Задача №3

Пункт а)

Классический асимптотический нормальный интервал:

In [5]:

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

np.random.seed(13)
sample_size = 20
num_simulations = 10000
confidence_level = 0.95

samples = np.random.exponential(scale=1, size=(num_simulations, sample_size

means = np.mean(samples, axis=1)
std_devs = np.std(samples, axis=1, ddof=1)

z_critical = norm.ppf((1 + confidence_level) / 2)

count_covering = np.count_nonzero(np.logical_and((means - z_critical * (std

probability_covering = count_covering / num_simulations

print("Вероятность накрытия классическим асимптотическим нормальным интерв
```

```
Вероятность накрытия классическим асимптотическим нормальным интервалом:: 0
.9031
```

С помощью наивного бутстрэпа:

In [2]:

```
import numpy as np

np.random.seed(13)
sample_size = 20
num_simulations = 10000
confidence_level = 0.95

samples = np.random.exponential(scale=1, size=(num_simulations, sample_size

count_covering = 0

for i in range(num_simulations):
    bootstrap_samples = np.random.choice(samples[i], size=(num_simulations
    bootstrap_mean = np.mean(bootstrap_samples, axis=1)
    left = np.quantile(bootstrap_mean, (1 - confidence_level) / 2)
    right = np.quantile(bootstrap_mean, (1 + confidence_level) / 2)

    if left <= 1 and right >= 1:
        count_covering += 1

probability_covering = count_covering / num_simulations

print("Вероятность накрытия для наивного бутстрэпа: {:.4f}".format(probabi
```

```
Вероятность накрытия для наивного бутстрэпа: 0.9029
```

С помощью бутстрэпа t-статистики

In [3]:

```
import numpy as np

np.random.seed(13)
sample_size = 20
num_simulations = 10000
confidence_level = 0.95

samples = np.random.exponential(scale=1, size=(num_simulations, sample_size

count_covering = 0

for i in range(num_simulations):
    bootstrap_samples = np.random.choice(samples[i], size=(num_simulations
    bootstrap_mean = (np.mean(bootstrap_samples, axis=1) - np.mean(samples

    left = np.quantile(bootstrap_mean, (1 - confidence_level) / 2)
    right = np.quantile(bootstrap_mean, (1 + confidence_level) / 2)
    ql = np.mean(samples[i]) - right * (np.std(samples[i], ddof=1)/np.sqrt
    qr = np.mean(samples[i]) - left * (np.std(samples[i], ddof=1)/np.sqrt(s

    if ql <= 1 and qr >= 1:
        count_covering += 1

probability_covering = count_covering / num_simulations

print("Вероятность накрытия для наивного бутстрэпа: {:.4f}".format(probabi
```

```
Вероятность накрытия для наивного бутстрэпа: 0.9453
```

Пункт б)

Классический асимптотический нормальный интервал:

In [4]:

```
np.random.seed(13)
sample_size = 20
num_simulations = 10000
confidence_level = 0.95
degrees_of_freedom = 3

samples = np.random.standard_t(df=degrees_of_freedom, size=(num_simulations

means = np.mean(samples, axis=1)
std_devs = np.std(samples, axis=1, ddof=1)

z_critical = norm.ppf((1 + confidence_level) / 2)

count_covering = np.count_nonzero(np.logical_and((means - z_critical * (std

probability_covering = count_covering / num_simulations

print("Вероятность накрытия классическим асимптотическим нормальным интерв
```

```
Вероятность накрытия классическим асимптотическим нормальным интервалом:: 0
.9450
```

С помощью наивного бутстрэпа:

In [5]:

```
import numpy as np

np.random.seed(13)
sample_size = 20
num_simulations = 10000
confidence_level = 0.95
degrees_of_freedom = 3

samples = np.random.standard_t(df=degrees_of_freedom, size=(num_simulations

count_covering = 0

for i in range(num_simulations):
    bootstrap_samples = np.random.choice(samples[i], size=(num_simulations,
    bootstrap_mean = np.mean(bootstrap_samples, axis=1)
    left = np.quantile(bootstrap_mean, (1 - confidence_level) / 2)
    right = np.quantile(bootstrap_mean, (1 + confidence_level) / 2)

    if left <= 0 and right >= 0:
        count_covering += 1

probability_covering = count_covering / num_simulations

print("Вероятность накрытия для наивного бутстрэпа: {:.4f}".format(probabi
```

```
Вероятность накрытия для наивного бутстрэпа: 0.9215
```

С помощью бутстрэпа t-статистики

In [6]:

```
import numpy as np

np.random.seed(13)
sample_size = 20
num_simulations = 10000
confidence_level = 0.95
degrees_of_freedom = 3

samples = np.random.standard_t(df=degrees_of_freedom, size=(num_simulations

count_covering = 0

for i in range(num_simulations):
    bootstrap_samples = np.random.choice(samples[i], size=(num_simulations,
    bootstrap_mean = (np.mean(bootstrap_samples, axis=1) - np.mean(samples[

    left = np.quantile(bootstrap_mean, (1 - confidence_level) / 2)
    right = np.quantile(bootstrap_mean, (1 + confidence_level) / 2)
    ql = np.mean(samples[i]) - right * (np.std(samples[i], ddof=1)/np.sqrt(
    qr = np.mean(samples[i]) - left * (np.std(samples[i], ddof=1)/np.sqrt(s

    if ql <= 0 and qr >= 0:
        count_covering += 1

probability_covering = count_covering / num_simulations

print("Вероятность накрытия для наивного бутстрэпа: {:.4f}".format(probabi
```

```
Вероятность накрытия для наивного бутстрэпа: 0.9263
```

Пункт в)

Вывод: Для экспоненциального распределения лучший результат показал бутстрэп t-статистики: накрытие = 94.5%. Для распределения Стьюдента с 3 степенями свободы лучший результат показал классический асимптотический способ: накрытие = 94.5%

Задача №4

Пункт а)

In [1]:

```
import pandas as pd
import scipy.stats as stats

data = pd.read_csv('22-23_hse_probability - Exam.csv', skiprows=range(1, 6

# Разделение данных на две группы по начальной букве фамилии
vowels = ['А', 'Е', 'Ё', 'И', 'О', 'У', 'Ы', 'Э', 'Ю', 'Я']
consonants = ['Б', 'В', 'Г', 'Д', 'Ж', 'З', 'Й', 'К', 'Л', 'М', 'Н', 'П',

group_vowel = data[data['Last name'].str[0].str.upper().isin(vowels)]['Unna
group_consonant = data[data['Last name'].str[0].str.upper().isin(consonants

# Проверка гипотезы о равенстве ожидаемых результатов с помощью теста Уэлч
statistic, p_value = stats.ttest_ind(group_vowel, group_consonant, equal_va

print("Статистика теста:", statistic)
print("p-значение:", p_value)

alpha = 0.05  # Уровень значимости
if p_value < alpha:
    print("Отвергаем нулевую гипотезу")
else:
    print("Не отвергаем нулевую гипотезу")
```

```
Статистика теста: -0.8519661870595602
p-значение: 0.3974027153843839
Не отвергаем нулевую гипотезу
```

Пункт б)

In [2]:

```
import numpy as np

np.random.seed(13)

n_vowel = len(group_vowel)
n_consonant = len(group_consonant)

# Вычисление разности средних оценок
observed_diff = np.mean(group_vowel) - np.mean(group_consonant)

num_bootstraps = 10000

# Инициализация массива для сохранения разностей средних оценок на каждой
bootstrap_diffs = np.zeros(num_bootstraps)

# Наивный бутстрэп
for i in range(num_bootstraps):
    # Перемешиваем оценки двух групп
    bootstrap_vowel = np.random.choice(group_vowel, size=n_vowel, replace=
    bootstrap_consonant = np.random.choice(group_consonant, size=n_consona

    # Вычисляем разность средних оценок для текущей итерации бутстрэпа
    bootstrap_diffs[i] = np.mean(bootstrap_vowel) - np.mean(bootstrap_cons

alpha = 0.05
lower = np.percentile(bootstrap_diffs, 100 * alpha / 2)
upper = np.percentile(bootstrap_diffs, 100 * (1 - alpha / 2))

# Проверяем, попадает ли 1 в доверительный интервал
is_one_in_interval = lower <= 1 <= upper

print("Разность средних оценок: {:.4f}".format(observed_diff))
print("Доверительный интервал ({:.2%}): [{:.4f}, {:.4f}]".format(1 - alpha
print("Единица{}попадает в доверительный интервал".format(" " if is_one_in
if is_one_in_interval:
    print(" => Не отвергаем нулевую гипотезу")
else:
    print(" => Отвергаем нулевую гипотезу")
```

```
Разность средних оценок: -1.0782
Доверительный интервал (95.00%): [-3.6000, 1.3552]
Единица попадает в доверительный интервал
 => Не отвергаем нулевую гипотезу
```

Пункт в)

In [3]:

```
import numpy as np

np.random.seed(13)

bootstrap_diffs = np.zeros(num_bootstraps)

# Бутстрэп t-статистики
for i in range(num_bootstraps):
    # Перемешиваем оценки двух групп
    bootstrap_vowel = np.random.choice(group_vowel, size=n_vowel, replace=T
    bootstrap_consonant = np.random.choice(group_consonant, size=n_consonar

    # Вычисляем средние оценки для текущей итерации бутстрэпа
    bootstrap_mean_vowel = np.mean(bootstrap_vowel)
    bootstrap_mean_consonant = np.mean(bootstrap_consonant)

    # Вычисляем стандартные ошибки для текущей итерации бутстрэпа
    bootstrap_std_vowel = np.std(bootstrap_vowel, ddof=1) / np.sqrt(n_vowel
    bootstrap_std_consonant = np.std(bootstrap_consonant, ddof=1) / np.sqrt

    # Вычисляем t-статистику для текущей итерации бутстрэпа
    bootstrap_t_stat = (bootstrap_mean_vowel - bootstrap_mean_consonant) /

    # Сохраняем разность средних оценок на текущей итерации бутстрэпа
    bootstrap_diffs[i] = bootstrap_t_stat

alpha = 0.05
lower_critical = np.percentile(bootstrap_diffs, alpha/2 * 100)
upper_critical = np.percentile(bootstrap_diffs, (1 - alpha/2) * 100)

is_one_in_critical = lower_critical <= 1 <= upper_critical

print("Разность средних оценок: {:.4f}".format(observed_diff))
print("Критическая область ({:.2%}): [{:.4f}, {:.4f}]".format(1 - alpha, lo
print("Единица{}попадает в критическую область".format(" " if is_one_in_cri
if is_one_in_critical:
    print(" => Не отвергаем нулевую гипотезу")
else:
    print(" => Отвергаем нулевую гипотезу")
```

```
Разность средних оценок: -1.0782
Критическая область (95.00%): [-2.7720, 1.1654]
Единица попадает в критическую область
 => Не отвергаем нулевую гипотезу
```

Пункт г)

In [4]:

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

combined_group = np.concatenate((group_vowel, group_consonant))

num_permutations = 10000

permutation_diffs = np.zeros(num_permutations)

# Перестановочный тест
for i in range(num_permutations):
    # Перестановка меток групп
    np.random.shuffle(combined_group)

    # Вычисление разности средних оценок для текущей перестановки
    perm_group_vowel = combined_group[:n_vowel]
    perm_group_consonant = combined_group[n_vowel:]
    permutation_diffs[i] = np.mean(perm_group_vowel) - np.mean(perm_group_c

p_value = (np.abs(permutation_diffs) >= np.abs(observed_diff)).mean()

print("Разность средних оценок: {:.4f}".format(observed_diff))
print("p-значение: {:.4f}".format(p_value))

alpha = 0.05
if p_value > alpha:
    print("Не отвергаем нулевую гипотезу")
else:
    print("Отвергаем нулевую гипотезу")
```

```
Разность средних оценок: -1.0782
p-значение: 0.3876
Не отвергаем нулевую гипотезу
```

Задание №7

Решал 4 задачу из дз с помощью чата gpt

Мой запрос: Представь, что у тебя есть csv файл с оценками по экзамену и реши мне эту задачу: 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. Используй тест Уэлча.

In [ ]:
```
import pandas as pd
import scipy.stats as stats

data = pd.read_csv('grades.csv')

group_vowel = data[data['Surname'].str[0].isin(['A', 'E', 'I', 'O', 'U'])]
group_consonant = data[~data['Surname'].str[0].isin(['A', 'E', 'I', 'O', 'U

statistic, p_value = stats.ttest_ind(group_vowel, group_consonant, equal_va

print("Статистика теста:", statistic)
print("p-значение:", p_value)

alpha = 0.05
if p_value < alpha:
    print("Отвергаем нулевую гипотезу. Есть статистически значимые различи
else:
    print("Не отвергаем нулевую гипотезу. Нет статистически значимых разли
```

Мой запрос: в файле фамилии будут на русском языке

In [ ]:
```
import pandas as pd
import scipy.stats as stats

data = pd.read_csv('grades.csv', encoding='utf-8')

vowels = ['А', 'Е', 'Ё', 'И', 'О', 'У', 'Ы', 'Э', 'Ю', 'Я']
consonants = ['Б', 'В', 'Г', 'Д', 'Ж', 'З', 'Й', 'К', 'Л', 'М', 'Н', 'П',

group_vowel = data[data['Фамилия'].str[0].upper().isin(vowels)]['Оценка']
group_consonant = data[data['Фамилия'].str[0].upper().isin(consonants)]['Оц

statistic, p_value = stats.ttest_ind(group_vowel, group_consonant, equal_va

print("Статистика теста:", statistic)
print("p-значение:", p_value)

alpha = 0.05
if p_value < alpha:
    print("Отвергаем нулевую гипотезу. Есть статистически значимые различи
else:
    print("Не отвергаем нулевую гипотезу. Нет статистически значимых разли
```

Мой запрос: поменяй название файла на '22-23_hse_probability - Exam.csv';
поменяй 'Score' на 'Unnamed: 72' поменяй "Фамилия" на "Last name"

In [ ]:
```
data = pd.read_csv( '22-23_hse_probability - Exam.csv')
group_vowel = data[data['Unnamed: 72'].str[0].upper().isin(vowels)]['Unname
group_consonant = data[data['Unnamed: 72'].str[0].upper().isin(consonants)
```

Мой запрос: Ошибка: AttributeError: 'Series' object has no attribute 'upper'

In [ ]:
```
group_vowel = data[data['Unnamed: 72'].str[0].str.upper().isin(vowels)]['Un
group_consonant = data[data['Unnamed: 72'].str[0].str.upper().isin(consonan
```

In [11]:
```
import pandas as pd
import scipy.stats as stats

data = pd.read_csv('22-23_hse_probability - Exam.csv')

vowels = ['А', 'Е', 'Ё', 'И', 'О', 'У', 'Ы', 'Э', 'Ю', 'Я']
consonants = ['Б', 'В', 'Г', 'Д', 'Ж', 'З', 'Й', 'К', 'Л', 'М', 'Н', 'П',

group_vowel = data[data['Last name'].str[0].str.upper().isin(vowels)]['Unna
group_consonant = data[data['Last name'].str[0].str.upper().isin(consonants

statistic, p_value = stats.ttest_ind(group_vowel, group_consonant, equal_va

print("Статистика теста:", statistic)
print("p-значение:", p_value)

alpha = 0.05  # Уровень значимости
if p_value < alpha:
    print("Отвергаем нулевую гипотезу")
else:
    print("Не отвергаем нулевую гипотезу")
```

```
Статистика теста: -0.8519661870595602
p-значение: 0.3974027153843839
Не отвергаем нулевую гипотезу
```

Ещё один вариант: https://chat.openai.com/share/ead0523f-40ca-42a1-85ed-90fae6b1cd03

Задание №8

1. http://mathprofi.ru/statisticheskie_ocenki_parametrov_generalnoy_sovokupnosti.html - помогли понять базовые вещи про ДИ. На самом деле, этот сайт меня постоянно выручает, частенько на него захожу, чтобы разобраться с теорией на простых примерах
2. На stepik пару курсов смотрел, но очень мало, поэтому не могу порекомендовать конкретныйц курс
3. https://www.youtube.com/@user-bg8cd4fn7d/playlists

Задание №1

Пункт а)

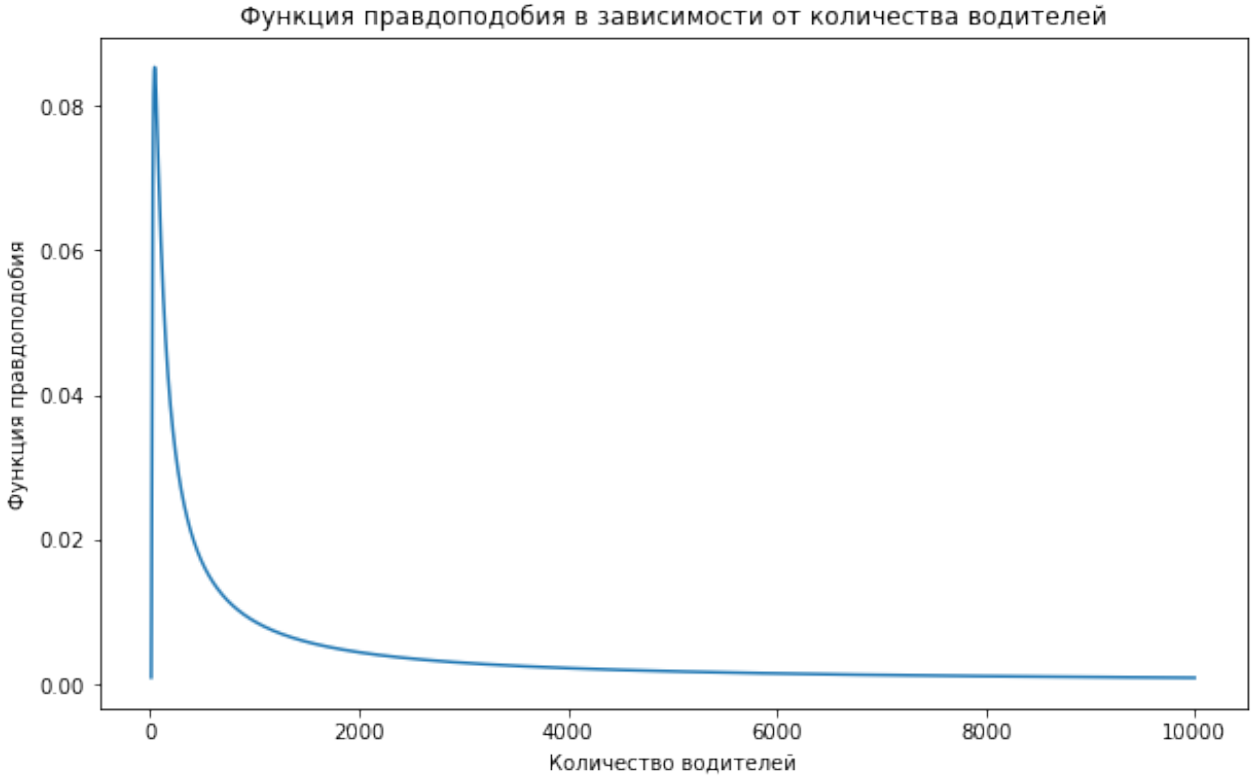
In [146…

```
max_ML = -1
arg_max_ML = -1
for n in range(9, 10001):
    x = n
    ML = 9 / x
    for i in range(1, 9):
        ML *= (x - i) / x
    if ML > max_ML:
        max_ML = ML
        arg_max_ML = n

print(arg_max_ML)
```

```
42
```

In [58]:

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

ns = np.arange(9, 10000)
likelihoods = [calculate_likelihood(n) for n in ns]

plt.figure(figsize=(10, 6))
plt.plot(ns, likelihoods)
plt.xlabel('Количество водителей')
plt.ylabel('Функция правдоподобия')
plt.title('Функция правдоподобия в зависимости от количества водителей')
plt.show()
```



Пункт б)

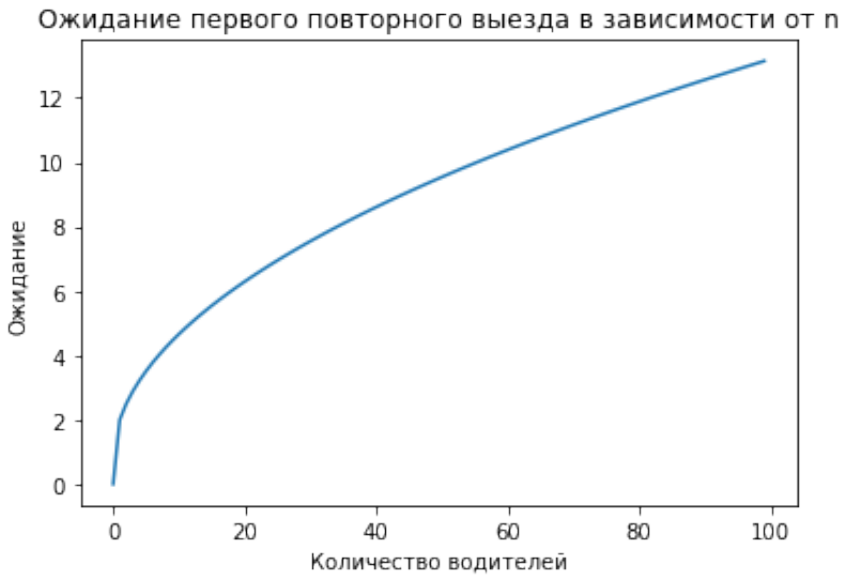
In [150…

```
n = 100  # Пример значения n
obs = 0

for i in range(2, n + 2):
    prod = 1
    for j in range(i - 1):
        prod *= (n - j) / n
    obs += i * (i - 1) * prod / n

print(obs)
```

```
13.209960630215972
```

In [155…

```
ns = np.arange(100)
obses = np.array([])

for n in ns:
    obs = 0
    for i in range(2, n + 2):
        prod = 1
        for j in range(i - 1):
            prod *= (n - j) / n
        obs += i * (i - 1) * prod / n
    obses = np.append(obses, obs)

plt.plot(ns, obses)
plt.xlabel('Количество водителей')
plt.ylabel('Ожидание')
plt.title('Ожидание первого повторного выезда в зависимости от n')
plt.show()
```



In [154…

```
mse = np.array([])
for i in range(0,100):
    mse = np.append(mse, (obses[i] - 10) ** 2)
ns[mse.argmin()]
```

Out[154… 55

Пункт в)

In [56]:

```
n_simulations = 10000
n_taxists = 100

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

for i in range(n_simulations):
    # Создание списка таксистов
    taxists = list(range(n_taxists))
    # Счетчик вызовов
    count = 0
    while True:
        # Выбор случайного таксиста
        selected_taxist = np.random.choice(taxists)
        # Удаление таксиста из списка
        taxists.remove(selected_taxist)
        count += 1
        # Если таксист уже был выбран ранее, сохраняем количество вызовов
        if selected_taxist in taxists:
            moment_estimates[i] = count
            mle_estimates[i] = count
            break

moment_mean = np.mean(moment_estimates)

mle_mean = np.mean(mle_estimates)

plt.hist(moment_estimates, bins=30, alpha=0.5, label='Метод моментов')
plt.hist(mle_estimates, bins=30, alpha=0.5, label='Метод максимального пра
plt.legend(loc='upper right')
plt.xlabel('Количество вызовов до повтора')
plt.ylabel('Частота')
plt.title('Гистограммы оценок методов')
plt.show()

moment_bias = moment_mean - n_taxists
moment_variance = np.var(moment_estimates)
moment_mse = moment_bias**2 + moment_variance

mle_bias = mle_mean - n_taxists
mle_variance = np.var(mle_estimates)
mle_mse = mle_bias**2 + mle_variance

print('Метод моментов:')
print('Смещение:', moment_bias)
print('Дисперсия:', moment_variance)
print('Среднеквадратичная ошибка:', moment_mse)

print('\nМетод максимального правдоподобия:')
print('Смещение:', mle_bias)
print('Дисперсия:', mle_variance)
print('Среднеквадратичная ошибка:', mle_mse)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/var/folders/09/_8mjdc0s6f3gncfnf5n7sp6w0000gn/T/ipykernel_79905/573021634.
py in <module>
     14     while True:
     15         # Выбор случайного таксиста
---> 16         selected_taxist = np.random.choice(taxists)
     17         # Удаление таксиста из списка
     18         taxists.remove(selected_taxist)

mtrand.pyx in numpy.random.mtrand.RandomState.choice()

ValueError: 'a' cannot be empty unless no samples are taken
```

Задача №2

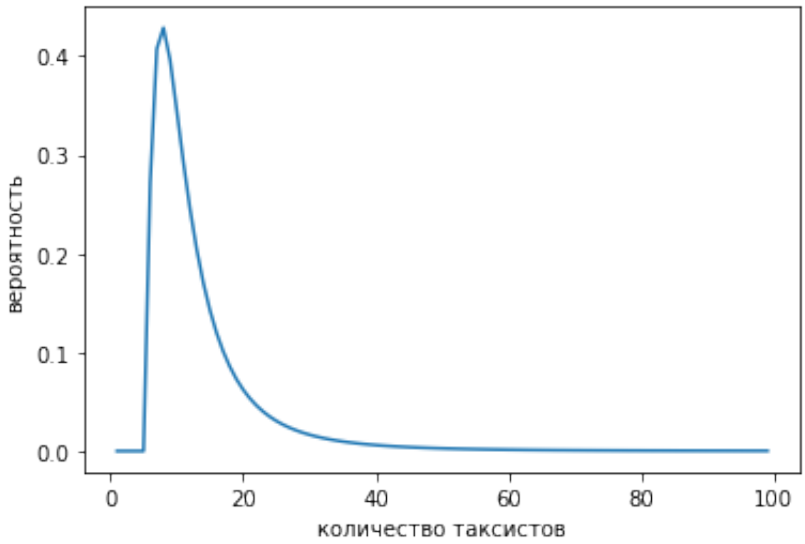
Пункт а)

In [40]:

```
import itertools
def prob(n, days, unique):
    likelihood = 1
    for i in range(1, unique):
        likelihood *= ((n-i)/n)
    combinations = itertools.combinations_with_replacement(np.arange(1, uni
    count = 0
    for combination in combinations:
        product = 1
        for i in range(days - unique):
            product *= combination[i]
        count += product
    likelihood *= (count / (n ** (days - unique)))
    return likelihood
```

In [55]:

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

plt.plot(n, L_2)
plt.xlabel('количество таксистов')
plt.ylabel('вероятность')
plt.show()
```



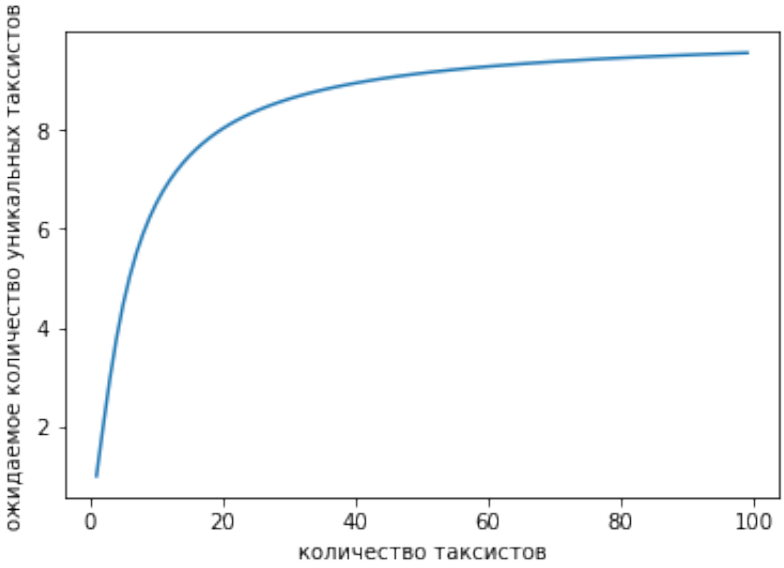
In [45]:
```
np.argmax(L_2) + 1
```

Out[45]: 8

Пункт б)

In [48]:
```
def alt(n, day):
    a = 0
    for unique_names in range(day+1):
        P = prob(n, day, unique_names)
        a += P * unique_names
    return a
```

In [53]:
```
n = np.arange(1, 100)
days = 10
ev_vec = np.vectorize(alt)
E_2 = ev_vec(n, days)

plt.plot(n, E_2)
plt.xlabel('количество таксистов')
plt.ylabel('ожидаемое количество уникальных таксистов')
plt.show()
```



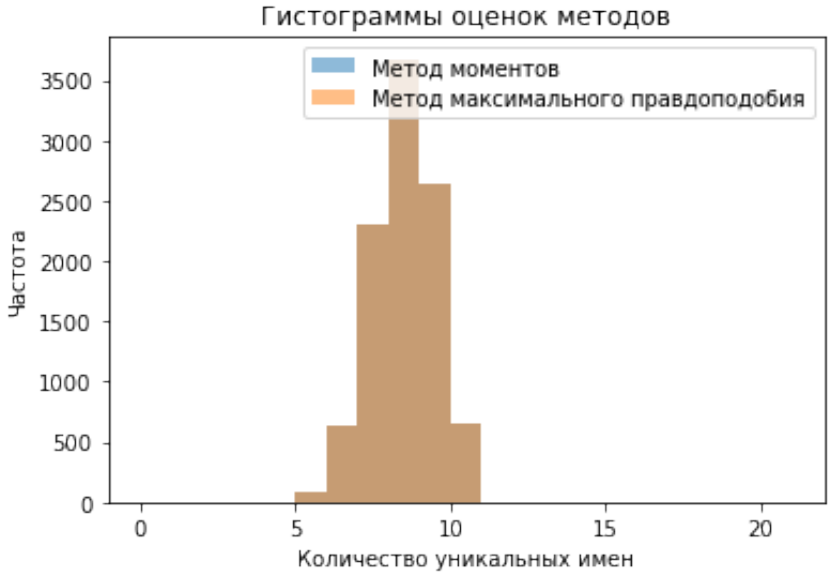
In [54]:
```
np.argmin(np.abs(E_2 - 6)) + 1
```

Out[54]: 8

Пункт в)

In [57]:

```
n_simulations = 10000  # Количество симуляций
n_names = 20  # Количество разных имен среди таксистов
n_orders = 10  # Количество заказов такси

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

for i in range(n_simulations):
    # Генерация случайного списка имен таксистов
    names = np.random.choice(range(n_names), size=n_orders, replace=True)
    # Подсчет уникальных имен
    unique_names = len(set(names))
    # Сохранение оценок
    moment_estimates[i] = unique_names
    mle_estimates[i] = unique_names

moment_mean = np.mean(moment_estimates)

mle_mean = np.mean(mle_estimates)

plt.hist(moment_estimates, bins=range(n_names + 2), alpha=0.5, label='Метод
plt.hist(mle_estimates, bins=range(n_names + 2), alpha=0.5, label='Метод ма
plt.legend(loc='upper right')
plt.xlabel('Количество уникальных имен')
plt.ylabel('Частота')
plt.title('Гистограммы оценок методов')
plt.show()

moment_bias = moment_mean - n_names
moment_variance = np.var(moment_estimates)
moment_mse = moment_bias**2 + moment_variance

mle_bias = mle_mean - n_names
mle_variance = np.var(mle_estimates)
mle_mse = mle_bias**2 + mle_variance

print('Метод моментов:')
print('Смещение:', moment_bias)
print('Дисперсия:', moment_variance)
print('Среднеквадратичная ошибка:', moment_mse)

print('\nМетод максимального правдоподобия:')
print('Смещение:', mle_bias)
print('Дисперсия:', mle_variance)
print('Среднеквадратичная ошибка:', mle_mse)
```




```
Метод моментов:
Смещение: -11.9849
Дисперсия: 1.0882719900000004
Среднеквадратичная ошибка: 144.7261

Метод максимального правдоподобия:
Смещение: -11.9849
Дисперсия: 1.0882719900000004
Среднеквадратичная ошибка: 144.7261
```

Задача №5

Пункт а)

In [5]:

```
median = data['Unnamed: 72'].median()

contingency_table = pd.DataFrame({
    'More': [(group_vowel > median).sum(), (group_consonant > median).sum(
    'Less': [(group_vowel < median).sum(), (group_consonant < median).sum(
}, index=['Vowel', 'Cons'])

contingency_table
```

Out[5]:

	More	Less
Vowel	21	28
Cons	145	138

In [6]:
```
from scipy import stats as sts

vowel_kryshka  = (group_vowel > median).sum() / (group_vowel < median).sum
cons_kryshka = (group_consonant > median).sum() / (group_consonant < media
OR_kryshka = vowel_kryshka / cons_kryshka

se_OR_kryshka = np.sqrt(1 / (group_vowel > median).sum() + 1 / (group_vowe

l = OR_kryshka * np.exp(-1.96 * se_OR_kryshka)
r = OR_kryshka * np.exp(1.96 * se_OR_kryshka)

Z_obs = np.log(OR_kryshka) / np.sqrt(1 / (group_vowel > median).sum() + 1
p_value = sts.norm.cdf(Z_obs, loc=0, scale=1)
print(f'Интервал: [{l:.3f}, {r:.3f}]')
print(f'P_value = {p_value:.3f}')
```

```
Интервал: [0.387, 1.316]
P_value = 0.140
```

Пункт б)

In [7]:
```
vowel_kryshka_2 =  (group_vowel > median).sum() / ((group_vowel < median).s
cons_kryshka_2 = (group_consonant > median).sum()  / ((group_consonant > me
RR_kryshka = vowel_kryshka_2 / cons_kryshka_2

se_RR_kryshka = np.sqrt((group_vowel < median).sum()/((group_vowel > media
                        (group_consonant < median).sum()/((group_consonant > me

l = RR_kryshka * np.exp(-1.96 * se_RR_kryshka)
r = RR_kryshka * np.exp(1.96 * se_RR_kryshka)

Z_obs = np.log(RR_kryshka) / np.sqrt(1/(group_vowel > median).sum() + 1/(gr
p_value = sts.norm.cdf(Z_obs, loc=0, scale=1)

print(f'Интервал: [{l:.3f}, {r:.3f}]')
print(f'P_value = {p_value:.3f}')
```

```
Интервал: [0.594, 1.178]
P_value = 0.284
```

Пункт в)

In [8]:
```
vowel_boot = np.random.choice(group_vowel, size=(10000, len(group_vowel)))
cons_boot = np.random.choice(group_consonant, size=(10000, len(group_consor

vowel_more_med_boot = (vowel_boot > median).sum(axis=1)
vowel_less_med_boot = (vowel_boot < median).sum(axis=1)
cons_more_med_boot = (cons_boot > median).sum(axis=1)
cons_less_med_boot = (cons_boot < median).sum(axis=1)

odds_vowel_kryshka_boot =  vowel_more_med_boot / vowel_less_med_boot
odds_cons_kryshka_boot = cons_more_med_boot / cons_less_med_boot
OR_kryshka_boot = odds_vowel_kryshka_boot / odds_cons_kryshka_boot

alpha = 0.05
l = np.quantile(OR_kryshka_boot, alpha/2)
r = np.quantile(OR_kryshka_boot, 1 - alpha/2)

p_value = np.mean(OR_kryshka_boot >= OR_kryshka)
print(f'Интервал: [{l:.3f}, {r:.3f}]')
print(f'P_value = {p_value:.3f}')
```

```
Интервал: [0.375, 1.308]
P_value = 0.506
```

Задача №6

Пункт а)

In [9]:
```
data['Last name'] = data['Last name'].astype(str)  # Convert to string typ
data['len_name'] = data['Last name'].apply(lambda x: len(x) if pd.notnull(x
```

In [10]:
```
beta = data['Unnamed: 72'].mean() / data['len_name'].mean()
beta
```

Out[10]:
```
2.0613026819923372
```

In [11]:
```
corr_table = data[['Unnamed: 72', 'len_name']].corr() # выборочная корреля
corr_table
```

Out[11]:

	Unnamed: 72	len_name
Unnamed: 72	1.000000	0.025328
len_name	0.025328	1.000000

Пункт б)

In [13]:
```
np.random.seed(13)
p_permuted = []
for i in range(10000):
    y_permuted = np.random.permutation(data['Unnamed: 72'])
    p_new = np.corrcoef(y_permuted, data.len_name)[0,1]
    p_permuted.append(p_new)

p_permuted = np.array(p_permuted)
```

In [18]:
```
p_value = 2 * min(np.sum(p_permuted > corr_table.iloc[0,1])/10000, np.sum(
np.round(p_value, 2)
```

Out[18]: 0.65