```python
import numpy as np
import random
import scipy.stats as sts
from tqdm.notebook import tqdm
import matplotlib.pyplot as plt
```

№ 1

Однажды в Самарканде турист заказывал Яндекс-такси. На десятом заказе впервые приехал таксист, который уже раньше приезжал к туристу. Для упрощения предположим, что все n таксистов Самарканда всегда на работе и приезжают равновероятно.

а)

_[5] Постройте график функции правдоподобия как функции от общего количества такси

n. Найдите оценку числа n методом максимального правдоподобия._

Вероятность того, что приехал новый таксист на i-ый день (при условии, что до этого все приезжавшие таксисты были различны): $\frac{n-i+1}{n}$

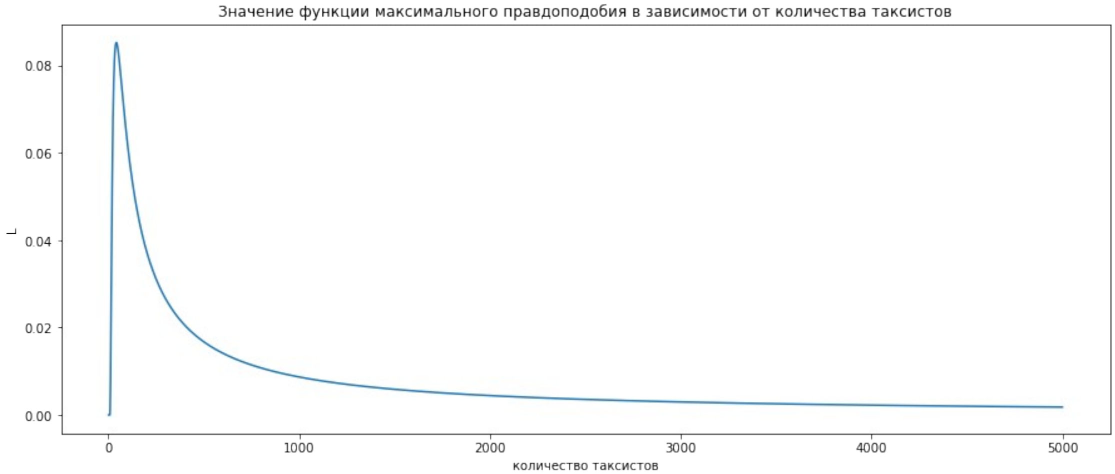
```python
maximum_likelihood_estimation(12, 10)
```

```
0.01160421489197531
```

```python
def maximum_likelihood_estimation(n, days):
    L = 1
    for i in range(1, days - 1):
        L *= (n-i)/n
    L *= (days - 1) / n
    return L
```

```python
maximum_likelihood_estimation_vec = np.vectorize(maximum_likelihood_estimation)
n = np.arange(1, 5000)
days = 10
L = maximum_likelihood_estimation_vec(n, days)

plt.figure(figsize = (15, 6))
plt.plot(n, L)
plt.xlabel("количество таксистов")
plt.ylabel("L")
plt.title("Значение функции максимального правдоподобия в зависимости от количества таксистов")
plt.show()
```



Если бы я решал на листочке, то чтобы найти максимум сделал бы так:

$$\begin{cases} \frac{L(n+1)}{L(n)} < 1 \\ \frac{L(n)}{L(n-1)} > 1 \end{cases}$$

Но тут просто посмотрю на то, где L приняло максимально значение

```
n_ml = np.argmax(L) + 1
print(f'n_ml = {n_ml}')
```

```
n_ml = 42
```

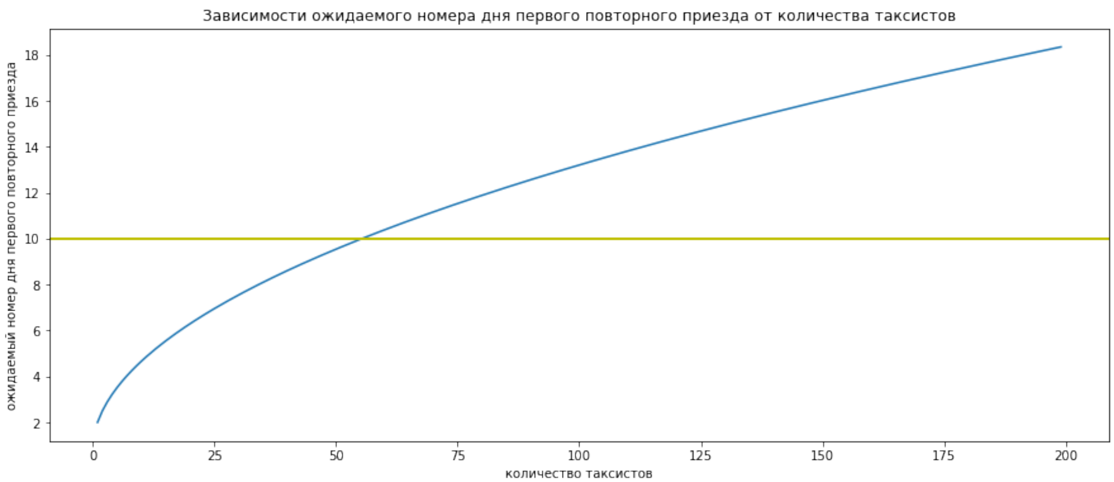
б)

[5] Постройте график математического ожидания номера заказа, на котором происходит первый повторный приезда, как функции от общего количества такси n. Найдите оценку числа n методом моментов.

Буду оценивать с помощью первого начального момента

```
def expected_value(n):
    """
    здесь n - количество таксистов, поэтому дней не может быть больше
n
    """
    E = 0
    for day in range(1, n+2):
        P = maximum_likelihood_estimation(n, day)
        E += P * day
    return E
```

```
n = np.arange(1, 200)
expected_value_vec = np.vectorize(expected_value)
E = expected_value_vec(n)
plt.figure(figsize = (15, 6))
plt.plot(n, E)
plt.axhline(10, c='y', linewidth=2, label = 'day__number_obs')
plt.xlabel('количество таксистов')
plt.ylabel('ожидаемый номер дня первого повторного приезда')
plt.title('Зависимости ожидаемого номера дня первого повторного приезда от количества таксистов')
plt.show()
```



Если я получил методов моментов, что $E(days)=f(n)$, значит чтобы получить оценку n методом моментов, делаем так: $n_{mm}=f^{-1}(10)$

```
n_mm = np.argmin(np.abs(E-10)) + 1
print(f'n_mm = {round(n_mm, 2)}')
```

```
n_mm = 55
```

в)

[15] Предположим, что настоящее n равно 100. Проведя 10000 симуляций вызовов такси до первого повторного, 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.

```
n_sim = 10**4
n = 100
n_obs = []
np.random.seed(42)

def gen_taxist(n):
```

```
    return sts.randint(0, n).rvs(1)[0]

for n_i in tqdm(range(n_sim)):
    arrived_taxists = set()
    taxist = gen_taxist(n)
    n_obs_i = 0
    while taxist not in arrived_taxists:
        n_obs_i += 1
        arrived_taxists.add(taxist)
        taxist = gen_taxist(n)
    n_obs.append(n_obs_i)
```

{"model_id":"60484a7d8452467ab5d41168947a0011","version_major":2,"version_minor":0}

Рассмотрим ML оценки:

```
n_obs = np.array(n_obs)
n_ml_list = []
for i in tqdm(n_obs):

n_ml_list.append(np.argmax(maximum_likelihood_estimation_vec(np.arange
(1, 1000), i)) + 1)
n_ml_list = np.array(n_ml_list)
```

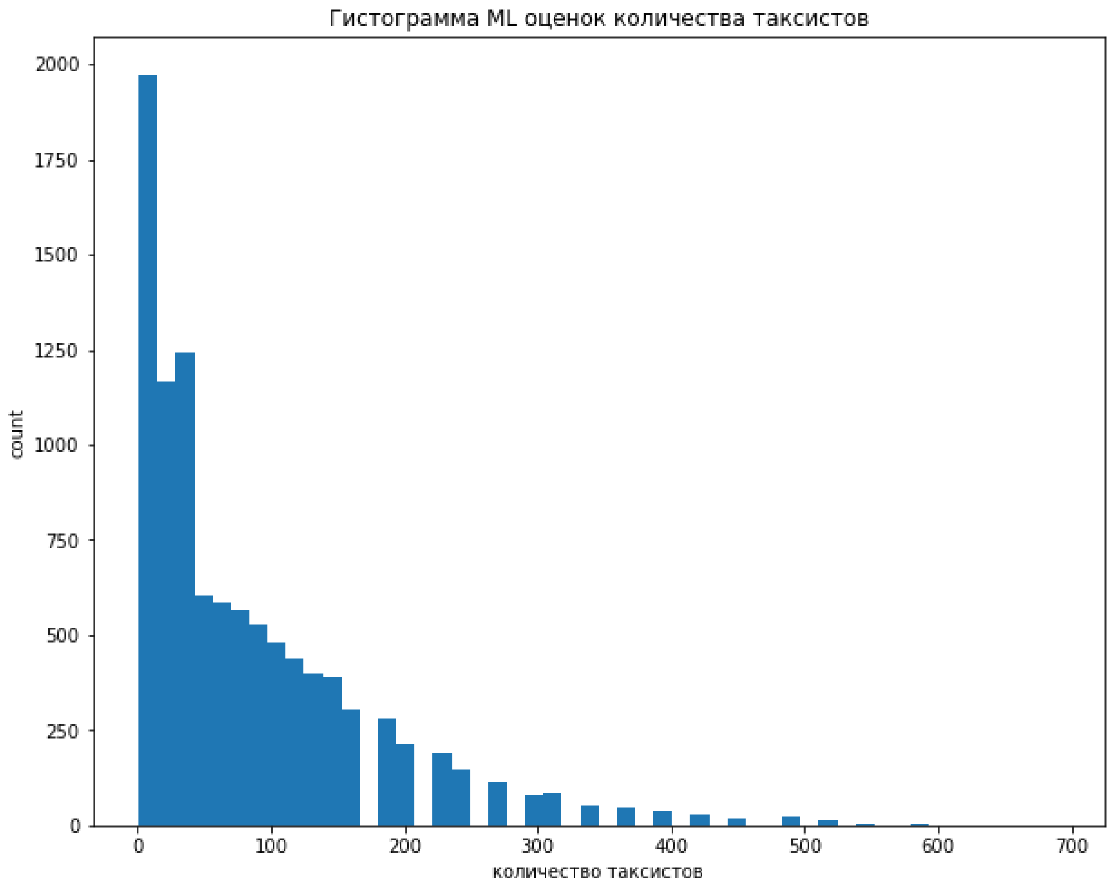
{"model_id":"5778518c30c147f88faeab850f30be40","version_major":2,"version_minor":0}

```
from sklearn.metrics import mean_squared_error

print(f'Среднее: {np.mean(n_ml_list)}')
print(f'Смещение: {np.mean(np.abs(n_ml_list - n))}')
print(f'Дисперсия: {np.var(n_ml_list)}')
print(f'Среднеквадратичная ошибка: {mean_squared_error([100] * 10000,
n_ml_list)}')
```

```
Среднее: 84.0453
Смещение: 69.4043
Дисперсия: 7442.721847909999
Среднеквадратичная ошибка: 7697.2743
```

```
plt.figure(figsize = (10, 8))
plt.hist(n_ml_list, bins = 50)
plt.xlabel('количество таксистов')
plt.ylabel('count')
plt.title('Гистограмма ML оценок количества таксистов')
plt.show()
```



Теперь рассмотрим MM оценки:

```
E = expected_value_vec(np.arange(1, 1000))
n_mm_list = []
for i in tqdm(n_obs):
    n_mm_list.append(np.argmin(np.abs(E-i)) + 1)
n_mm_list = np.array(n_mm_list)
```

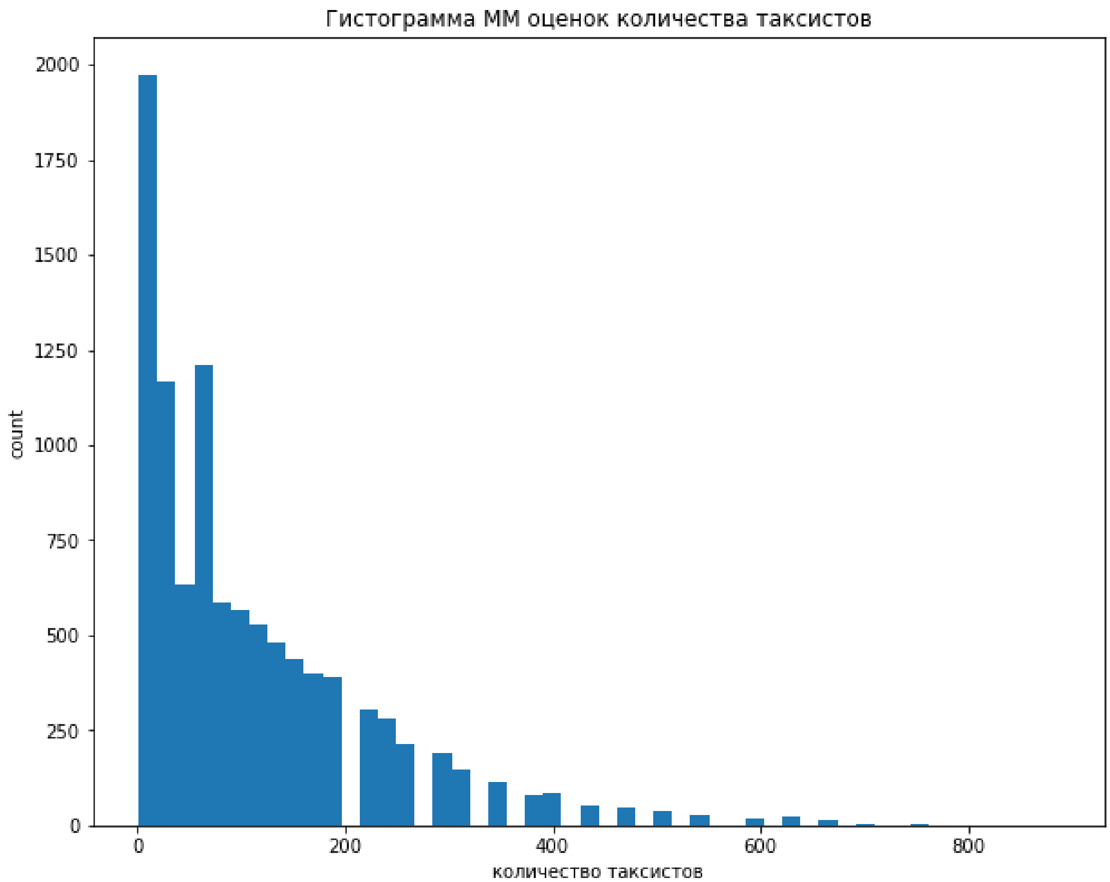
{"model_id":"bdc2bfa436894a23a814b055c31e93b0","version_major":2,"version_minor":0}

```
print(f'Среднее: {np.mean(n_mm_list)}')
print(f'Смещение: {np.mean(np.abs(n_mm_list - n))}')
print(f'Дисперсия: {np.var(n_mm_list)}')
print(f'Среднеквадратичная ошибка: {mean_squared_error([100] * 10000,
n_mm_list)}')
```

```
Среднее: 109.7593
Смещение: 80.8179
Дисперсия: 12358.55596351
Среднеквадратичная ошибка: 12453.7999
```

```
plt.figure(figsize = (10, 8))
plt.hist(n_mm_list, bins = 50)
```

```
plt.xlabel('количество таксистов')
plt.ylabel('count')
plt.title('Гистограмма ММ оценок количества таксистов')
plt.show()
```



№2

Однажды в Самарканде турист заказывал Яндекс-такси. На десятом заказе он обнаружил, что у таксистов было 6 разных имён. Для упрощения предположим, что все n имён среди таксистов встречаются равновероятно и независимо от поездки к поездке.

а)

_[5] Постройте график функции правдоподобия как функции от общего количества имён

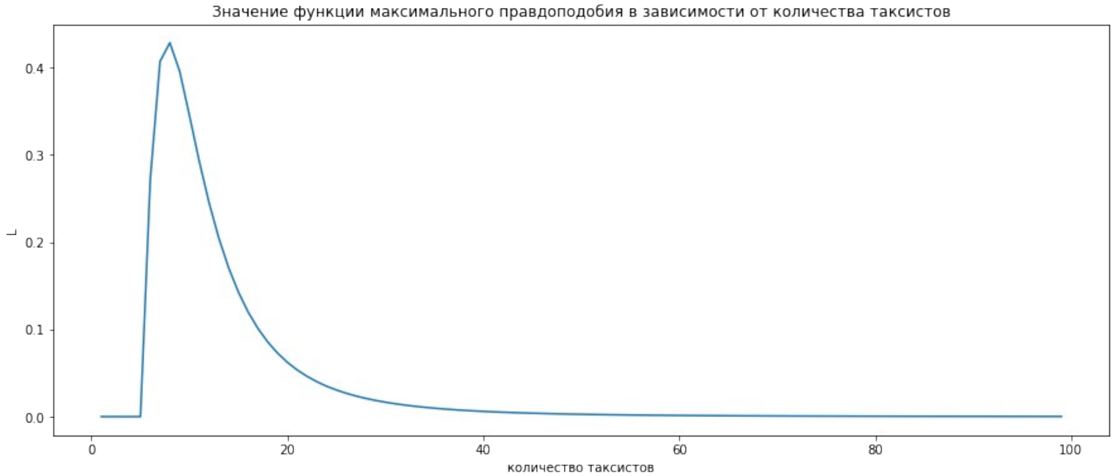
n. Найдите оценку числа n методом максимального правдоподобия._

В общем, тут надо построить дерево с ходами. Ход вправо - новый таксист, влево - имеющийся. Тогда ходов влево должно быть days - unique. Ходов вправо = unique. Поэтому, т.к. добавление i-го таксиста происходит с вероятностью $\frac{n-i+1}{n}$, то в нашей вероятности будет произведение следующее: $\Pi\left[\frac{n-i}{n}\right)$, i from 0 to unique - 1. Далее каждый ход влево при i имеющихся таксистов с вероятностью $\frac{i}{n}$. Поэтому должна быть сумма произведений из days - unique множителей от 1 до days - unique. Потому что мы идем от вероятностей $\frac{1}{n}, $ до $\frac{days-unique}{n}$. Поэтому такая формула, как в ячейке

```
import itertools
def mle(n, days, unique):
    L = 1
    for i in range(1, unique):
        L *= ((n-i)/n)
    combinations = 
itertools.combinations_with_replacement(np.arange(1, unique+1), days - 
unique)
    cnt = 0
    for combination in combinations:
        mult = 1
        for i in range(days - unique):
            mult *= combination[i]
        cnt += mult
    L *= (cnt / (n ** (days - unique)))
    return L

mle_vec = np.vectorize(mle)
n = np.arange(1, 100)
days = 10
unique_names = 6
L_2 = mle(n, days, unique_names)

plt.figure(figsize = (15, 6))
plt.plot(n, L_2)
plt.xlabel("количество таксистов")
plt.ylabel("L")
plt.title("Значение функции максимального правдоподобия в зависимости 
от количества таксистов")
plt.show()
```




```
n_ml_2 = np.argmax(L_2) + 1
print(f'n_ml = {n_ml_2}')
```

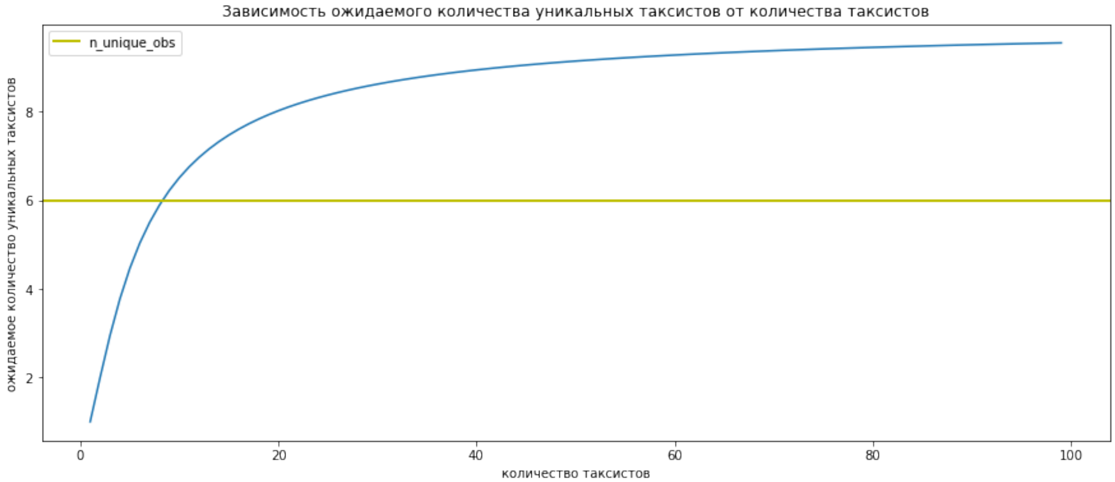
```
n_ml = 8
```

б)

[5] Постройте график математического ожидания числа разных имён у 10 таксистов, как функции от общего количества имён n. Найдите оценку числа n методом моментов.

```
def ev(n, day):
    E = 0
    for unique_names in range(day+1):
        P = mle(n, day, unique_names)
        E += P * unique_names
    return E

n = np.arange(1, 100)
days = 10
ev_vec = np.vectorize(ev)
E_2 = ev_vec(n, days)
plt.figure(figsize = (15, 6))
plt.plot(n, E_2)
plt.axhline(6, c='y', linewidth=2, label = 'n_unique_obs')
plt.xlabel('количество таксистов')
plt.ylabel('ожидаемое количество уникальных таксистов')
plt.title('Зависимость ожидаемого количества уникальных таксистов от
количества таксистов')
plt.legend()
plt.show()
```




```
n_mm_2 = np.argmin(np.abs(E_2 - 6)) + 1
print(f'n_mm = {round(n_mm_2, 2)}')
```

```
n_mm = 8
```

в)

[15] Предположим, что настоящее 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.

```
def num_of_unique_taxists(n):
    return len(set(sts.randint(0, n).rvs(10)))
```

ml оценки

```
np.random.seed(42)
n_sim_2 = 10000
n_obs_2 = []
n = 20
for n_i in tqdm(range(n_sim_2)):
    n_obs_2.append(num_of_unique_taxists(n))
n_obs_2 = np.array(n_obs_2)
```

{"model_id":"f7ca76e4c5224eb6aee28bd7690f7187","version_major":2,"version_minor":0}

```
import warnings
warnings.filterwarnings("ignore")
n_ml_list_2 = []
for i in tqdm(n_obs_2):
    n_ml_list_2.append(np.argmax(mle_vec(np.arange(1, 100), days, i))
```

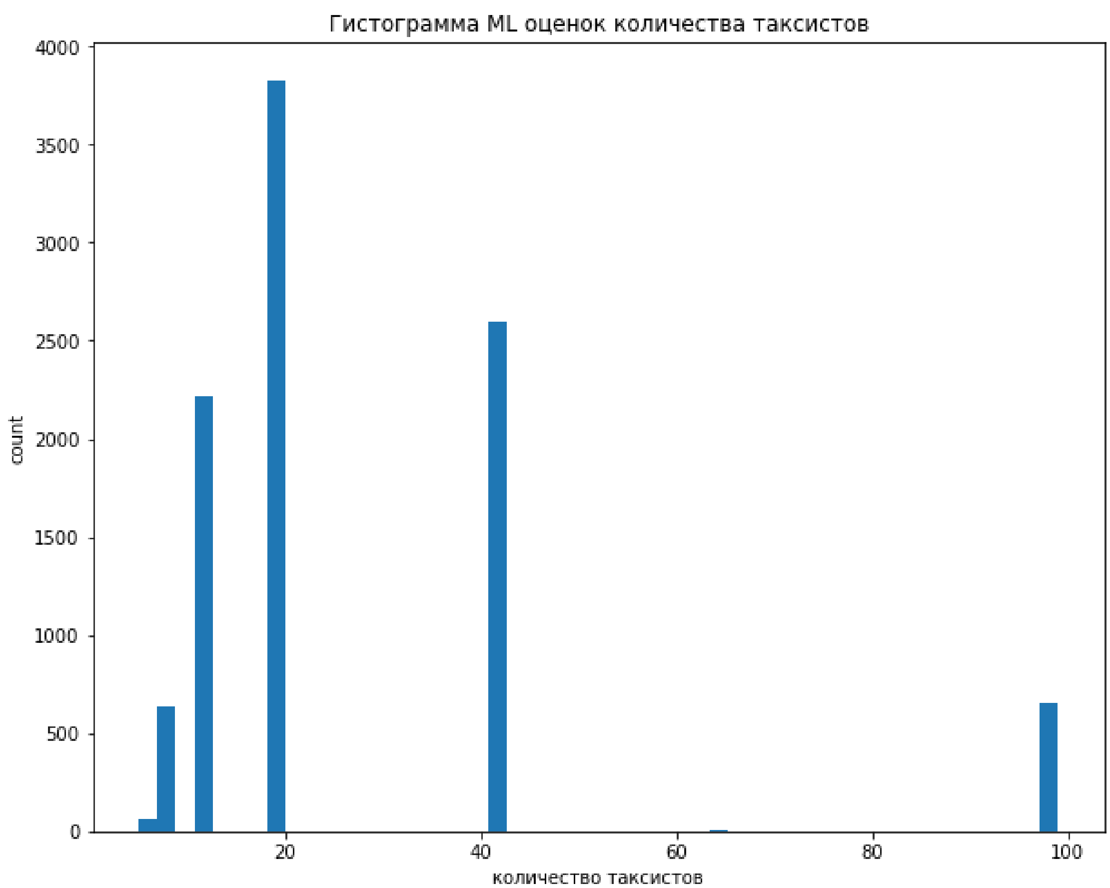
```
+ 1)
n_ml_list_2 = np.array(n_ml_list_2)
```

{"model_id":"4a280042e7dc4b1fb027375c1d30275d","version_major":2,"version_minor":0}

```
print(f'Среднее: {np.mean(n_ml_list_2)}')
print(f'Смещение: {np.mean(np.abs(n_ml_list_2 - n))}')
print(f'Дисперсия: {np.var(n_ml_list_2)}')
print(f'Среднеквадратичная ошибка: {mean_squared_error([100] * 10000,
n_ml_list_2)}')
```

```
Среднее: 27.8563
Смещение: 13.8915
Дисперсия: 496.40365031000005
Среднеквадратичная ошибка: 5701.1171
```

```
plt.figure(figsize = (10, 8))
plt.hist(n_ml_list_2, bins = 50)
plt.xlabel('количество таксистов')
plt.ylabel('count')
plt.title('Гистограмма ML оценок количества таксистов')
plt.show()
```



mm оценки

```
s = n_obs_2[0]
ev_vec(np.arange(1, 200), s)
```

```
array([1.        , 1.984375  , 2.82441701, 3.46606445, 3.951424  ,
       4.32551012, 4.62058326, 4.85843277, 5.05383853, 5.217031  ,
       5.3552607 , 5.47378586, 5.57650178, 5.66635009, 5.74559087,
       5.81598765, 5.87893574, 5.93555302, 5.9867453 , 6.03325408,
       6.07569207, 6.11456989, 6.15031642, 6.18329448, 6.21381305,
       6.24213686, 6.26849404, 6.29308223, 6.31607347, 6.33761823,
       6.35784868,        nan, 6.39481954, 6.41175482, 6.42776896,
       6.44293505, 6.45731867, 6.47097878, 6.48396861, 6.49633625,
       6.50812536, 6.51937555, 6.53012294, 6.54040047, 6.55023828,
       6.55966398, 6.56870292, 6.57737842, 6.58571195, 6.59372334,
       6.60143089, 6.60885157, 6.61600109, 6.62289403, 6.62954398,
       6.63596354, 6.64216446, 6.64815777, 6.65395368, 6.65956179,
       6.66499104, 6.67024998, 6.67534636,        nan, 6.68508082,
       6.68973244, 6.69424867, 6.69863535, 6.70289795, 6.70704169,
       6.71107145, 6.71499188, 6.71880737, 6.72252195, 6.7261398 ,
       6.72966454, 6.73309972, 6.73644871, 6.7397147 , 6.74290075,
       6.74600973, 6.74904441, 6.75200736, 6.75490055, 6.75772972,
       6.76049188, 6.76319218, 6.76583236, 6.76841434, 6.77094002,
       6.77341122, 6.77582956, 6.77819693, 6.78051482, 6.78278476,
              nan, 6.7871388 , 6.78932191, 6.79141412, 6.79346528,
       6.79547653, 6.79744887, 6.79938366, 6.80128175, 6.80314352,
       6.8049728 , 6.80676704, 6.80852867, 6.81025837, 6.81195719,
       6.81362565, 6.81526567, 6.81687627, 6.81845917, 6.82001482,
       6.82154401, 6.82304815, 6.82452643, 6.82598025, 6.82741026,
       6.82881791, 6.83020128, 6.83156294, 6.832903  , 6.83422244,
       6.83552057, 6.83679857,        nan, 6.8392964 , 6.8405166 ,
       6.841717  , 6.84290275, 6.84406899, 6.8452188 , 6.84635078,
       6.84746651, 6.84856667, 6.84965059, 6.85071974, 6.85177295,
       6.85281066, 6.85383633, 6.85484613, 6.85584294, 6.85682528,
       6.85779521, 6.85874997, 6.85969517, 6.86062666, 6.86154514,
       6.86245215, 6.8633468 , 6.86423045, 6.86510279, 6.86596341,
       6.86681357, 6.8676531 , 6.86848139, 6.86930021,        nan,
       6.87090728, 6.87169444, 6.87247478, 6.87324455, 6.87400515,
       6.87475657, 6.8754991 , 6.87623291, 6.87695816, 6.87767505,
       6.87838364, 6.87908378, 6.87977577, 6.88045997, 6.88113648,
       6.88180542, 6.88246688, 6.88312087, 6.88376769, 6.88440184,
       6.88504049, 6.88566629, 6.88628519, 6.88689803, 6.88750369,
       6.88810285, 6.88869621, 6.8892828 , 6.88986373, 6.89043804,
       6.89100691,        nan, 6.89212656, 6.89267625, 6.89322335,
       6.89376367, 6.89429866, 6.89482747, 6.89535166])
```

```
E_2 = ev_vec(np.arange(1, 100), 10)
n_mm_list_2 = []
for i in tqdm(n_obs_2):
    n_mm_list_2.append(np.argmin(np.abs(E_2-i)) + 1)
n_mm_list_2 = np.array(n_mm_list_2)
```

{"model_id":"66ba875232574a3d970cae66bcf0f94b","version_major":2,"version_minor":0}

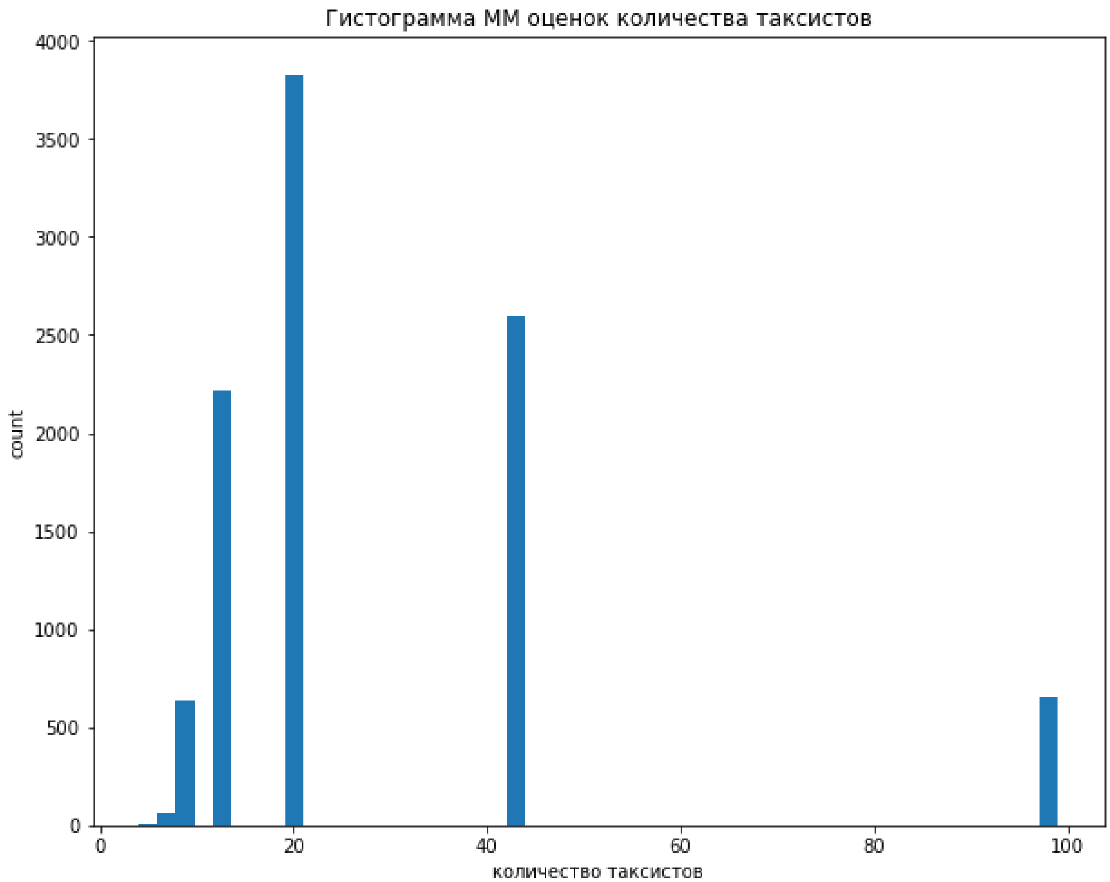
```
pd.DataFrame(n_mm_list_2)[0].value_counts()
```

```
20    3830
42    2593
12    2220
99     651
8      638
6       62
4        6
Name: 0, dtype: int64
```

```
print(f'Среднее: {np.mean(n_mm_list_2)}')
print(f'Смещение: {np.mean(np.abs(n_mm_list_2 - n))}')
print(f'Дисперсия: {np.var(n_mm_list_2)}')
print(f'Среднеквадратичная ошибка: {mean_squared_error([100] * 10000,
n_mm_list_2)}')
```

```
Среднее: 28.2095
Смещение: 13.4855
Дисперсия: 489.1584097500001
Среднеквадратичная ошибка: 5643.0343
```

```
plt.figure(figsize = (10, 8))
plt.hist(n_mm_list_2, bins = 50)
plt.xlabel('количество таксистов')
plt.ylabel('count')
plt.title('Гистограмма ММ оценок количества таксистов')
plt.show()
```



№3

Иноагент Иннокентий по 20 наблюдениям строит 95%-й доверительный интервал для математического ожидания несколькими способами: классический асимптотический нормальный интервал, с помощью наивного бутстрэпа, с помощью бутстрэпа t-статистики.

а)

[15] Для каждого способа с помощью 10000 симуляций оцените вероятность того, что номинально 95%-й доверительный интервал фактически накрывает математическое ожидание, если наблюдения распределены экспоненциально с интенсивностью 1.

```
n_sim = 10000
samples = sts.expon.rvs(scale=1, size=(n_sim, sample_size),
random_state = 42)
```

- асимптотический нормальный интервал

```
def as_norm_CI(sample, sample_size = 20, alpha = 0.05):
    interval = sts.norm.interval(1-alpha, np.mean(sample),
np.std(sample, ddof = 1)/np.sqrt(sample_size))
```

```
    return interval

as_norm = []
for sample in tqdm(samples):
    is_in_interval = ((as_norm_CI(sample)[0] < 1) &
(as_norm_CI(sample)[1] > 1))
    as_norm.append(is_in_interval)
```

{"model_id":"3c58e5d44df145359e983373d258a007","version_major":2,"version_minor":0}

```
print(f'Вероятность накрытия математического ожидания асимптотическим
нормальным ДИ - {np.mean(as_norm)}')
```

```
Вероятность накрытия математического ожидания асимптотическим
нормальным ДИ - 0.9036
```

- наивный бутстрап

```
np.random.seed(42)
def naive_boot_CI(sample, n_boots = 10000, sample_size = 20, alpha =
0.05):
    boot = np.random.choice(sample, size = (n_boots, sample_size),
replace = True)
    boot_mean = np.mean(boot, axis = 1)
    q_L, q_R = np.quantile(boot_mean, alpha / 2),
np.quantile(boot_mean, 1 - (alpha / 2))
    return (q_L, q_R)

naiv_boot = []
for sample in tqdm(samples):
    is_in_interval = ((naive_boot_CI(sample)[0] < 1) &
(naive_boot_CI(sample)[1] > 1))
    naiv_boot.append(is_in_interval)
```

{"model_id":"281b0f1fde46453c8130be62a9b83706","version_major":2,"version_minor":0}

```
print(f'Вероятность накрытия математического ожидания ДИ, построенным
наивным бутстрэпом - {np.mean(naiv_boot)}')
```

```
Вероятность накрытия математического ожидания ДИ, построенным наивным
бутстрэпом - 0.9035
```

- бутстрэп t-статистики

```
np.random.seed(42)
def t_boot_CI(sample, n_boots = 10000, sample_size = 20, alpha =
0.05):
    mu_hat = np.mean(sample)
    boot = np.random.choice(sample, size = (n_boots, sample_size),
replace = True)
    boot_mean = np.mean(boot, axis = 1)
```

```
    boot_se = np.std(boot, ddof = 1, axis = 1)
    R = (boot_mean - mu_hat) / (boot_se / np.sqrt(sample_size))
    ql, qr = np.quantile(R, alpha / 2), np.quantile(R, 1 - (alpha /
2))
    q_L, q_R = mu_hat - qr * (np.std(sample,
ddof=1)/np.sqrt(sample_size)), mu_hat - ql * (np.std(sample,
ddof=1)/np.sqrt(sample_size))
    return (q_L, q_R)

t_boot = []
for sample in tqdm(samples):
    is_in_interval = ((t_boot_CI(sample)[0] < 1) & (t_boot_CI(sample)
[1] > 1))
    t_boot.append(is_in_interval)
```

{"model_id":"498f17c477ea40cbabd0355ea9faaafd","version_major":2,"version_minor":0}

```
print(f'Вероятность накрытия математического ожидания ДИ, построенным
бутстрэпом t-статистики - {np.mean(t_boot)}')
```

```
Вероятность накрытия математического ожидания ДИ, построенным
бутстрэпом t-статистики - 0.9467
```

б)

[5] Пересчитайте вероятности накрытия, если наблюдения имеют распределение Стьюдента с тремя степенями свободы.

```
samples_t = sts.t.rvs(3, size=(n_sim, sample_size), random_state=42)
```

- асимптотический нормальный интервал

```
as_norm_t = []
for sample in tqdm(samples_t):
    is_in_interval = ((as_norm_CI(sample)[0] < 0) &
(as_norm_CI(sample)[1] > 0))
    as_norm_t.append(is_in_interval)
```

{"model_id":"237693231f23468bb7d04fa74bdf88dc","version_major":2,"version_minor":0}

```
print(f'Вероятность накрытия математического ожидания асимптотическим
нормальным ДИ - {np.mean(as_norm_t)}')
```

```
Вероятность накрытия математического ожидания асимптотическим
нормальным ДИ - 0.9438
```

- наивный бутстрап

```
naiv_boot_t = []
for sample in tqdm(samples_t):
    is_in_interval = ((naive_boot_CI(sample)[0] < 0) &
```

```
(naive_boot_CI(sample)[1] > 0))
    naiv_boot_t.append(is_in_interval)
```

{"model_id":"95a94695c310450aa6db00c8592869ef","version_major":2,"version_minor":0}

```
print(f'Вероятность накрытия математического ожидания ДИ, построенным наивным бутстрэпом - {np.mean(naiv_boot_t)}')
```

```
Вероятность накрытия математического ожидания ДИ, построенным наивным бутстрэпом - 0.9202
```

- бутстрэп t-статистики

```
t_boot_t = []
for sample in tqdm(samples_t):
    is_in_interval = ((t_boot_CI(sample)[0] < 0) & (t_boot_CI(sample)[1] > 0))
    t_boot_t.append(is_in_interval)
```

{"model_id":"717755f440e448eb9be4d0f70928d3e4","version_major":2,"version_minor":0}

```
print(f'Вероятность накрытия математического ожидания ДИ, построенным бутстрэпом t-статистики - {np.mean(t_boot_t)}')
```

```
Вероятность накрытия математического ожидания ДИ, построенным бутстрэпом t-статистики - 0.9245
```

Выводы:

- *Для экспоненциального распределения лучший результат дал бутстрэп t-статистики*
- *Для распределения стьюдента лучший результат показал асимптотический нормальный интервал*

№4

Проверьте гипотезу о том, что ожидаемые результаты экзамена по теории вероятностей тех, у кого фамилия начинается с гласной буквы и с согласной буквы, равны. В качестве альтернативной гипотезы возьмите гипотезу о неравенстве.

```
df = pd.read_csv('22-23_hse_probability - Exam.csv', skiprows = 5)[['Unnamed: 1', 'Unnamed: 72']]
df.columns = ['surname', 'grade']

def first_vowel(surname):
    if surname[0] in set(['У', 'Е', 'Ы', 'А', 'О', 'Э', 'Я', 'И', 'Ю']):
        return 1
    return 0
first_vowel_vec = np.vectorize(first_vowel)
```

```
mask = first_vowel_vec(df['surname']) * df['surname']
vowel_surnames = mask.values[mask.values != '']
```

```
df_v = df[df['surname'].isin(vowel_surnames)]
df_c = df[~df['surname'].isin(vowel_surnames)]
```

$$H_0 : mu_v = mu_c \\ H_a : mu_v \ne mu_c $$

где $mu_i - $ средний результат за экзамен у $i-$ой выборки

а)

[5] Используйте тест Уэлча

```
from scipy.stats import ttest_ind
p_value = ttest_ind(df_c['grade'], df_v['grade'], equal_var=False)[1]
var_c = df_c['grade'].var(ddof = 1)
var_v = df_v['grade'].var(ddof = 1)
n_c, n_v = df_c.shape[0], df_v.shape[0]
var_0 = (var_c / n_c + var_v / n_v)**2 / (var_c ** 2 / ((n_c - 1) *
n_c) + var_v ** 2 / ((n_v -1) * n_v))
diff = np.mean(df_v['grade']) - np.mean(df_c['grade'])
l, r = diff - sts.t.ppf(df = n_c + n_v - 2, q= 1-(alpha/2)) *
np.sqrt(var_0 / n_c + var_0 / n_v), diff + sts.t.ppf(df = n_c + n_v -
2, q= 1 - (alpha/2)) * np.sqrt(var_0 / n_c + var_0 / n_v)
ne = ' не' * (p_value > 0)
print(f'95% CI: [{l, r}]')
print(f'p_value = {round(p_value, 3)} => Гипотеза{ne} отвергается')
```

```
95% CI: [(-1.4230103062864228, -0.7334763166312921)]
p_value = 0.397 => Гипотеза не отвергается
```

Какая-то лажа получилась с ДИ, но p_value правильно посчитал

б)

[5] Используйте наивный бутстрэп.

```
df_c.shape[0]
```

```
283
```

```
np.random.seed(43)
vow_boot = np.random.choice(df_v['grade'], size=(n_sim,
df_v.shape[0]))
con_boot = np.random.choice(df_c['grade'], size=(n_sim,
df_c.shape[0]))
diff_hat = np.mean(df_v['grade']) - np.mean(df_c['grade'])
diff_boot = np.mean(vow_boot, axis = 1) - np.mean(con_boot, axis = 1)

l, r = np.percentile(diff_boot, 2.5), np.percentile(diff_boot, 97.5)
p_value = 2 * min(np.mean((np.array(diff_boot) > 0)),
np.mean((np.array(diff_boot) <= 0)))
```

```
ne = ' не' * (l <= 0 <= r)
print(f'95% CI: {l, r}')
print(f'p_value = {round(p_value, 3)} => Гипотеза{ne} отвергается')
```

```
95% CI: (-3.529972236244321, 1.3649293286219053)
p_value = 0.386 => Гипотеза не отвергается
```

в)

[5] Используйте бутстрэп t-статистики.

```
var_vow_boot, var_con_boot = np.var(vow_boot, ddof=1, axis=1),
np.var(con_boot, ddof=1, axis=1)
var_vow, var_con = np.var(df_v['grade'], ddof=1),
np.var(df_c['grade'], ddof=1)
se_boot = np.sqrt(var_vow_boot/df_v.shape[0] +
var_con_boot/df_c.shape[0])
se = np.sqrt(var_vow/df_v.shape[0] + var_con/df_c.shape[0])
R= (diff_boot - diff_hat)/se_boot
R_hat = diff_hat / se

pvalue = 2 * min(np.mean(R<= R_hat), np.mean(R >= R_hat))
r, l = np.percentile(R, 2.5), np.percentile(R, 97.5)
ne = ' не' * (pvalue > alpha)
print(f'95% CI: {l, r}')
print(f'p_value = {round(pvalue, 3)} => Гипотеза{ne} отвергается')
```

```
95% CI: (2.0887261081750292, -1.9044655238939623)
p_value = 0.375 => Гипотеза не отвергается
```

г)

[5] Используйте перестановочный тест.

```
from itertools import permutations

pt_list = []
for i in range(n_sim):
    pt_df = np.random.permutation(np.array(df['grade']))
    pt_c, pt_v = pt_df[:n_c], pt_df[n_c:]
    pt_mean = pt_v.mean() - pt_c.mean()
    pt_list.append(pt_mean)
pt_list = np.array(pt_list)

pvalue = 2 * min(np.mean(pt_list > diff), np.mean(pt_list <= diff))
l, r = np.percentile(pt_list, 2.5), np.percentile(pt_list, 97.5)
ne = ' не' * (pvalue > alpha)
print(f'95% CI: {l, r}')
print(f'p_value = {round(pvalue, 3)} => Гипотеза{ne} отвергается')
```

```
95% CI: (-2.3950385808033463, 2.4172495853465055)
p_value = 0.379 => Гипотеза не отвергается
```

№5

Составьте таблицу сопряжённости, поделив студентов писавших экзамен на четыре группы по двум признакам: набрал ли больше медианы или нет, на согласную или гласную букву начинается фамилия.

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 P-значение

```
df = pd.DataFrame({'surname':df['surname'], 'grade': df['grade'],
'start_vow': df['surname'].isin(vowel_surnames), 'more_tha_med':
df['grade'] >
np.median(df['grade'])}).reset_index().drop(columns=['index'])
table = pd.crosstab(df['more_tha_med'], df['start_vow'])
```

```
table
```

```
start_vow     False  True
more_tha_med
False           138    28
True            145    21
```

```
con_less, vow_less, con_more, vow_more = table.iloc[0, 0],
table.iloc[0, 1], table.iloc[1, 0], table.iloc[1, 1]

odds_vow = vow_more / vow_less
odds_con = con_more / con_less
ln_OR = np.log(odds_vow / odds_con)
se_ln_OR = np.sqrt(1/con_less + 1/vow_less + 1/con_more + 1/vow_more)
l_OR, r_OR = np.exp(ln_OR-1.96*se_ln_OR), np.exp(ln_OR+1.96*se_ln_OR)

z_obs = (ln_OR) / se_ln_OR
p_value = 2*(sts.norm.cdf(z_obs))
ne = ' не' * (CI_ln_OR[0] <= 0 <= CI_ln_OR[1])
print(f'95% CI: [{l_OR, r_OR}]')
print(f'p_value = {round(p_value, 3)} => Гипотеза{ne} отвергается')
```

```
95% CI: [(0.3870902431823096, 1.3162320763800788)]
p_value = 0.28 => Гипотеза не отвергается
```

б)

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

Вместо того, чтобы рассматривать отношение долей, можно рассматривать разность их логарифмов. Вопользуемся дельта методом для $\ln \hat{p}$ в окрестности точки $p : \ln \hat{p} \approx \ln p + \frac{1}{p} \cdot (\hat{p} - p)$.

Значит, $E(\ln \hat{p}) \approx \ln p$, $D(\ln \hat{p}) = \frac{1}{p^2} D(\hat{p}) = \frac{p \cdot (1-p)}{p^2 \cdot n} = \frac{(1-p)}{n p}$.

Т.к. выборки хороших и плохих оценок независимы, значит дисперсия разности будет суммой дисперсией, а матожидание разности будет разностью матожиданий в силу линейности. Поэтому:

$$ E(\frac{\ln{p_{wov}}}{\ln{p_{con}}}) = \ln{p_{wov}} - \ln{p_{con}} \\ D(\frac{\ln{p_{wov}}}{\ln{p_{con}}}) = \frac{(1-p_{wov})}{np_{wov}} + \frac{(1-p_{con})}{np_{con}} $$

```
p_w = vow_more / (vow_less + vow_more)
p_c = con_more / (con_less + con_more)
log_ratio = np.log(p_w / p_c)
se_log_ratio = np.sqrt((1 - p_w) / (p_w * (vow_more + vow_less)) + (1
- p_c) / (p_c * (con_more + con_less)))
l,r = np.exp(log_ratio -sts.norm.ppf(1- alpha/2) * se_log_ratio),
np.exp(log_ratio + sts.norm.ppf(1- alpha/2) * se_log_ratio)
print(f'95% CI: {round(l, 3), round(r, 3)}')
z_obs = (log_ratio)/se_log_ratio
p_value = 2*(sts.norm.cdf(z_obs))
ne = ' не' * (CI_ln_OR[0] <= 0 <= CI_ln_OR[1])
print(f'p_value = {round(p_value, 3)} => Гипотеза{ne} отвергается')
```

```
95% CI: (0.594, 1.178)
p_value = 0.307 => Гипотеза не отвергается
```

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.

```
vow_boot = np.random.choice(df_v['grade'], size=(n_sim, n_v))
con_boot = np.random.choice(df_c['grade'], size=(n_sim, n_c))
con_less_boot, con_more_boot, vow_less_boot, vow_more_boot = (con_boot
< med).sum(axis = 1), (con_boot > med).sum(axis = 1), (vow_boot <
med).sum(axis = 1), (vow_boot > med).sum(axis = 1)
OR_boot = (vow_more_boot / vow_less_boot) / (con_more_boot /
con_less_boot)
```

```
pvalue = 2 * min(np.mean(OR_boot > np.exp(ln_OR)), np.mean(OR_boot <=
np.exp(ln_OR)))
l, r = np.percentile(OR_boot, 2.5), np.percentile(OR_boot, 97.5)
ne = ' не' * (pvalue > alpha)
```

```
print(f'95% CI: {l, r}')
print(f'p_value = {round(pvalue, 3)} => Гипотеза{ne} отвергается')
```

```
95% CI: (0.37485582468281425, 1.3053613053613053)
p_value = 0.984 => Гипотеза не отвергается
```

№6

Иноагент Иннокентий Вероятностно-Статистический считает, что длина фамилии положительно влияет на результат экзамена по теории вероятностей. А именно, он предполагает, что ожидаемый результат за экзамен прямо пропорционален длине фамилии, $E(Y_i) = \beta F_i$, где Y_i — результат за экзамен по 30-балльной шкале, F_i — количество букв в фамилии.

а)

[10] Оцените β методом моментов. Рассчитайте выборочную корреляцию.

```
df['len'] = df['surname'].str.len()
beta_hat_mm = (df['grade'].mean())/(df['len'].mean())
print(f'ММ оценка = {beta_hat_mm}')
corr = np.corrcoef(df['grade'], df['len'])
print(f'Выборочная корреляция = {corr[0][1]}')
```

```
ММ оценка = 2.0613026819923372
Выборочная корреляция = 0.0253280526691477
```

№7

[10] С помощью chatgpt решите любую задачу из нашего курса теории вероятностей и статистики. Можно брать задачи из прошлых контрольных, лекций, семинаров и даже этого домашнего задания. В качестве ответа приведите полный диалог с chatgpt.

Лог моего диалога с Chat-GPT

№8

[5] Укажите любой источник по теории вероятностей или статистике, который вам оказался полезен в течение года. Это может быть статья, видео, задача, всё что угодно. Объясните, с чем конкретно этот источник помог разобраться. Лучше привести в пример внешний источник, не упомянутый на вики курса, но можно и внутренний.

Помимо нашего курса и курса Фила я занимался по учебникам, которые кидали в Поступашки. Но вообще, топ моих источников такой:

- Вообще классный учебник - я его читаю параллельно с курсом, очень помог к экзамену. Там немного перебор с математикой, но в целом норм, я его всем советую
- Вот еще учебник со всякими идейными и теоретическими приколами. Мне очень понравилось его читать и разбираться, когда готовился к устной части экзамена
- Немного еще смотрел сюда, чисто, чтобы базу вспомнить
- Еще есть слитые курсы с ШАДа по AB-тестам, но их прикреплять не буду, а то еще удалят (вдруг кто из проверяющих в Яндексе работает)
- Также я еще добавлял в этом году в свой учебный план Дискретную математику с ФКН ПМИ и там научился хорошо решать комбу и всякие счетные тяжкие задачки на теорию вероятностей. Ну и прикольные факты из теорвера оттуда узнал, например парадокс Симпсона. Вот учебник с конспектом лекций.