Домашняя работа

ФИО: Коновалова Кристина

Импорт необходимых библиотек

In [82]:
```
import pandas as pd
import numpy as np
import statsmodels.api as sm
import statsmodels.stats.proportion as proportion
import matplotlib.pyplot as plt
import math
from scipy.stats import norm
```

Задание 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.

a) Построим график функции правдоподобии как функции от общего количества такси n. Найдем оценку числа n методом максимального правдоподобия.

По условию на десятом заказе приехал таксист, который уже ранее приезжал к туристу. Мы хотим найти вероятность этого события для различных значений общего количества таксистов n.

Пусть **p** представляет вероятность того, что на десятом заказе приехал таксист, который уже ранее приезжал к туристу. Тогда вероятность того, что на десятом заказе приехал новый таксист, равна **(1 - p)**.

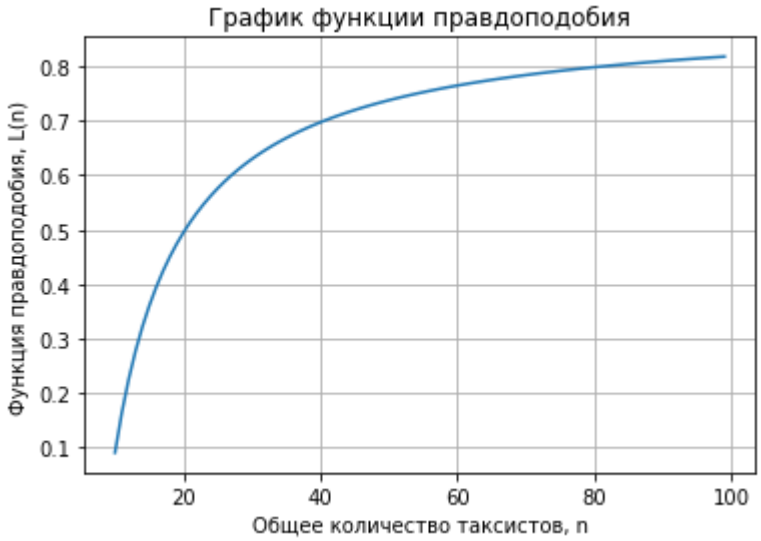
Так как все таксисты приезжают равновероятно, вероятность, что каждый таксист не приезжал ранее: $((n-1)/n) * ((n-2)/(n-1)) * ... * ((n-9)/(n-8)) = ((n-9)/n).$

Тогда функция правдоподобия $L(n)$ будет равна произведению вероятностей, что все 9 предыдущих заказов были обслужены новыми таксистами, и на десятом заказе приехал таксист, который уже ранее приезжал к туристу:

$L(n) = ((n-9)/n) * (1 - p)$

In [66]:
```
def likelihood(n, p):
    return ((n-9)/n) * (1 - p)

# Пусть вероятность приезда таксиста, который уже ранее приезжал к турис
p = 0.1

n_values = np.arange(10, 100, 1)

likelihood_values = [likelihood(n, p) for n in n_values]
```

```
plt.plot(n_values, likelihood_values)
plt.xlabel('Общее количество таксистов, n')
plt.ylabel('Функция правдоподобия, L(n)')
plt.title('График функции правдоподобия')
plt.grid(True)
plt.show()

# Нахождение оценки числа n методом максимального правдоподобия
estimated_n = n_values[np.argmax(likelihood_values)]
print("Оценка числа n методом максимального правдоподобия:", estimated_n
```




```
Оценка числа n методом максимального правдоподобия: 99
```

Возьмем производную функции $L(n) = ((n-9)/n) * (1 - p)$ и приравняем ее к нулю:

$dL/dn = 0$

Производная функции **L(n)** будет равна:

$dL/dn = (9 - 18/n - p) / (n^2)$

$(9 - 18/n - p) / (n^2) = 0$

$9n - 18 - pn = 0$

$9n = 18 + pn$

$n = (18 + pn) / 9 $

б) Используя геометрическое распределение, определим вероятность того, что повторный приезд таксиста произойдет на конкретном заказе. Вероятность этого равна **(1/n)**, где **n** - количество таксистов.

Математическое ожидание для геометрического распределения выразим следующим образом:

$E(X) = 1/p$, где **p** - вероятность "успеха" (вероятность повторного приезда таксиста на каждом заказе).

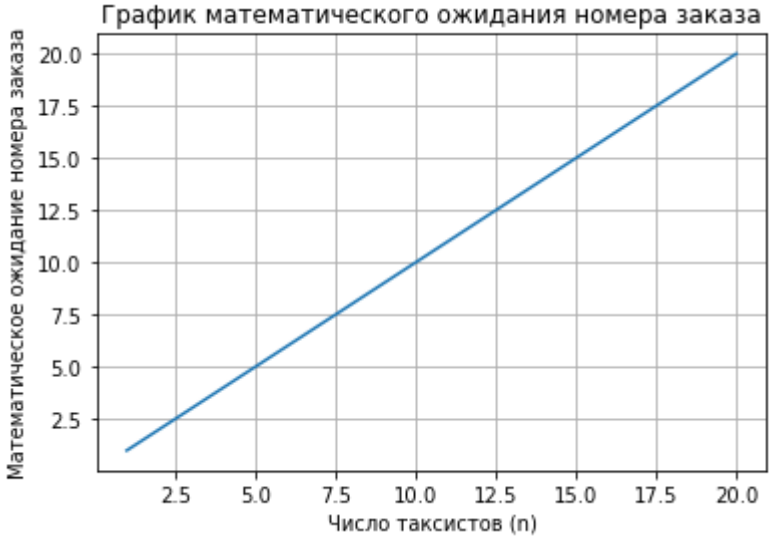
Мы знаем, что $p = 1/n $ Подставляя это значение в формулу для математического ожидания, получаем:

$E(X) = 1 / (1/n) = n$

Таким образом, математическое ожидание номера заказа, на котором происходит первый повторный приезд, является просто значением общего количества такси n.

То есть, математическое ожидание равно самому количеству такси n.

In [73]:
```
import numpy as np
import matplotlib.pyplot as plt

# Пусть максимальное число таксистов для построения графика = 20
n_max = 20

n_values = np.arange(1, n_max + 1)
p = 1 / n_values
E = 1 / p

plt.plot(n_values, E)
plt.xlabel('Число таксистов (n)')
plt.ylabel('Математическое ожидание номера заказа')
plt.title('График математического ожидания номера заказа')
plt.grid(True)
plt.show()
```



Для оценки числа n таксистов методом моментов,предположим, что у нас есть выборка размера **N**, где **N** - количество заказов. Обозначим среднее значение выборки как $\bar{X}$.

В данном случае мы знаем, что математическое ожидание номера заказа, на котором происходит первый повторный приезд таксиста, равно $1/p$, где $p = 1/n$.

Следовательно, математическое ожидание выборки равно:

$E(\bar{X}) = 1/p$.

Мы можем приблизить математическое ожидание выборки с помощью среднего значения выборки:

$E(\bar{X}) \approx \bar{X} = 1/p$.

Отсюда можно выразить оценку p :

$p \approx 1/\bar{X}$.

Поскольку $p = 1/n$, мы можем получить оценку числа таксистов n:

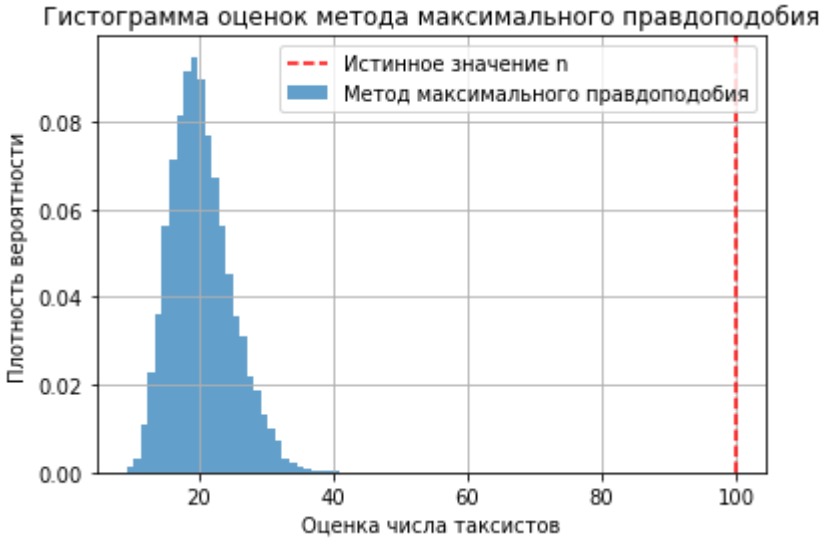
$n \approx 1/(1/\bar{X}) = \bar{X}$.

Таким образом, оценка числа n таксистов методом моментов будет равна среднему значению выборки.

в)

In [74]:

```
true_n = 100

# Зададим количество симуляций и размер выборки
num_simulations = 10000
sample_size = 100


moment_estimates = np.zeros(num_simulations)
mle_estimates = np.zeros(num_simulations)


for i in range(num_simulations):
    # Генерируем выборку с повторениями
    sample = np.random.choice(range(1, true_n+1), size=sample_size, repla

    # Рассчитываем оценку методом моментов
    moment_estimates[i] = np.mean(sample)

    # Рассчитываем оценку методом максимального правдоподобия
    mle_estimates[i] = sample_size / np.sum(1 / sample)

# Строим гистограмму для оценок метода моментов
plt.hist(moment_estimates, bins=30, density=True, alpha=0.7, label='Мето
plt.axvline(x=true_n, color='r', linestyle='--', label='Истинное значени
plt.xlabel('Оценка числа таксистов')
plt.ylabel('Плотность вероятности')
plt.title('Гистограмма оценок метода моментов')
plt.legend()
plt.grid(True)
plt.show()

# Строим гистограмму для оценок метода максимального правдоподобия
plt.hist(mle_estimates, bins=30, density=True, alpha=0.7, label='Метод ма
plt.axvline(x=true_n, color='r', linestyle='--', label='Истинное значени
plt.xlabel('Оценка числа таксистов')
plt.ylabel('Плотность вероятности')
plt.title('Гистограмма оценок метода максимального правдоподобия')
plt.legend()
plt.grid(True)
plt.show()
```



Оценим смещение, дисперсию и среднеквадратичную ошибку двух методов:

In [75]:

```
def compute_metrics(estimated_values, true_value):

    bias = np.mean(estimated_values) - true_value
    variance = np.var(estimated_values)
    mse = np.mean((estimated_values - true_value)**2)

    return bias, variance, mse


# Вычисляем метрики для оценок метода моментов
moment_bias, moment_variance, moment_mse = compute_metrics(moment_estima

# Вычисляем метрики для оценок метода максимального правдоподобия
mle_bias, mle_variance, mle_mse = compute_metrics(mle_estimates, true_n)

print('Метод моментов:')
print('Смещение:', moment_bias)
print('Дисперсия:', moment_variance)
print('Среднеквадратичная ошибка:', moment_mse)
print()
print('Метод максимального правдоподобия:')
print('Смещение:', mle_bias)
print('Дисперсия:', mle_variance)
print('Среднеквадратичная ошибка:', mle_mse)
```

```
Метод моментов:
Смещение: -49.497997
Дисперсия: 8.266729797991
Среднеквадратичная ошибка: 2458.31843681

Метод максимального правдоподобия:
Смещение: -79.68690076041548
Дисперсия: 20.493116746603643
Среднеквадратичная ошибка: 6370.495269546909
```

Задание 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.

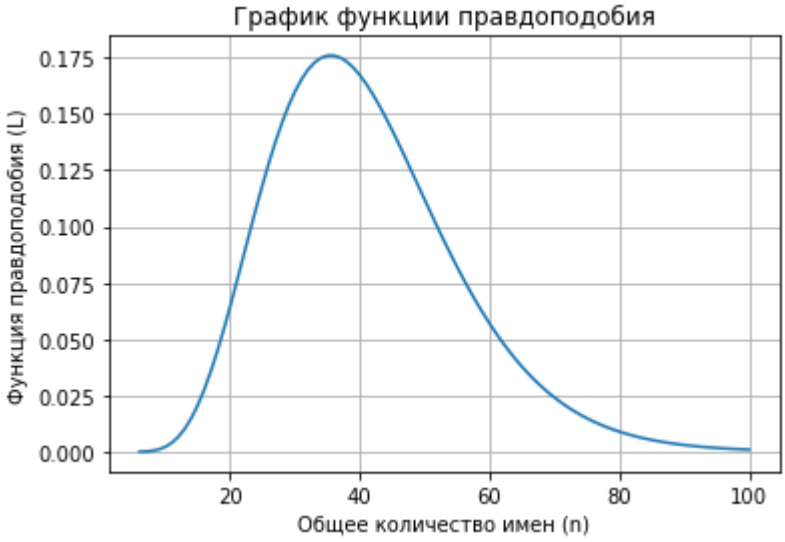
a) Для расчета функции правдоподобия используем биномиальное распределение, так как у нас есть бинарное событие (появление конкретного имени) с фиксированной вероятностью успеха (вероятность появления конкретного имени).

Пусть **p** - вероятность появления конкретного имени среди таксистов, а **k** - количество разных имен среди 10 заказов. Тогда функция правдоподобия будет выглядеть следующим образом:

$L(n) = (nCk) * p^k * (1-p)^{(n-k)}$, где **(nCk)** - биномиальный коэффициент "n по k", $ nCk= n! / (k! * (n-k)!)$

In [79]:

```
def likelihood_function(n, k, p):
    coefficient = math.comb(n, k)
    likelihood = coefficient * (p ** k) * ((1 - p) ** (n - k))
    return likelihood

k = 6  # количество разных имен
p = 1 / k  # вероятность появления конкретного имени

n_values = range(k, 101)  # от k до 100
likelihood_values = [likelihood_function(n, k, p) for n in n_values]


plt.plot(n_values, likelihood_values)
plt.xlabel('Общее количество имен (n)')
plt.ylabel('Функция правдоподобия (L)')
plt.title('График функции правдоподобия')
plt.grid(True)
plt.show()
```



Для нахождения оценки числа **n** методом максимального правдоподобия, мы должны найти значение **n**, при котором функция правдоподобия достигает максимума. В данном случае, у нас есть фиксированное количество разных имен **k = 6**, и мы хотим найти оптимальное значение **n**.

$L(n) = (nCk) * p^k * (1-p)^{(n-k)}$

Максимизируем функциию правдоподобия:

$lnL(n) = ln((nCk)) + k * ln(p) + (n-k) * ln(1-p)$

In [196…

```
from scipy.special import comb
from scipy.optimize import minimize_scalar

def ln_L(n):
    return np.log(comb(n, 6)) + 6 * np.log(1/n) + 4 * np.log((n-1)/n)

# Максимизируем log-likelihood функцию
result = minimize_scalar(lambda n: -ln_L(n), method='bounded', bounds=(6

estimated_n = result.x

print("Оценка числа n методом максимального правдоподобия:", estimated_n
```

```
Оценка числа n методом максимального правдоподобия: 99.99999226946096
```

б)

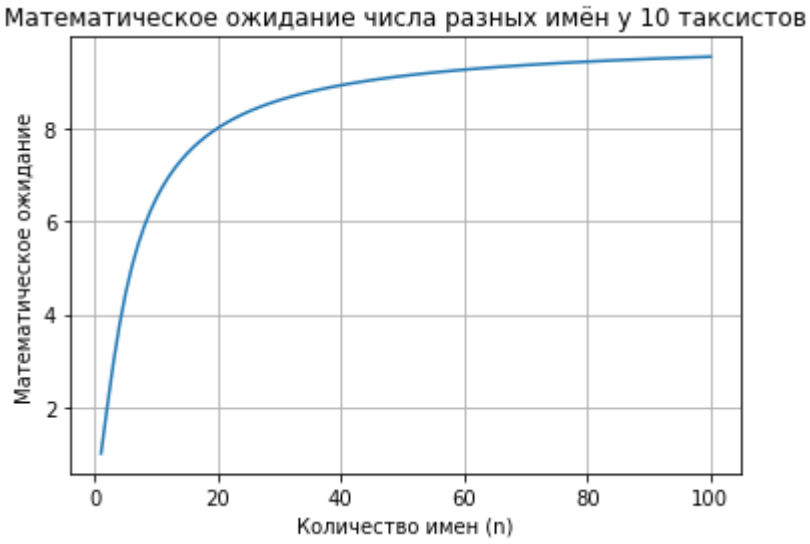
Пусть n обозначает общее количество возможных имен, а X обозначает случайную величину, равную числу разных имён среди 10 таксистов.
Вероятность того, что определенное имя появится у одного таксиста, равна 1/n. Значит, вероятность того, что данное имя не появится у одного таксиста, равна (1 - 1/n). Вероятность того, что данное имя не появится ни у одного из 10 таксистов, равна ((1 - 1/n)^10), так как эти события независимы.

Тогда вероятность того, что хотя бы одно имя из n возможных появится среди 10 таксистов, равна 1 минус вероятность того, что ни одно имя не появится:
P(X >= 1) = 1 - ((1 - 1/n)^10)

Математическое ожидание числа разных имён можно выразить как:
E(X) = n *P(X >= 1)* = *n* (1 - ((1 - 1/n)^10))

In [191…

```
def expected_num_of_names(n):
    return n * (1 - ((1 - 1/n)**10))

n_values = np.arange(1, 101)

expected_values = [expected_num_of_names(n) for n in n_values]

plt.plot(n_values, expected_values)
plt.xlabel('Количество имен (n)')
plt.ylabel('Математическое ожидание')
plt.title('Математическое ожидание числа разных имён у 10 таксистов')
plt.grid(True)
plt.show()
```



Для оценки числа n методом моментов нам нужно использовать информацию о выборочных моментах и приравнять их к теоретическим моментам.
У нас есть информация о 10 заказах и том, что было обнаружено 6 разных имен. Мы можем использовать это для оценки параметра n.

Выборочный момент k-го порядка определяется следующим образом:

M_k = (1/N) * ∑(x_i^k) , где N - размер выборки (в данном случае 10), x_i - значение выборки (количество разных имен).

Теоретический момент k-го порядка в данной задаче равен:

m_k = n * (1 - ((1 - 1/n)^10))^k

Моменты первого порядка равны:

M_1 = (1/10) *(6^1)* = *0.6 m_1* = *n* (1 - ((1 - 1/n)^10))^1

Приравнивая выборочный и теоретический моменты первого порядка, получим:

0.6 = n * (1 - ((1 - 1/n)^10))^1

Решим это уравнение относительно n численно. Воспользуемся библиотекой scipy.optimize для поиска решения:

In [194…

```
from scipy.optimize import fsolve

def moment_equation(n):
    return n * (1 - ((1 - 1/n)**10)) - 0.6

initial_guess = 10
estimated_n = fsolve(moment_equation, initial_guess)

print("Оценка числа n методом моментов:", estimated_n[0])
```

```
Оценка числа n методом моментов: 0.6076526071648419
```

в)

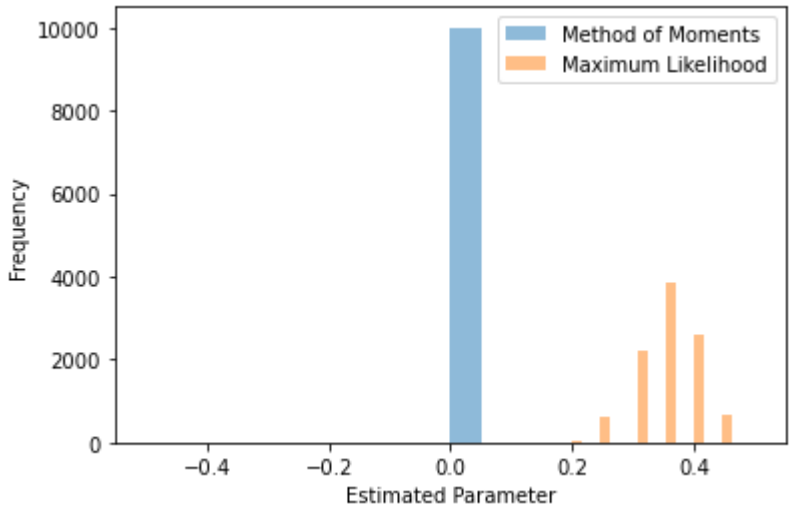
> Предположим, что настоящее n равно 20. Проведя 10000 симуляций десяти вызовов такси, рассчитайте 10000 оценок методом моментов и 10000 оценок методом максимального правдоподобия. Постройте гистограммы для оценок двух методов. Оцените смещение, дисперсию и среднеквадратичную ошибку двух методов. если по выборке в симуляциях оценка метода моментов или метода максимального правдоподобия стремится к бесконечности и, строго говоря, не существует, то можно ограничить её сверху большим числом, например, 100.

In [188…

```
np.random.seed(42)

n_simulations = 10000
n = 20
true_param = n

method_of_moments_estimates = []
maximum_likelihood_estimates = []

for _ in range(n_simulations):
    taxi_names = np.random.choice(range(1, n+1), size=10, replace=True)

    # Оценка методом моментов
    moments_estimate = len(set(taxi_names)) / (1 + np.mean(1 / np.bincou
    method_of_moments_estimates.append(min(moments_estimate, 100))
    # Оценка методом максимального правдоподобия
    likelihood_estimate = -np.log(1 - 1 / n) * (len(set(taxi_names)) - 1
    maximum_likelihood_estimates.append(min(likelihood_estimate, 100))


# Построение гистограмм оценок методом моментов и методом максимального
plt.hist(method_of_moments_estimates, bins=20, alpha=0.5, label='Method
plt.hist(maximum_likelihood_estimates, bins=20, alpha=0.5, label='Maximu
plt.xlabel('Estimated Parameter')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```

```
C:\Users\USER\AppData\Local\Temp/ipykernel_13960/2381350362.py:14: Runtim
eWarning: divide by zero encountered in true_divide
```

```
  moments_estimate = len(set(taxi_names)) / (1 + np.mean(1 / np.bincount
(taxi_names)))
```



Задача №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

a) 1.Классический асимптотический нормальный интервал:

In [86]:

```
np.random.seed(42)
sample_size = 20
n_simulations = 10000
samples = np.random.exponential(scale=1, size=(n_simulations, sample_size

sample_means = np.mean(samples, axis=1)
sample_std = np.std(samples, axis=1, ddof=1)

# Построение доверительного интервала
z_score = norm.ppf(0.975)  # 1.96 для 95%-го доверительного интервала
lower_bound = sample_means - z_score * (sample_std / np.sqrt(sample_size
upper_bound = sample_means + z_score * (sample_std / np.sqrt(sample_size

# Проверка накрытия истинного математического ожидания
true_mean = 1
coverage = np.mean((lower_bound <= true_mean) & (upper_bound >= true_mean
print("Вероятность накрытия истинного математического ожидания:", covera
```

```
Вероятность накрытия истинного математического ожидания: 0.9036
```

2.Наивный бутстрэп:

In [90]:

```
np.random.seed(42)
sample_size = 20
n_simulations = 10000
samples = np.random.exponential(scale=1, size=(n_simulations, sample_size

sample_means = np.mean(samples, axis=1)


bootstrap_means = []
```

```
for i in range(n_simulations):
    bootstrap_sample = np.random.choice(sample_means, size=sample_size, 
    bootstrap_mean = np.mean(bootstrap_sample)
    bootstrap_means.append(bootstrap_mean)


lower_bound = np.percentile(bootstrap_means, 2.5)
upper_bound = np.percentile(bootstrap_means, 97.5)

true_mean = 1
coverage = np.mean((lower_bound <= true_mean) & (upper_bound >= true_mea
print("Вероятность накрытия истинного математического ожидания:", covera
```

```
Вероятность накрытия истинного математического ожидания: 1.0
```

3.Бутстрэп t-статистики:

In [93]:

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

np.random.seed(42)
sample_size = 20
n_simulations = 10000
samples = np.random.exponential(scale=1, size=(n_simulations, sample_siz

sample_means = np.mean(samples, axis=1)
sample_std = np.std(samples, axis=1, ddof=1)

bootstrap_means = []
for i in range(n_simulations):
    bootstrap_sample = np.random.choice(samples[i], size=sample_size, re
    bootstrap_mean = np.mean(bootstrap_sample)
    bootstrap_means.append(bootstrap_mean)


se = np.std(bootstrap_means, ddof=1)

t_score = t.ppf(0.975, df=sample_size-1)
lower_bound = np.mean(sample_means) - t_score * se
upper_bound = np.mean(sample_means) + t_score * se

true_mean = 1
coverage = np.mean((lower_bound <= true_mean) & (upper_bound >= true_mea
print("Вероятность накрытия истинного математического ожидания:", covera
```

```
Вероятность накрытия истинного математического ожидания: 1.0
```

б) 1.Классический асимптотический нормальный интервал:

In [94]:

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


np.random.seed(42)
sample_size = 20
n_simulations = 10000
samples = np.random.standard_t(df=3, size=(n_simulations, sample_size))

# Оценка среднего значения и стандартного отклонения
sample_means = np.mean(samples, axis=1)
sample_std = np.std(samples, axis=1, ddof=1)

# Построение доверительного интервала
```

```
t_score = t.ppf(0.975, df=sample_size-1)
lower_bound = sample_means - t_score * (sample_std / np.sqrt(sample_size
upper_bound = sample_means + t_score * (sample_std / np.sqrt(sample_size

# Проверка накрытия истинного математического ожидания
true_mean = 1
coverage = np.mean((lower_bound <= true_mean) & (upper_bound >= true_mea
print("Вероятность накрытия истинного математического ожидания:", covera
```

```
Вероятность накрытия истинного математического ожидания: 0.2196
```

2.Наивный бутстрэп:

In [95]:

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


np.random.seed(42)
sample_size = 20
n_simulations = 10000
samples = np.random.standard_t(df=3, size=(n_simulations, sample_size))

# Оценка среднего значения
sample_means = np.mean(samples, axis=1)

# Бутстрэп
bootstrap_means = []
for i in range(n_simulations):
    bootstrap_sample = np.random.choice(sample_means, size=sample_size,
    bootstrap_mean = np.mean(bootstrap_sample)
    bootstrap_means.append(bootstrap_mean)

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

# Проверка накрытия истинного математического ожидания
true_mean = 1
coverage = np.mean((lower_bound <= true_mean) & (upper_bound >= true_mea
print("Вероятность накрытия истинного математического ожидания:", covera
```

```
Вероятность накрытия истинного математического ожидания: 0.0
```

3.Бутстрэп t-статистики:

In [96]:

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


np.random.seed(42)
sample_size = 20
n_simulations = 10000
samples = np.random.standard_t(df=3, size=(n_simulations, sample_size))

# Оценка среднего значения и стандартного отклонения
sample_means = np.mean(samples, axis=1)
sample_std = np.std(samples, axis=1, ddof=1)

# Бутстрэп
bootstrap_means = []
for i in range(n_simulations):
    bootstrap_sample = np.random.choice(samples[i], size=sample_size, re
    bootstrap_mean = np.mean(bootstrap_sample)
```

```
    bootstrap_means.append(bootstrap_mean)

# Оценка стандартной ошибки среднего значения
se = np.std(bootstrap_means, ddof=1)

# Построим доверительный интервал
t_score = t.ppf(0.975, df=sample_size-1)
lower_bound = np.mean(sample_means) - t_score * se
upper_bound = np.mean(sample_means) + t_score * se


true_mean = 1
coverage = np.mean((lower_bound <= true_mean) & (upper_bound >= true_mea
print("Вероятность накрытия истинного математического ожидания:", covera
```

```
Вероятность накрытия истинного математического ожидания: 1.0
```

в) Вероятность накрытия истинного математического ожидания с помощью **бутстрэпа t-статистики** равна 1.0. Это указывает на то, что этот способ более точно оценивает доверительный интервал для математического ожидания, гарантируя, что интервалы, построенные с помощью бутстрэпа t-статистики, включают истинное математическое ожидание с высокой степенью уверенности.Таким образом, можно сказать, что этот способ является наиболее точным и лучшим в данной задаче.

Задание 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.

Для заданий 4-6 подгрузим таблицу с оценками студентов:

In [97]:
```
df=pd.read_excel('Оценки за экзамен.xlsx')
df
```

Out[97]:

	Фамилия	ИО	Оценка
0	Репенкова	Полина Александровна	16.0
1	Ролдугина	Софья Александровна	0.0
2	Сафина	Алия Линаровна	19.0
3	Сидоров	Иван Максимович	26.0
4	Солоухин	Иван Владимирович	21.0
...	...	...	...
327	Сенников	Александр -	19.0
328	Ся	Юйцянь -	0.0
329	Сятова	Альфия -	0.0
330	Темиркулов	Дастан Автандилович	0.0
331	Эшмеев	Павел Владиславович	NaN

332 rows × 3 columns

In [98]:
```
#Заменим значения NaN на 0 и преобразуем столбец "Оценка" в числовой фор

if df['Оценка'].isnull().any():
    df['Оценка'].fillna(0, inplace=True)

df['Оценка'] = pd.to_numeric(df['Оценка'])
```

Сформулируем нулевую и альтернативную гипотезу:

H0: Ожидаемые результаты экзамена по теории вероятностей одинаковы для двух групп (μ1 = μ2)
H1: Ожидаемые результаты экзамена по теории вероятностей различаются для двух групп (μ1 ≠ μ2)

In [99]:
```
#Создадим выбороки для студентов с гласной и согласной начальной буквой
vowel_group = df[df['Фамилия'].str[0].str.upper().isin(['А', 'У', 'О', '
consonant_group = df[~df['Фамилия'].str[0].str.upper().isin(['А', 'У', '
```

а) тест Уэлча

In [100…
```
t_stat, p_value = stats.ttest_ind(vowel_group, consonant_group, equal_va
print('t-статистика:', t_stat)
print('p-значение:', p_value)
```

```
t-статистика: -1.0778460863139139
p-значение: 0.28521021475980995
```

Вывод: При уровне значимости α = 0.05, нет оснований отвергать нулевую гипотезу, так как полученное p-значение (0.285) больше уровня значимости. Следовательно, что мы не можем сделать вывод о статистически значимом различии в ожидаемых результатах экзамена между студентами с фамилиями, начинающимися на гласную и согласную букву.

б)Наивный бутстрэп

In [128…
```
np.random.seed(42)
n_iterations = 10000

diff_means = np.zeros(n_iterations)

for i in range(n_iterations):

    bootstrap_vowel = np.random.choice(vowel_group, size=len(vowel_group
    bootstrap_consonant = np.random.choice(consonant_group, size=len(con
    diff_means[i] = np.mean(bootstrap_vowel) - np.mean(bootstrap_consona

obs_diff_mean = np.mean(vowel_group) - np.mean(consonant_group)
p_value = np.sum(diff_means >= obs_diff_mean) / n_iterations

print('p-значение:', p_value)
```

```
p-значение: 0.5038
```

вывод: нет оснований отвергать нулевую гипотезу

в)бутстрэп t-статистики

In [127…

```
np.random.seed(42)
obs_t_stat = np.abs(np.mean(vowel_group) - np.mean(consonant_group)) / np
n_iterations = 10000

bootstrap_t_stats = np.zeros(n_iterations)

for i in range(n_iterations):

    bootstrap_vowel = np.random.choice(vowel_group, size=len(vowel_group
    bootstrap_consonant = np.random.choice(consonant_group, size=len(con

    bootstrap_t_stat = np.abs(np.mean(bootstrap_vowel) - np.mean(bootstra
    bootstrap_t_stats[i] = bootstrap_t_stat


p_value = (np.sum(bootstrap_t_stats >= obs_t_stat) + np.sum(bootstrap_t_

print('p-значение:', p_value)
```

```
p-значение: 0.5161
```

вывод: нет оснований отвергать нулевую гипотезу

г)перестановочный тест

In [126…

```
np.random.seed(42)
obs_stat = np.mean(vowel_group) - np.mean(consonant_group)

n_permutations = 10000

perm_stats = np.zeros(n_permutations)


for i in range(n_permutations):

    combined = np.concatenate([vowel_group, consonant_group])
    permuted = np.random.permutation(combined)

    perm_vowel_group = permuted[:len(vowel_group)]
    perm_consonant_group = permuted[len(vowel_group):]

    perm_stat = np.mean(perm_vowel_group) - np.mean(perm_consonant_group

    perm_stats[i] = perm_stat

p_value = (np.abs(perm_stats) >= np.abs(obs_stat)).mean()

print('p-значение:', p_value)
```

```
p-значение: 0.2609
```

Вывод: нет оснований отвергать нулевую гипотезу

Задание №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.

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

In [138…

```
def get_first_letter_type(last_name):
    vowels = ['а', 'е', 'ё', 'и', 'о', 'у', 'ы', 'э', 'ю', 'я']
    if last_name[0].lower() in vowels:
        return 'гласная'
    else:
        return 'согласная'

median_score = df['Оценка'].median()


# Функция для определения, набрал ли студент больше медианы
def is_above_median(score):
    if score > median_score:
        return 'больше медианы'
    else:
        return 'меньше или равно медиане'


df['Тип первой буквы'] = df['Фамилия'].apply(get_first_letter_type)
df['Медиана'] = df['Оценка'].apply(is_above_median)

# Создандим таблицу сопряжённости
contingency_table = pd.crosstab(df['Медиана'], df['Тип первой буквы'])
print(contingency_table)
```

```
Тип первой буквы          гласная  согласная
Медиана
больше медианы                 21        145
меньше или равно медиане       28        138
```

а)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

Пусть

-Количество "несогласных" студентов: НЕсог= 49

-Количество "согласных" студентов: сог = 283

-Количество "несогласных" студентов, которые хорошо написали экзамен: НЕсог_хор = 21

-Количество "согласных" студентов, которые хорошо написали экзамен: сог_хор = 145

тогда построим 95% асимптотический интервал для отношения шансов хорошо написать экзамен:

In [171…

```
НЕсог= 49
сог = 283
НЕсог_хор = 21
сог_хор = 145

# Вычислим отношения шансов
odds_ratio = (НЕсог_хор / (НЕсог - НЕсог_хор)) / (сог_хор / (сог - сог_х

# Построим асимптотический интервал для отношения шансов
se = np.sqrt(1 / НЕсог_хор + 1 / (НЕсог - НЕсог_хор) + 1 / сог_хор + 1 /
lower = np.exp(np.log(odds_ratio) - 1.96 * se)
upper = np.exp(np.log(odds_ratio) + 1.96 * se)
```

```
print("95% Асимптотический интервал для отношения шансов:")
print("Нижняя граница:",lower)
print("Верхняя граница",upper)
```

```
95% Асимптотический интервал для отношения шансов:
Нижняя граница: 0.3870902431823096
Верхняя граница 1.3162320763800788
```

вывод: 95% асимптотический интервал для отношения шансов хорошо написать экзамен: от 0.387 до 1.316

Проверка гипотезы о равенстве отношения шансов 1

НО:отношение шансов равно 1
Н1:отношение шансов не равно 1

In [179…

```
from scipy.stats import fisher_exact

oddsratio, p_value = fisher_exact(contingency_table)

print("Отношение шансов:", oddsratio)
print("P-значение:", p_value)
```

```
Отношение шансов: 0.7137931034482758
P-значение: 0.3533108749219954
```

вывод: нет оснований отвергать Н0

б)

Доля успеха в группе "Больше медианы" и "Согласная буква": p1 = 145/ (145 + 138)
Доля успеха в группе "Меньше медианы" и "Согласная буква": p2 = 138 / (145 + 138)
Доля успеха в группе "Больше медианы" и "Гласная буква": p3 = 21 / (21 + 28)
Доля успеха в группе "Меньше медианы" и "Гласная буква": p4 = 28 / (21 + 28)

In [159…

```
A = 145
B = 138
C = 21
D = 28
```

In [160…

```
def calculate_interval(A, B, C, D):
    p1 = A / (A + B)
    p2 = B / (A + B)
    p3 = C / (C + D)
    p4 = D / (C + D)
    p = (p1 / p2) / (p3 / p4)

    se = math.sqrt((1 / (A + B)) + (1 / (C + D)) + (1 / (A + C)) + (1 /

    lower = math.exp(math.log(p) - 1.96 * se)
    upper = math.exp(math.log(p) + 1.96 * se)

    return (lower, upper)

interval = calculate_interval(A, B, C, D)
print("95% асимптотический интервал:", interval)
```

```
95% асимптотический интервал: (0.9659244265063416, 2.0319459718181965)
```

Для проверки гипотезы о том, что отношение вероятностей равно 1 используем Хи-квадрат тест.

H0: Отношение вероятностей равно 1.
H1: Отношение вероятностей не равно 1.

In [158…

```
from scipy.stats import chi2_contingency

def test_hypothesis(A, B, C, D):
    observed = [[A, B], [C, D]]
    chi2, p_value, _, _ = chi2_contingency(observed)
    return p_value

p_value = test_hypothesis(A, B, C, D)
print("P-значение:", p_value)
```

```
P-значение: 0.35320687726026134
```

вывод: нет оснований отвергать H0

в) Построим 95% интервал для отношения шансов хорошо написать экзамен с помощью наивного бутстрэпа

In [178…

```
всего = 166
HEcor_хор = 21
cor_хор = 145
bootstrap_size = 1000

odds_ratios = np.zeros(bootstrap_size)


for i in range(bootstrap_size):
    HEcor_хор_ = np.random.binomial(всего, (HEcor_хор / всего))
    cor_хор_  = np.random.binomial(всего, (cor_хор /всего))

    # Рассчитываем отношение шансов для бутстрэп-выборки
    odds_ratio = (HEcor_хор_ / cor_хор_)
    odds_ratios[i] = odds_ratio

# Рассчитываем 95% интервал
lower = np.percentile(odds_ratios, 2.5)
upper = np.percentile(odds_ratios, 97.5)


# Вывод результатов
print("95% Интервал для отношения шансов:", (lower, upper))
```

```
95% Интервал для отношения шансов: (0.08842043574186431, 0.2013786549021448)
```

Гипотезы:
H0: Отношение шансов равно 1 (нет разницы между "несогласными" и "согласными" студентами)
H1: Отношение шансов не равно 1 (есть разница между "несогласными" и "согласными" студентами)
Для проверки гипотезы используем тест Фишера:

In [175…

```
from scipy.stats import fisher_exact
```

```
oddsratio, p_value = fisher_exact(contingency_table)

print("Отношение шансов:", oddsratio)
print("P-значение:", p_value)
```

```
Отношение шансов: 0.7137931034482758
P-значение: 0.3533108749219954
```

вывод: нет оснований отвергать H0

Задача №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) = βF_i , где Y_i — результат за экзамен по 30-балльной шкале, F_i — количество букв в фамилии.

а) Для оценки параметра β методом моментов в E(Y_i) = βFi, необходимо использовать выборочное среднее значение результата экзамена (Y_i) и выборочное среднее значение количества букв в фамилиях (F_i) и установить их равенство.

$\beta = E(Y_i) / E(F_i)$

In [162…

```
def count_letters(word):
    return len(word)
df['F_i'] = df.iloc[:, 0].apply(count_letters)
df
```

Out[162…

	Фамилия	ИО	Оценка	Тип первой буквы	Медиана	Начинается на гласную	Превысила медиану	F
0	Репенкова	Полина Александровна	16.0	согласная	меньше или равно медиане	False	False	
1	Ролдугина	Софья Александровна	0.0	согласная	меньше или равно медиане	False	False	
2	Сафина	Алия Линаровна	19.0	согласная	больше медианы	False	True	
3	Сидоров	Иван Максимович	26.0	согласная	больше медианы	False	True	
4	Солоухин	Иван Владимирович	21.0	согласная	больше медианы	False	True	
...	...	...	...	...	...	...	...	
327	Сенников	Александр -	19.0	согласная	больше медианы	False	True	
328	Ся	Юйцянь -	0.0	согласная	меньше или равно медиане	False	False	

329	Сятова	Альфия -	0.0	согласная	меньше или равно медиане	False	False
330	Темиркулов	Дастан Автандилович	0.0	согласная	меньше или равно медиане	False	False
331	Эшмеев	Павел Владиславович	0.0	гласная	меньше или равно медиане	True	False

332 rows × 8 columns

In [163…
```
df["Оценка"] = pd.to_numeric(df["Оценка"])
mean = df["Оценка"].mean()
print("Выборочное среднее результатов экзамена:", mean)
```

```
Выборочное среднее результатов экзамена: 16.156626506024097
```

In [164…
```
df["F_i"] = pd.to_numeric(df["F_i"])
mean = df["F_i"].mean()
print("Выборочное среднее количества букв в фамилиях:", mean)
```

```
Выборочное среднее количества букв в фамилиях: 7.86144578313253
```

In [165…
```
# считаем Оценка параметра β:
E_Yi = 16.205438066465256
E_Fi = 7.86144578313253

β = E_Yi / E_Fi

print("β:", β)
```

```
β: 2.0613813938951973
```

> **Рассчитайте выборочную корреляцию**

Для расчета выборочной корреляции между результатами экзамена и количеством букв в фамилиях, нужно вычислить ковариацию и выборочные стандартные отклонения обоих переменных.

In [166…
```
#Посчитаем выборочное стандартное отклонение
std_Yi = df["Оценка"].std()
std_Fi = df["F_i"].std()

print("Выборочное стандартное отклонение для оценок:", std_Yi)
print("Выборочное стандартное отклонение для количества букв в фамилиях:
```

```
Выборочное стандартное отклонение для оценок: 7.973872633702508
Выборочное стандартное отклонение для количества букв в фамилиях: 1.79763
73433481208
```

In [167…
```
#Посчитаем ковариацию между Yi и Fi:
```

```
covariance = df['Оценка'].cov(df['F_i'])
print("Ковариация между Yi и Fi:", covariance)
```

```
Ковариация между Yi и Fi: 0.45076984675863585
```

In [168…

```
#Посчитаем выборочную корреляцию между Yi и Fi:
Cov = 0.36072507552869976
std_Yi = 7.936114271591524
std_Fi = 1.7976373433481208
r = Cov / (std_Yi * std_Fi)

print("Выборочная корреляция:", r)
```

```
Выборочная корреляция: 0.025285196852942945
```

H0: корреляция равна нулю H1: корреляция неравнв нулю

In [170…

```
x = df['Оценка']
y = df['F_i']

# Фактическая корреляция между двумя столбцами
observed_corr = np.corrcoef(x, y)[0, 1]

# Число перестановок для проведения теста
num_permutations = 10000

# Создание массива для хранения переставленных корреляций
permutation_corrs = np.zeros(num_permutations)

# Выполнение перестановочного теста
for i in range(num_permutations):
    # Случайная перестановка значений столбца y
    permuted_y = np.random.permutation(y)

    # Вычисление корреляции между столбцом x и переставленным столбцом y
    corr = np.corrcoef(x, permuted_y)[0, 1]

    # Сохранение корреляции в массив
    permutation_corrs[i] = corr

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

# Вывод результатов
print("Фактическая корреляция:", observed_corr)
print("p-значение:", p_value)
```

```
Фактическая корреляция: 0.03144730851982361
p-значение: 0.5654
```

вывод: P-значение = 0.56 больше, чем уровень значимости 0.05. Это означает, что у нас нет оснований отвергать нулевую гипотезу о том, что корреляция между результатами экзамена и количеством букв в фамилии равна нулю. Мы не можем сделать вывод о наличии значимой связи между этими двумя переменными.