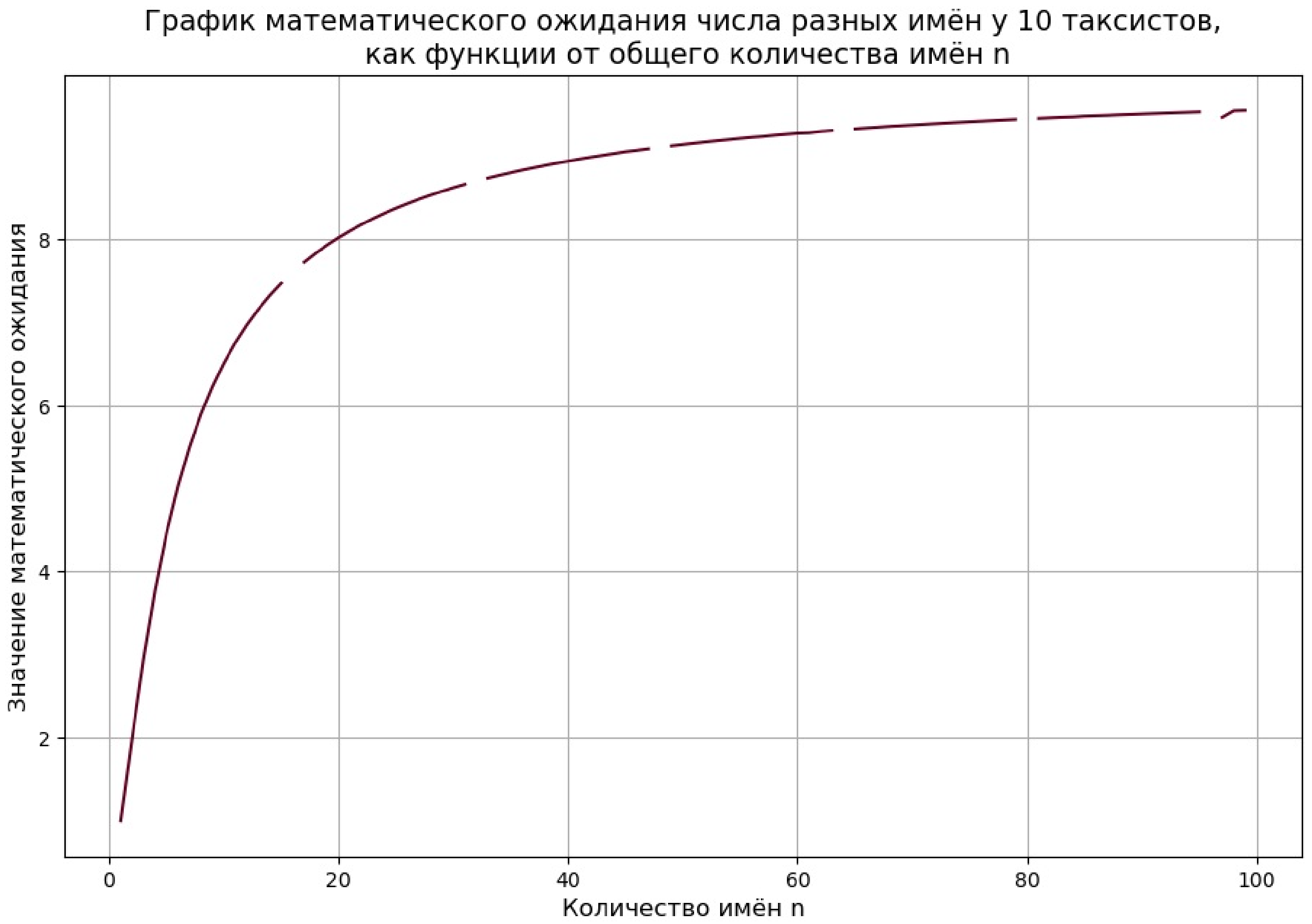
In [19]:
```
def E_2(n, N):
    vec_k = np.arange(1, N+1)
    P_v = np.vectorize(P_2)
    Vec_p = P_v(vec_k, n, N)
```

```
    return vec_k@Vec_p
```

In [20]:
```
n_all = np.arange(1, 100)
E_list = []
otv = []
for n in n_all:
    E_list.append(E_2(n, 10))
    f = [E_2(n, 10), n]
    otv.append(f)
plt.figure(figsize=(11,7))
plt.grid(True)
plt.plot(n_all, E_list,color='#650021')
plt.xlabel('Количество имён n', fontsize=12)
plt.ylabel('Значение математического ожидания',fontsize=12)
plt.title('График математического ожидания числа разных имён у 10 таксистов,\n как функции от общего количества
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_20756\3074648869.py:13: RuntimeWarning: divide by zero encountered i
n long_scalars
  p *= (cnt/(n**(N - k)))
```

Out[20]:
```
Text(0.5, 1.0, 'График математического ожидания числа разных имён у 10 таксистов,\n как функции от общего колич
ества имён n')
```



In [21]:
```
a = list(n_all[abs(np.array(E_list) - 6) == min(abs(np.array(E_list) - 6))])
print('Оценка n методом моментов:',*a)
```

```
Оценка n методом моментов: 8
```

пункт в

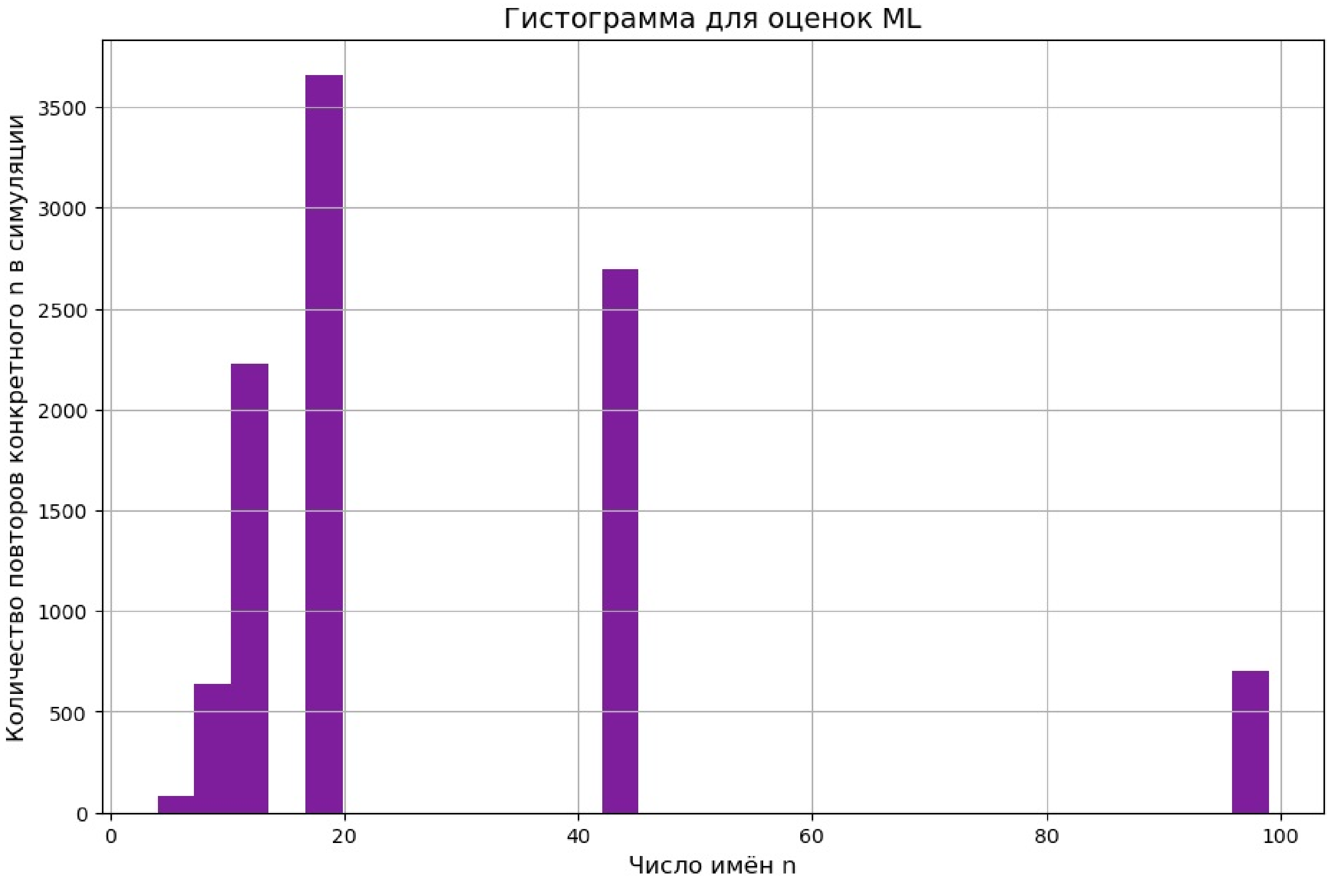
In [22]:
```
# проводим эксперименты
np.random.seed(3)
n = 20
names = np.arange(1, n+1)
otv = []
for i in range(1, 10001):
    one = np.random.choice(names)
    biv = []
    M = 10
    for i in range(M):
        biv.append(one)
        one = np.random.choice(names)
    un = len(set(biv))
    otv.append(un)
```

Теперь напишем функцию правдоподобия, при этом ограничим сверху числом 100, чтобы не получилось так, что значения просто не существует

In [23]:
```
def L_max2(k):
    n = 1
    L_f1 = P_2(k, n, 10)
    L_f2 = P_2(k, n, 10)
    while L_f1 <= L_f2 and n <100:
        L_f1 = P_2(k, n, 10)
        n += 1
        L_f2 = P_2(k, n, 10)
    return n-1
```

Построим гистограмму для ML оценки

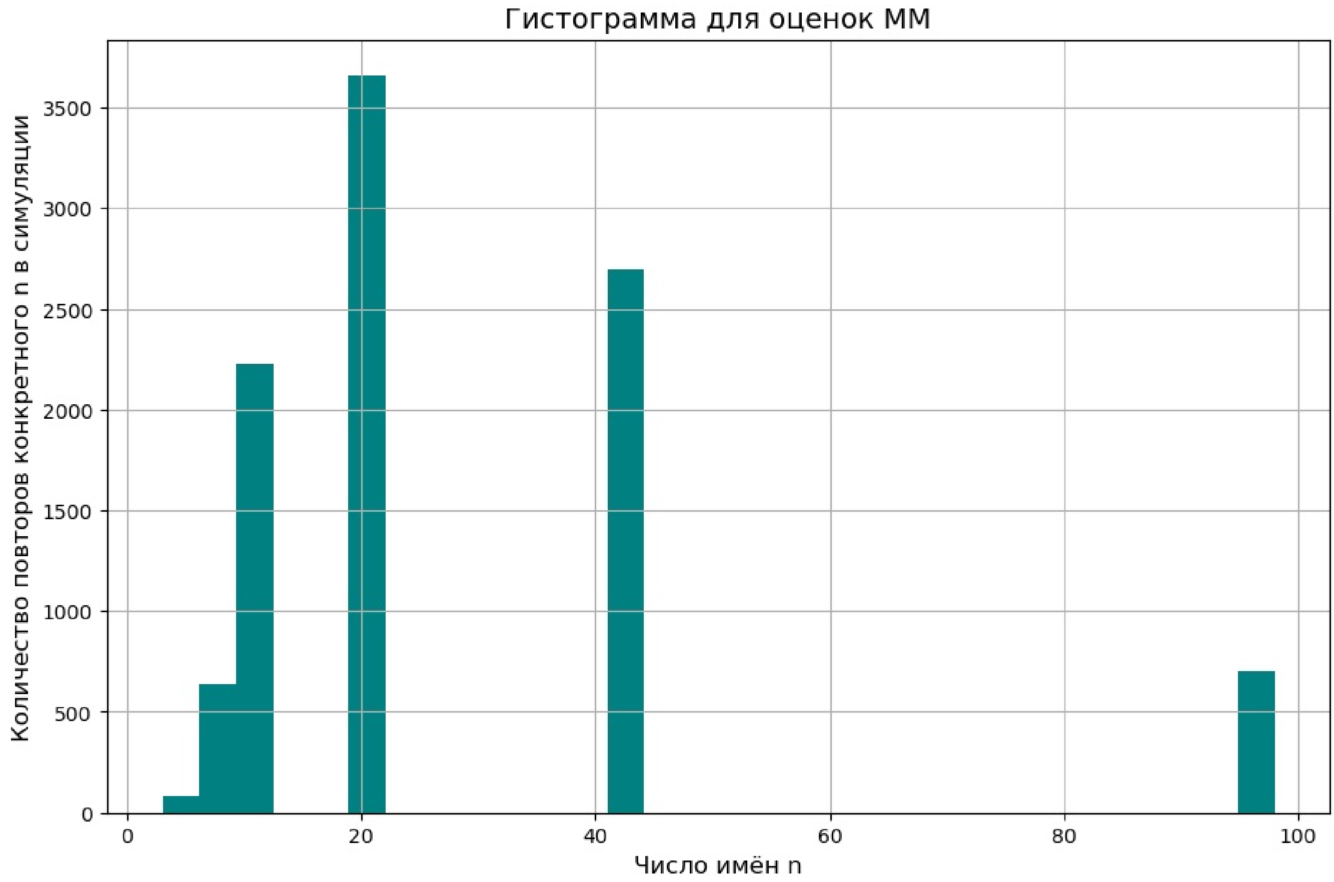
In [24]:
```
L_v = np.vectorize(L_max2)
all_L = L_v(otv)

plt.figure(figsize = (11,7))
plt.grid(True)
plt.hist(all_L, bins = 30,color='#7E1E9C')
plt.title('Гистограмма для оценок ML',fontsize=14)
plt.xlabel('Число имён n',fontsize=12)
plt.ylabel('Количество повторов конкретного n в симуляции',fontsize=12)
plt.show()
```



Аналогично, гистограмма для MM оценки

In [25]:
```
otv =np.array(otv)
nn = np.arange(1, 100)
E_vect = np.vectorize(E_2)
E_res = E_vect(nn, 10)
E_res2 = E_res[:, np.newaxis]
DN_res2 = otv[np.newaxis, :]
E_result = np.absolute(E_res2 - DN_res2)
all_M = np.argmin(E_result, axis = 0)

plt.figure(figsize = (11,7))
plt.hist(all_M, bins = 30,color='#008080')
plt.grid(True)
plt.title('Гистограмма для оценок MM',fontsize=14)
plt.xlabel('Число имён n',fontsize=12)
plt.ylabel('Количество повторов конкретного n в симуляции',fontsize=12)
plt.show()
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_20756\3074648869.py:13: RuntimeWarning: divide by zero encountered i
n long_scalars
  p *= (cnt/(n**(N - k)))
```



Посчитаем дисперсию, смещение и среднквадратичную ошибку для каждого из способов

In [26]:
```
s_L = abs(20 - np.mean(all_L))
s_M = abs(20 - np.mean(all_M))
var_L = np.var(all_L)
var_M = np.var(all_M)
sig_L = np.std(all_L)
sig_M = np.std(all_M)
```

Для более удобного восприятия занесём полученные данные в таблицу

In [27]:
```
result = pd.DataFrame({'Оценка': ('ML', 'MM'),'Смещение': (s_L, s_M), 'Дисперсия': (var_L, var_M), 'Среднеквадр
result.set_index('Оценка')
```

Out[27]:

Оценка	Смещение	Дисперсия	Среднеквадратичная_ошибка
ML	8.4312	522.102067	22.849553
MM	7.8048	515.062297	22.694984

Вывод: Как мы видим, в этом случае ММ оценка оказалась чуть лучше, чем ML

Задача 3

пункт а

In [28]:
```
np.random.seed(123456789)
def simulate_ci_coverage(data, true_mean, num_simulations, confidence_level):
    num_samples = 20
    ci_covered = 0


    for _ in range(num_simulations):
        # Генерация случайной выборки
        sample = np.random.exponential(scale=1, size=num_samples)

        # Классический асимптотический нормальный интервал
        z_critical = sts.norm.ppf(1 - (1 - confidence_level) / 2)
```

```
        std_error = np.std(sample, ddof=1) / np.sqrt(num_samples)
        ci_lower_normal = np.mean(sample) - z_critical * std_error
        ci_upper_normal = np.mean(sample) + z_critical * std_error

        # Наивный бутстрэп интервал
        bootstrap_means = []
        for _ in range(num_samples):
            bootstrap_sample = np.random.choice(sample, size=num_samples, replace=True)
            bootstrap_mean = np.mean(bootstrap_sample)
            bootstrap_means.append(bootstrap_mean)
        bootstrap_means = np.array(bootstrap_means)
        ci_lower_bootstrap = np.percentile(bootstrap_means, (1 - confidence_level) / 2 * 100)
        ci_upper_bootstrap = np.percentile(bootstrap_means, (1 + confidence_level) / 2 * 100)

        # Бутстрэп интервал t-статистики
        t_critical = sts.t.ppf(1 - (1 - confidence_level) / 2, df=num_samples - 1)
        bootstrap_t = (bootstrap_means - np.mean(bootstrap_means)) / np.std(bootstrap_means, ddof=1)
        ci_lower_t = np.mean(sample) - t_critical * np.std(bootstrap_means, ddof=1)
        ci_upper_t = np.mean(sample) + t_critical * np.std(bootstrap_means, ddof=1)

        # Проверка попадания истинного среднего значения в доверительные интервалы
        if ci_lower_normal <= true_mean <= ci_upper_normal:
            ci_covered += 1
        if ci_lower_bootstrap <= true_mean <= ci_upper_bootstrap:
            ci_covered += 1
        if ci_lower_t <= true_mean <= ci_upper_t:
            ci_covered += 1

    coverage_probability = ci_covered / (num_simulations * 3)
    return coverage_probability

# Параметры симуляции
num_simulations = 10000
confidence_level = 0.95
true_mean = 1

# Симуляция для каждого способа
ci_coverage_normal = simulate_ci_coverage(data=None, true_mean=true_mean, num_simulations=num_simulations, conf
ci_coverage_bootstrap = simulate_ci_coverage(data=None, true_mean=true_mean, num_simulations=num_simulations, c
ci_coverage_t = simulate_ci_coverage(data=None, true_mean=true_mean, num_simulations=num_simulations, confidenc

# Вывод результатов
print("Вероятность покрытия (классический асимптотический нормальный интервал): {:.4f}".format(ci_coverage_norm
print("Вероятность покрытия (наивный бутстрэп интервал): {:.4f}".format(ci_coverage_bootstrap))
print("Вероятность покрытия (бутстрэп интервал t-статистики): {:.4f}".format(ci_coverage_t))
```

```
Вероятность покрытия (классический асимптотический нормальный интервал): 0.8741
Вероятность покрытия (наивный бутстрэп интервал): 0.8737
Вероятность покрытия (бутстрэп интервал t-статистики): 0.8768
```

пункт б

In [29]:

```
np.random.seed(123456789)

def simulate_ci_coverage(data, true_mean, num_simulations, confidence_level):
    num_samples = 20
    ci_covered = 0

    for _ in range(num_simulations):
        # Генерация случайной выборки
        sample = np.random.standard_t(df=3, size=num_samples)

        # Классический асимптотический нормальный интервал
        z_critical = sts.t.ppf(1 - (1 - confidence_level) / 2, df=num_samples-1)
        std_error = np.std(sample, ddof=1) / np.sqrt(num_samples)
        ci_lower_normal = np.mean(sample) - z_critical * std_error
        ci_upper_normal = np.mean(sample) + z_critical * std_error

        # Наивный бутстрэп интервал
        bootstrap_means = []
        for _ in range(num_samples):
            bootstrap_sample = np.random.choice(sample, size=num_samples, replace=True)
            bootstrap_mean = np.mean(bootstrap_sample)
            bootstrap_means.append(bootstrap_mean)
        bootstrap_means = np.array(bootstrap_means)
        ci_lower_bootstrap = np.percentile(bootstrap_means, (1 - confidence_level) / 2 * 100)
        ci_upper_bootstrap = np.percentile(bootstrap_means, (1 + confidence_level) / 2 * 100)

        # Бутстрэп интервал t-статистики
        t_critical = sts.t.ppf(1 - (1 - confidence_level) / 2, df=num_samples - 1)
        bootstrap_t = (bootstrap_means - np.mean(bootstrap_means)) / np.std(bootstrap_means, ddof=1)
        ci_lower_t = np.mean(sample) - t_critical * np.std(bootstrap_means, ddof=1)
        ci_upper_t = np.mean(sample) + t_critical * np.std(bootstrap_means, ddof=1)

        # Проверка попадания истинного среднего значения в доверительные интервалы
        if ci_lower_normal <= true_mean <= ci_upper_normal:
            ci_covered += 1
```

```
        if ci_lower_bootstrap <= true_mean <= ci_upper_bootstrap:
            ci_covered += 1
        if ci_lower_t <= true_mean <= ci_upper_t:
            ci_covered += 1

    coverage_probability = ci_covered / (num_simulations * 3)
    return coverage_probability

# Параметры симуляции
num_simulations = 10000
confidence_level = 0.95
true_mean = 0

# Симуляция для каждого способа
ci_coverage_normal = simulate_ci_coverage(data=None, true_mean=true_mean, num_simulations=num_simulations, conf
ci_coverage_bootstrap = simulate_ci_coverage(data=None, true_mean=true_mean, num_simulations=num_simulations, c
ci_coverage_t = simulate_ci_coverage(data=None, true_mean=true_mean, num_simulations=num_simulations, confidenc

# Вывод результатов
print("Вероятность покрытия (классический асимптотический нормальный интервал): {:.4f}".format(ci_coverage_norm
print("Вероятность покрытия (наивный бутстрэп интервал): {:.4f}".format(ci_coverage_bootstrap))
print("Вероятность покрытия (бутстрэп интервал t-статистики): {:.4f}".format(ci_coverage_t))
```

```
Вероятность покрытия (классический асимптотический нормальный интервал): 0.9129
Вероятность покрытия (наивный бутстрэп интервал): 0.9107
Вероятность покрытия (бутстрэп интервал t-статистики): 0.9123
```

пункт в

Вывод: Как мы можем увидеть, если случайные величины имеют экспоненциальное распределение, то лучше оказывается способ бутстрэпа интервала t-статистики, в случае с распределением Стьюдента - классический асимптотический нормальный интервал. Но а в целом, результаты получаются лучше для случайных величин, имеющих t-распределение с 3-мя степенями свободы.

Задача 4

пункт а

In [30]:
```
df_s = pd.read_excel("C:\\Users\\user\\OneDrive\\Рабочий стол\\тервер матстат\\согласные.xlsx")
df_s.head()
```

Out[30]:

	last name	score
0	Бабурина	10
1	Багаутдинов	16
2	Бартенева	27
3	Батмунх	16
4	Бгажноков	22

In [31]:
```
df_g = pd.read_excel("C:\\Users\\user\\OneDrive\\Рабочий стол\\тервер матстат\\гласные.xlsx")
df_g.head()
```

Out[31]:

	last name	score
0	Абдугаффорова	17
1	Абдулаева	21
2	Аврамидис	0
3	Аврамчук	29
4	Авсеенко	26

In [32]:
```
mu_s = np.array(df_s['score'])
mu_g = np.array(df_g['score'])
```

In [33]:
```
p_value = sts.ttest_ind(mu_s,mu_g,equal_var=False, alternative='two-sided')[1]
p_value
```

Out[33]:
```
0.3974027153843839
```

In [34]:
```
if p_value>0.05:
    print('H0 не отвергается')
else:
    print('H0 отвергается')
```

```
H0 не отвергается
```

пункт б

In [35]:
```
np.random.seed(19)
num_samples = 10**4
mean = np.mean(mu_s)-np.mean(mu_g)
bootstrap_group1 = np.random.choice(np.arange(len(mu_s)), size=(num_samples, len(mu_s)))
bootstrap_group2 = np.random.choice(np.arange(len(mu_g)), size=(num_samples, len(mu_g)))
bootstrap_diff = np.mean(mu_s[bootstrap_group1], axis=1) - np.mean(mu_g[bootstrap_group2], axis=1)
mask = (bootstrap_diff >= np.mean(mu_s) - np.mean(mu_g))
right = np.mean(mask)
left = np.mean(~mask)
p_value = 2*np.min([right,left])

print("Bootstrap p-value:",p_value)
```

```
Bootstrap p-value: 0.9816
```

In [36]:
```
if p_value>0.05:
    print('H0 не отвергается')
else:
    print('H0 отвергается')
```

```
H0 не отвергается
```

пункт в

In [37]:
```
np.random.seed(23)
num_samples = 10**4
mean = np.mean(mu_s)-np.mean(mu_g)
bootstrap_group1 = np.random.choice(np.arange(len(mu_s)), size=(num_samples, len(mu_s)))
bootstrap_group2 = np.random.choice(np.arange(len(mu_g)), size=(num_samples, len(mu_g)))
bootstrap_diff = np.mean(mu_s[bootstrap_group1], axis=1) - np.mean(mu_g[bootstrap_group2], axis=1)
bootstrap_diff_se = np.sqrt(((np.std(mu_s[bootstrap_group1], axis=1, ddof=1))**2 /(len(mu_s))
                + np.std(mu_g[bootstrap_group2], axis=1, ddof=1))**2 /(len(mu_g)))
t_stat = (bootstrap_diff-mean)/bootstrap_diff_se
mask = (sts.ttest_ind(mu_s,mu_g,equal_var=False, alternative='two-sided')[0]< t_stat)
right = np.mean(mask)
left = np.mean(~mask)
p_value = 2*np.min([right,left])

print("Bootstrap p-value:",p_value)
```

```
Bootstrap p-value: 0.4086
```

In [38]:
```
if p_value>0.05:
    print('H0 не отвергается')
else:
    print('H0 отвергается')
```

```
H0 не отвергается
```

пункт г

In [39]:
```
# !pip install mlxtend
```

In [40]:
```
from mlxtend.evaluate import permutation_test
```

In [41]:
```
p_value = permutation_test(mu_s, mu_g,
                           method='approximate',
                           num_rounds=10000,
                           seed=0)
print(p_value)
```

```
0.37446255374462556
```

In [42]:
```
if p_value>0.05:
    print('H0 не отвергается')
else:
    print('H0 отвергается')
```

```
H0 не отвергается
```

Задача 5

In [43]:
```
df = pd.read_excel("C:\\Users\\user\\OneDrive\\Рабочий стол\\тервер матстат\\экзамен.xlsx")
df.head()
```

Out[43]:

	last name	score
0	Бабурина	10
1	Багаутдинов	16
2	Бартенева	27
3	Батмунх	16
4	Бгажноков	22

In [44]:
```
median = df['score'].median()
median
```

Out[44]:
```
17.5
```

In [45]:
```
pivot = pd.read_excel("C:\\Users\\user\\OneDrive\\Рабочий стол\\тервер матстат\\табл сопр.xlsx")
pivot = pivot.drop('Unnamed: 0',axis=1)
pivot.head()
```

Out[45]:

	набрали больше медианы	набрали меньше медианы
0	145	138
1	21	28

пункт а

In [46]:
```
orr = (pivot['набрали больше медианы'][0]*pivot['набрали меньше медианы'][1])/(pivot['набрали меньше медианы'][
orr #значение отношения шансов
```

Out[46]:
```
1.4009661835748792
```

Асимтотический доверительный интервал будем искать в виде:

$$[e^{ln(OR)} - z_{crit} * \sqrt{\frac{1}{A}+\frac{1}{B}+\frac{1}{C}+\frac{1}{D}}; e^{ln(OR)} + z_{crit} * \sqrt{\frac{1}{A}+\frac{1}{B}+\frac{1}{C}+\frac{1}{D}}]$$

Тогда:

In [47]:
```
z_crit =sts.norm.ppf(0.95)
```

In [48]:
```
se = np.sqrt(1/145+1/138+1/21+1/28)
```

In [49]:
```
lower_edge = math.exp(np.log(orr)-z_crit*se)
upper_edge = math.exp(np.log(orr)+z_crit*se)
```

In [50]:
```
print(f'CI:[{np.round(lower_edge,3)};{np.round(upper_edge,3)}]')
```

```
CI:[0.838;2.341]
```

Из-за того, что значение "1" на уровне значимости 5% попадает в построенный доверительный интервал, нет оснований отвергать H0: OR = 1

In [51]:
```
z_obs = (orr-1)/se
```

In [52]:
```
area1 = sts.norm.cdf(z_obs, loc = 0, scale = 1)
area2 = 1 -sts.norm.cdf(z_obs, loc = 0, scale = 1)
```

In [53]:
```
p_value = 2* min(area1,area2)
print(f'P-value:{np.round(p_value,3)}')
```

```
P-value:0.199
```

пункт б

In [54]:
```
p_sogl = 145/(145+138)
p_glas = 21/(21+28)
```

In [55]:
```
odd_rat = np.log(p_glas)-np.log(p_sogl)
```

In [56]:
```
se_2 = np.sqrt(1/145-1/(145+135)+1/21-1/(21+28))
```

In [57]:
```
lower_edge2 = odd_rat-1.96*se_2
upper_edge2 = odd_rat+1.96*se_2
```

```
print(f'CI:[{np.round(math.exp(lower_edge2),3)};{np.round(math.exp(upper_edge2),3)}]')
```

```
CI:[0.594;1.178]
```

Из-за того, что значение "1" на уровне значимости 5% попадает в построенный доверительный интервал, нет оснований отвергать H0

In [58]:
```
z_obs2 = (odd_rat-np.log(1))/se_2
```

In [59]:
```
area1 = sts.norm.cdf(z_obs2, loc = 0, scale = 1)
area2 = 1 -sts.norm.cdf(z_obs2, loc = 0, scale = 1)
```

In [60]:
```
p_value = 2* min(area1,area2)
print(f'P-value:{np.round(p_value,3)}')
```

```
P-value:0.307
```

пункт в

In [61]:
```
np.random.seed(23)
# Функция для вычисления отношения шансов
def odds_ratio(data):
    a = data[0][0]  # Количество успехов в первой группе
    b = data[0][1]  # Количество неудач в первой группе
    c = data[1][0]  # Количество успехов во второй группе
    d = data[1][1]  # Количество неудач во второй группе

    # Вычисление отношения шансов
    odds_ratio = (a / b) / (c / d)
    return odds_ratio

# Данные из таблицы [[145, 138], [21, 28]]
data = np.array([[145, 138], [21, 28]])

# Вычисление истинного отношения шансов
true_odds_ratio = odds_ratio(data)

# Параметры для бутстрэпа
n_iterations = 10000
bootstrap_ratio = np.zeros(n_iterations)

# Наивный бутстрэп
for i in range(n_iterations):
    resampled_data = np.zeros_like(data)
    for j in range(2):
        total = np.sum(data[j])
        resampled_data[j] = np.random.binomial(total, data[j] / total)
    bootstrap_ratio[i] = odds_ratio(resampled_data)

# Построение доверительного интервала
lower_bound = np.percentile(bootstrap_ratio, 2.5)
upper_bound = np.percentile(bootstrap_ratio, 97.5)

# Проверка гипотезы о равенстве отношения шансов 1
p_value = 2*min(np.mean(bootstrap_ratio >= true_odds_ratio),np.mean(bootstrap_ratio <= true_odds_ratio))

# Вывод результатов
print("95% Доверительный интервал для отношения шансов:", [np.round(lower_bound,3), np.round(upper_bound,3)])
print("P-значение для гипотезы о равенстве отношения шансов 1:", p_value)
```

```
95% Доверительный интервал для отношения шансов: [0.912, 2.261]
P-значение для гипотезы о равенстве отношения шансов 1: 0.9826
```

Задача 6

пункт а

In [62]:
```
dlina = []
for i in df['last name']:
    dlina.append(len(i))
```

In [63]:
```
y = df['score'].mean()
f = np.mean(dlina)
```

In [64]:
```
y == f
```

Out[64]:
```
False
```

In [65]:
```
b =np.round(y/f,2)
b
```

Out[65]: 2.06

Вывод: значит, коэффициент β равен 2.06

пункт б

In [66]:
```
#выборочная корреляция
np.corrcoef(df['score'],dlina)[0][1]
```

Out[66]: 0.025328052669147668

Я буду переставлять набранное количество баллов - Y_i

In [67]:
```
[r,pv] =sts.pearsonr(df['score'],dlina)
pS = copy.copy(df['score'])
pR = []
p=10000
for i in range(0,p):
    random.shuffle(pS)
    pR.append(sts.pearsonr(pS,dlina)[0])
p_val = len(np.where(np.abs(pR)>=np.abs(r))[0])/p
```

In [68]:
```
p_val
```

Out[68]: 0.6423

Таким образом, на уровне значимости 5% нет оснований отвергать основную гипотезу о нулевой корреляции

Задача 7

DA

Известно, что $E(X) = -1$, $E(Y) = 2$, $Var(X) = 4$, $Var(Y) = 9$, $Cov(X, Y) = -3$. Чему равна ковариация
$Cov(X + 2Y, 2X + 3)$?

Для нахождения ковариации $Cov(X + 2Y, 2X + 3)$ используем свойства ковариации:

$Cov(aX + bY, cW + d) = ac * Cov(X, W) + ad * Cov(X, 1) + bc * Cov(Y, W) + bd * Cov(Y, 1)$.

В данном случае у нас есть:

$E(X) = -1$,
$E(Y) = 2$,
$Var(X) = 4$,
$Var(Y) = 9$,
$Cov(X, Y) = -3$.

Теперь можем вычислить ковариацию $Cov(X + 2Y, 2X + 3)$:

$Cov(X + 2Y, 2X + 3) = Cov(X, 2X) + Cov(X, 3) + Cov(2Y, 2X) + Cov(2Y, 3)$.

Подставим значения:

$Cov(X + 2Y, 2X + 3) = Cov(X, 2X) + Cov(X, 3) + Cov(Y, 2X) + Cov(Y, 3)$.

Распишем каждое слагаемое:

$Cov(X, 2X) = 2 * Var(X) = 2 * 4 = 8$,

$Cov(X, 3) = 0$, так как X и константа 3 независимы,

$Cov(Y, 2X) = 2 * Cov(X, Y) = 2 * (-3) = -6$,

$Cov(Y, 3) = 0$, так как Y и константа 3 независимы.

Теперь можем сложить все слагаемые:

$Cov(X + 2Y, 2X + 3) = 8 + 0 + (-6) + 0 = 2$.

Таким образом, ковариация $Cov(X + 2Y, 2X + 3)$ равна 2.

DA

Ты допустил ошибку: необходимо брать ковариацию от 2X,2Y, а не от Y,2X. Исправь ошибку в решении с учётом этого факта

Приношу извинения за путаницу. Действительно, для вычисления ковариации Cov(X + 2Y, 2X + 3) нужно учесть правильные комбинации переменных.

Исправим решение с учетом этого факта:

Дано:
E(X) = -1,
E(Y) = 2,
Var(X) = 4,
Var(Y) = 9,
Cov(X, Y) = -3.

Вычислим ковариацию Cov(X + 2Y, 2X + 3):

Cov(X + 2Y, 2X + 3) = Cov(X, 2X) + Cov(X, 3) + Cov(2Y, 2X) + Cov(2Y, 3).

Распишем каждое слагаемое:

Cov(X, 2X) = 2 * Var(X) = 2 * 4 = 8,

Cov(X, 3) = 0, так как X и константа 3 независимы,

Cov(2Y, 2X) = 2 * 2 * Cov(Y, X) = 4 * (-3) = -12,

Cov(2Y, 3) = 2 * Cov(Y, 1) = 2 * 0 = 0.

Теперь можем сложить все слагаемые:

Cov(X + 2Y, 2X + 3) = 8 + 0 + (-12) + 0 = -4.

Таким образом, ковариация Cov(X + 2Y, 2X + 3) равна -4.

Задача 8

Полезный ресурс:

[Математическая статистика 2022/2023 (пилотный поток) — Wiki - Факультет компьютерных наук (hse.ru)]

Вики-страничка математической статистики ПМИ. Есть куча интересных и сложных задачек, хорошо вписывающихся в наш курс. Дополнительный ресурс для тренировки и поиска элегантных решений.