

In [13]:

#ЗАДАЧА 3а

```
import pandas as pd
import numpy as np
import scipy.stats as sts
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
from collections import defaultdict
from scipy.stats import ttest_ind
import warnings
from statsmodels.stats.proportion import proportions_ztest
from scipy.stats import norm
np.random.seed(23)

intensity = 1.0 # Интенсивность распределения
sample_size = 20 # Размер выборки
num_samples = 10000 # Количество симуляций

# Заданная выборка (пример)
sample = np.random.exponential(scale=1/intensity, size=sample_size)

# Истинное математическое ожидание
true_mean = np.mean(sample)

# Вычисление выборочного среднего и выборочного стандартного отклонения
true_mean = np.mean(sample)
true_std = np.std(sample, ddof=1)
true_t_stat = true_mean / (true_std / np.sqrt(sample_size))

# Вычисление критического значения z для доверительного уровня 95%
z_critical = norm.ppf(0.975)

# Вычисление доверительного интервала
margin_of_error = z_critical * (true_std / np.sqrt(sample_size))

left_quantile = true_mean - margin_of_error
right_quantile = true_mean + margin_of_error

# Функция для наивного бутстрэпа
def naive_bootstrap(data):
    resampled_means = []
    for _ in range(num_simulations):
        resampled_data = np.random.choice(data, size=sample_size, replace=True)
        resampled_mean = np.mean(resampled_data)
        resampled_means.append(resampled_mean)
    return resampled_means

# Функция для бутстрэпа с использованием t-статистики
def bootstrap_t(data):
    resampled_means = []
    for _ in range(num_simulations):
        resampled_data = np.random.choice(data, size=sample_size, replace=True)
        resampled_mean = np.mean(resampled_data)
        resampled_std = np.std(resampled_data, ddof=1)
        t_stat = (resampled_mean - true_mean) / (resampled_std / np.sqrt(sample_size))
        resampled_means.append(t_stat)
    return resampled_means

# Выполнение бутстрэпа с использованием t-статистики
resampled_t_boot_means = bootstrap_t(sample)

# Выполнение наивного бутстрэпа
resampled_boot_means = naive_bootstrap(sample)

# Создание массива для хранения средних значений
prob_standart = np.zeros(num_samples)
```

```

prob_boot = np.zeros(num_samples)
prob_t_boot = np.zeros(num_samples)

# Генерация выборок и вычисление средних значений
for i in range(num_samples):
    sample = np.random.exponential(scale=1/intensity, size=sample_size)
    sample_mean = np.mean(sample)
    mean = np.mean(sample)
    std = np.std(sample, ddof=1) # Используем параметр ddof=1 для несмещенной оценки стандартн
    # Расчет t-статистики
    t_stat = (mean - true_mean) / (std / np.sqrt(sample_size))

    prob_standart[i] = ((sample_mean >= left_quantile) & (sample_mean <= right_quantile))

    prob_boot[i] = ((sample_mean >= np.percentile(resampled_boot_means, 2.5)) &
                    (sample_mean <= np.percentile(resampled_boot_means, 97.5)))
    prob_t_boot[i] = ((t_stat >= np.percentile(resampled_t_boot_means, 2.5)) &
                     (t_stat <= np.percentile(resampled_t_boot_means, 97.5)))

# Вывод результата
print("Вероятность накрытия истинного математического ожидания 95%-м интервалом:")
print(np.mean(prob_standart))

print("Вероятность накрытия истинного математического ожидания 95%-м наивным бутстрэп-интервалом")
print(np.mean(prob_boot))

print("Вероятность накрытия истинного математического ожидания 95%-м бутстрэп-интервалом с использо")
print(np.mean(prob_t_boot))

```

Вероятность накрытия истинного математического ожидания 95%-м интервалом:

0.9606

Вероятность накрытия истинного математического ожидания 95%-м наивным бутстрэп-интервалом:

0.9472

Вероятность накрытия истинного математического ожидания 95%-м бутстрэп-интервалом с использованием t-статистики:

0.9397

In [14]:

```

#ЗАДАЧА 36

import pandas as pd
import numpy as np
import scipy.stats as sts
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
from collections import defaultdict
from scipy.stats import ttest_ind
import warnings
from statsmodels.stats.proportion import proportions_ztest
from scipy.stats import norm
from scipy.stats import t
np.random.seed(123)

intensity = 1.0 # Интенсивность распределения
sample_size = 20 # Размер выборки
num_simulations = 10000 # Количество симуляций

df = 3 # Степени свободы

# Создание 20 наблюдений распределения Стьюдента
sample = t.rvs(df, size=sample_size)

# Истинное математическое ожидание
true_mean = np.mean(sample)

# Вычисление выборочного среднего и выборочного стандартного отклонения
true_mean = np.mean(sample)
true_std = np.std(sample, ddof=1)

```

```

true_t_stat = true_mean / (true_std / np.sqrt(sample_size))

# Расчет критического значения t
t_critical = t.ppf(0.975, df=sample_size-1)

# Расчет доверительного интервала
margin_error = t_critical * (true_std / np.sqrt(sample_size))
left_quantile = true_mean - margin_error
right_quantile = true_mean + margin_error

# Функция для наивного бутстрэпа
def naive_bootstrap(data):
    resampled_means = []
    for _ in range(num_simulations):
        resampled_data = np.random.choice(data, size=sample_size, replace=True)
        resampled_mean = np.mean(resampled_data)
        resampled_means.append(resampled_mean)
    return resampled_means

# Функция для бутстрэпа с использованием t-статистики
def bootstrap_t(data):
    resampled_means = []
    for _ in range(num_simulations):
        resampled_data = np.random.choice(data, size=sample_size, replace=True)
        resampled_mean = np.mean(resampled_data)
        resampled_std = np.std(resampled_data, ddof=1)
        t_stat = (resampled_mean - true_mean) / (resampled_std / np.sqrt(sample_size))
        resampled_means.append(t_stat)
    return resampled_means

# Выполнение бутстрэпа с использованием t-статистики
resampled_t_boot_means = bootstrap_t(sample)

# Выполнение наивного бутстрэпа
resampled_boot_means = naive_bootstrap(sample)

# Создание массива для хранения средних значений
prob_standart = np.zeros(num_samples)
prob_boot = np.zeros(num_samples)
prob_t_boot = np.zeros(num_samples)

# Генерация выборок и вычисление средних значений
for i in range(num_samples):
    sample = t.rvs(df, size=sample_size)
    sample_mean = np.mean(sample)
    std = np.std(sample, ddof=1) # Используем параметр ddof=1 для несмещенной оценки стандартн
    # Расчет t-статистики
    t_stat = (sample_mean - true_mean) / (std / np.sqrt(sample_size))

    prob_standart[i] = ((sample_mean >= left_quantile) & (sample_mean <= right_quantile))

    prob_boot[i] = ((sample_mean >= np.percentile(resampled_boot_means, 2.5)) &
                    (sample_mean <= np.percentile(resampled_boot_means, 97.5)))

    prob_t_boot[i] = ((t_stat >= np.percentile(resampled_t_boot_means, 2.5)) &
                     (t_stat <= np.percentile(resampled_t_boot_means, 97.5)))

# Вывод результата
print("Вероятность накрытия истинного математического ожидания 95%-м интервалом:")
print(np.mean(prob_standart))

print("Вероятность накрытия истинного математического ожидания 95%-м наивным бутстрэп-интервалом")
print(np.mean(prob_boot))

print("Вероятность накрытия истинного математического ожидания 95%-м бутстрэп-интервалом с исп")
print(np.mean(prob_t_boot))

```

Вероятность накрытия истинного математического ожидания 95%-м интервалом:

0.9587

Вероятность накрытия истинного математического ожидания 95%-м наивным бутстрэп-интервалом:

0.947

Вероятность накрытия истинного математического ожидания 95%-м бутстрэп-интервалом с использованием t-статистики:

0.9455

№3 В

Результаты всех трех методов довольно близки, на основании чего нельзя делать однозначных выводов о том какой способ лучше. Однако согласно результатам, можно сказать, что результаты оценки асимптотическим интервалом немного выше остальных.

Опять же, довольно странно говорить о понятии "лучше" глядя только на вероятности. Вероятность покрытия асимптотическим интервалом больше, потому что сам интервал больше. Если интервал -1000, 1000 имеет вероятность покрытия 1.0, то это не означает, что он делает хорошие оценки.

In [1]:

```
#ЗАДАЧА 4 (a, б)

import pandas as pd
import numpy as np
import scipy.stats as sts
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
from collections import defaultdict
from scipy.stats import ttest_ind

import warnings
warnings.filterwarnings("ignore")
sns.set_theme(style="whitegrid", palette="muted")
plt.rcParams['figure.figsize'] = (8, 4)
np.random.seed(123)

df = pd.read_csv('exam_results.csv', sep=';')

n_iterations = 100000 # Количество итераций бутстрэпа
alpha = 0.05 #уровень значимости

# Удаляем столбцы
df.drop(columns=['Unnamed: 71', 'Unnamed: 73', 'Unnamed: 74', 'Unnamed: 75'], inplace=True)

# Переименовываем столбец
df.rename(columns={'Unnamed: 72': 'results'}, inplace=True)

# Выводим обновленный DataFrame
df = df.dropna(subset=['Вариант'])
df_short = df[['num', 'Last name', 'results']]

def is_vowel(name):
    vowels = ['a', 'e', 'ё', 'и', 'о', 'у', 'ы', 'э', 'ю', 'я']
    return name.lower().startswith(tuple(vowels))

# Создание таблицы df_vowel с учениками, фамилии которых начинаются на гласную букву
df_vowel = df_short[df_short['Last name'].apply(is_vowel)]

# Создание таблицы df_consonant с учениками, фамилии которых начинаются на согласную букву
df_consonant = df_short[~df_short['Last name'].apply(is_vowel)]

vowel_results = df_vowel['results']
consonant_results = df_consonant['results']

# Это тест Уэлча, потому что "equal_var=False"
t_stat, p_value_a = ttest_ind(vowel_results, consonant_results, equal_var=False)
```

```
alpha = 0.05 # Уровень значимости
```

```
print("a) [5] Используйте тест Уэлча.")
print("p-значение:", p_value_a)
if p_value_a < alpha:
    print("Отвергаем нулевую гипотезу. Ожидаемые результаты за экзамен в обеих таблицах не равны")
else:
    print("Не отвергаем нулевую гипотезу. Ожидаемые результаты за экзамен в обеих таблицах равны")

def naive_bootstrap_diff_means(sample1, sample2, n_iterations):
    n1 = len(sample1)
    n2 = len(sample2)

    observed_diff = (np.mean(sample1) - np.mean(sample2))
    diffs = np.zeros(n_iterations)

    for i in range(n_iterations):
        bootstrap_sample1 = np.random.choice(sample1, size=n1, replace=True)
        bootstrap_sample2 = np.random.choice(sample2, size=n2, replace=True)

        bootstrap_diff = np.mean(bootstrap_sample1) - np.mean(bootstrap_sample2)
        diffs[i] = bootstrap_diff

    plt.hist(diffs, bins=100, color='blue', alpha=0.5)
    plt.axvline(x=observed_diff, color='red', linestyle='--', label='Observed Difference')
    plt.axvline(x=0, color='purple', linestyle='--', label='0')
    plt.xlabel('Difference')
    plt.ylabel('Frequency')
    plt.title('Histogram of Differences')
    plt.legend()
    plt.show()

    plt.hist(np.abs(diffs), bins=100, color='red')
    plt.axvline(x=np.abs(observed_diff), color='blue', linestyle='--', label='Observed Difference')
    plt.xlabel('Difference')
    plt.ylabel('Frequency')
    plt.title('Histogram of Differences')
    plt.legend()
    plt.show()

# p_value = np.mean(np.abs(diffs) >= np.abs(observed_diff))
p_value = 2*np.mean((diffs) >= 0)

return observed_diff, p_value
```

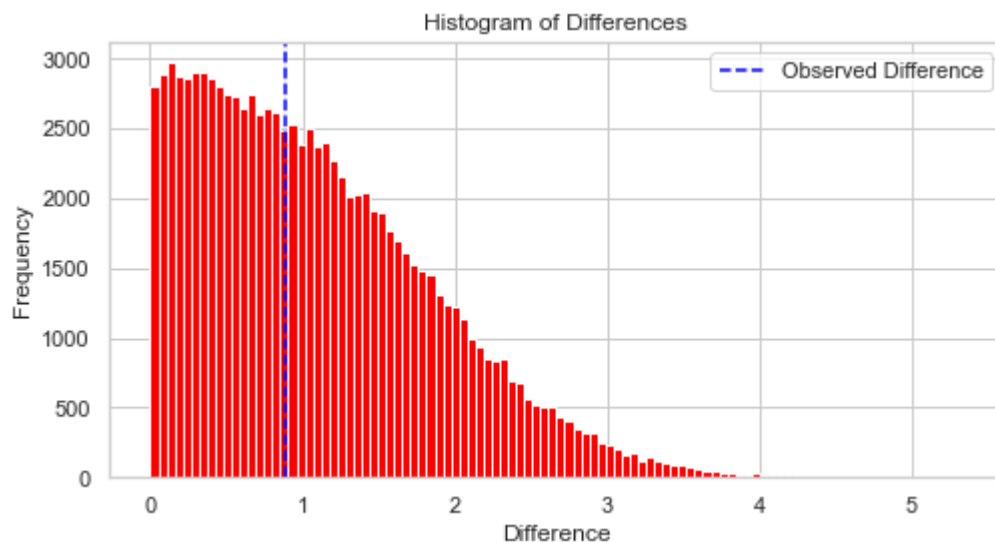
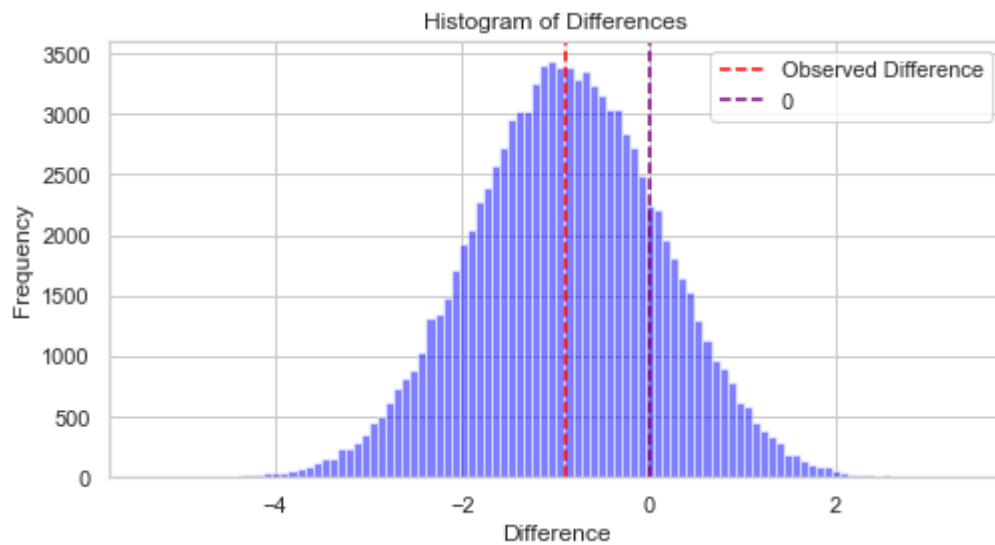
```
observed_diff, p_value_b = naive_bootstrap_diff_means(vowel_results, consonant_results, n_iterations)

print('b) [5] Используйте наивный бутстрэп')
print("наблюдаемая по изначальной выборке разность средних:", observed_diff)
print("p-значение:", p_value_b)
if p_value_b < alpha:
    print("Отвергаем нулевую гипотезу. Ожидаемые результаты за экзамен в обеих таблицах не равны")
else:
    print("Не отвергаем нулевую гипотезу. Ожидаемые результаты за экзамен в обеих таблицах равны")
```

a) [5] Используйте тест Уэлча.

p-значение: 0.3909901940797269

Не отвергаем нулевую гипотезу. Ожидаемые результаты за экзамен в обеих таблицах равны.



6) [5] Используйте наивный бутстрэп
наблюдаемая по изначальной выборке разность средних: -0.8857431749241655
р-значение: 0.3854
Не отвергаем нулевую гипотезу. Ожидаемые результаты за экзамен в обеих таблицах равны.

In []:

```
#ЗАДАЧА 4 ( $\theta, z$ )
```

```
import pandas as pd
import numpy as np
import scipy.stats as sts
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
from collections import defaultdict
from scipy.stats import ttest_ind
```

```
import warnings
warnings.filterwarnings("ignore")
sns.set_theme(style="whitegrid", palette="muted")
plt.rcParams['figure.figsize'] = (8, 4)
np.random.seed(123)
```

```
df = pd.read_csv('exam_results.csv', sep=';')
```

```
n_iterations = 100000 # Количество итераций бутстрэпа
alpha = 0.05 #уровень значимости
```

```
# Удаляем столбцы
```

```
df.drop(columns=['Unnamed: 71', 'Unnamed: 73', 'Unnamed: 74', 'Unnamed: 75'], inplace=True)
```

```
# Переименовываем столбец
```

```
df.rename(columns={'Unnamed: 72': 'results'}, inplace=True)
```

```

# Выводим обновленный DataFrame
df = df.dropna(subset=['Вариант'])
df_short = df[['num', 'Last name', 'results']]

def is_vowel(name):
    vowels = ['a', 'e', 'ё', 'и', 'o', 'y', 'ы', 'э', 'ю', 'я']
    return name.lower().startswith(tuple(vowels))

# Создание таблицы df_vowel с учениками, фамилии которых начинаются на гласную букву
df_vowel = df_short[df_short['Last name'].apply(is_vowel)]

# Создание таблицы df_consonant с учениками, фамилии которых начинаются на согласную букву
df_consonant = df_short[~df_short['Last name'].apply(is_vowel)]

vowel_results = df_vowel['results']
consonant_results = df_consonant['results']

# Создаём функцию бутстрэпа t-статистики

def bootstrap_t_test(sample1, sample2, n_iterations):
    n1 = len(sample1)
    n2 = len(sample2)

    observed_t_stat, _ = ttest_ind(sample1, sample2)
    t_stats = []

    for i in range(n_iterations):
        bootstrap_sample1 = np.random.choice(sample1, size=n1, replace=True)
        bootstrap_sample2 = np.random.choice(sample2, size=n2, replace=True)

        t_stat, _ = ttest_ind(bootstrap_sample1, bootstrap_sample2)
        t_stats.append(t_stat)

    p_value = np.mean(np.abs(t_stats) >= np.abs(observed_t_stat))

    plt.hist(t_stats, bins=100)
    plt.axvline(x=observed_t_stat, color='red', linestyle='--', label='Observed t-stat')
    plt.xlabel('t-stat')
    plt.ylabel('Frequency')
    plt.title('Histogram of t-stats')
    plt.legend()
    plt.show()

    plt.hist(np.abs(t_stats), bins=100, color='red')
    plt.axvline(x=np.abs(observed_t_stat), color='blue', linestyle='--', label='Observed t-stat')
    plt.xlabel('t-stat')
    plt.ylabel('Frequency')
    plt.title('Histogram of t-stats')
    plt.legend()
    plt.show()
    return observed_t_stat, p_value

# Вызов функции и проверка гипотезы
observed_t_stat, p_value_c = bootstrap_t_test(vowel_results, consonant_results, n_iterations)

print('в) [5] Используйте бутстрэп t-статистики')
print("наблюдаемая по изначальной выборке разность t-статистика:", observed_t_stat)
print("p-значение:", p_value_c)
if p_value_c < alpha:
    print("Отвергаем нулевую гипотезу. Ожидаемые результаты за экзамен в обеих таблицах не равны")
else:
    print("Не отвергаем нулевую гипотезу. Ожидаемые результаты за экзамен в обеих таблицах равны")

# Создаём функцию перестановочного теста
def permutation_test(sample1, sample2, n_iterations):

```

```

observed_diff = np.mean(sample1) - np.mean(sample2)
combined_results = np.concatenate((sample1, sample2))
n1 = len(sample1)
n2 = len(sample2)
n = n1 + n2
diffs = np.zeros(n_iterations)

for i in range(n_iterations):
    np.random.shuffle(combined_results)
    perm_sample1 = combined_results[:n1]
    perm_sample2 = combined_results[n1:]
    perm_diff = np.mean(perm_sample1) - np.mean(perm_sample2)
    diffs[i] = perm_diff

plt.hist(diffs, bins=50)
plt.axvline(x=observed_diff, color='red', linestyle='--', label='Observed Difference')
plt.xlabel('Difference')
plt.ylabel('Frequency')
plt.title('Histogram of Differences')
plt.legend()
plt.show()

plt.hist(np.abs(diffs), bins=50, color='red')
plt.axvline(x=np.abs(observed_diff), color='blue', linestyle='--', label='Observed Difference')
plt.xlabel('Difference')
plt.ylabel('Frequency')
plt.title('Histogram of Differences')
plt.legend()
plt.show()

p_value = np.mean(np.abs(diffs) >= np.abs(observed_diff))
return observed_diff, p_value

# Вызов функции и проверка гипотезы
observed_diff, p_value_d = permutation_test(vowel_results, consonant_results, n_iterations)

print('г) [5] Используйте перестановочный тест')
print("p-значение:", p_value_d)
if p_value_d < alpha:
    print("Отвергаем нулевую гипотезу. Ожидаемые результаты за экзамен в обеих таблицах не равны")
else:
    print("Не отвергаем нулевую гипотезу. Ожидаемые результаты за экзамен в обеих таблицах равны")

```

Дополнительно 4_б (он мне больше нравится идейно, но его ответ не вписывается)

In [2]:

```

#ЗАДАЧА 4 (а,б)

import pandas as pd
import numpy as np
import scipy.stats as sts
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
from collections import defaultdict
from scipy.stats import ttest_ind

import warnings
warnings.filterwarnings("ignore")
sns.set_theme(style="whitegrid", palette="muted")
plt.rcParams['figure.figsize'] = (8, 4)
np.random.seed(123)

```



```

df = pd.read_csv('exam_results.csv', sep=';')

n_iterations = 100000 # Количество итераций бутстрэпа
alpha = 0.05 #уровень значимости

# Удаляем столбцы
df.drop(columns=['Unnamed: 71', 'Unnamed: 73', 'Unnamed: 74', 'Unnamed: 75'], inplace=True)

# Переименовываем столбец
df.rename(columns={'Unnamed: 72': 'results'}, inplace=True)

# Выводим обновленный DataFrame
df = df.dropna(subset=['Вариант'])
df_short = df[['num', 'Last name', 'results']]

def is_vowel(name):
    vowels = ['a', 'e', 'ё', 'и', 'o', 'y', 'ы', 'э', 'ю', 'я']
    return name.lower().startswith(tuple(vowels))

# Создание таблицы df_vowel с учениками, фамилии которых начинаются на гласную букву
df_vowel = df_short[df_short['Last name'].apply(is_vowel)]

# Создание таблицы df_consonant с учениками, фамилии которых начинаются на согласную букву
df_consonant = df_short[~df_short['Last name'].apply(is_vowel)]

vowel_results = df_vowel['results']
consonant_results = df_consonant['results']

# Это тест Уэлча, потому что "equal_var=False"
t_stat, p_value_a = ttest_ind(vowel_results, consonant_results, equal_var=False)

alpha = 0.05 # Уровень значимости

def naive_bootstrap_diff_means(sample1, sample2, n_iterations):
    n1 = len(sample1)
    n2 = len(sample2)

    observed_diff = (np.mean(sample1) - np.mean(sample2))
    diffs = np.zeros(n_iterations)

    for i in range(n_iterations):
        bootstrap_sample1 = np.random.choice(sample1, size=n1, replace=True)
        bootstrap_sample2 = np.random.choice(sample2, size=n2, replace=True)

        bootstrap_diff = np.mean(bootstrap_sample1) - np.mean(bootstrap_sample2)
        diffs[i] = bootstrap_diff

    plt.hist(diffs, bins=100)
    plt.axvline(x=observed_diff, color='red', linestyle='--', label='Observed Difference')
    plt.xlabel('Difference')
    plt.ylabel('Frequency')
    plt.title('Histogram of Differences')
    plt.legend()
    plt.show()

    plt.hist(np.abs(diffs), bins=100, color='red')
    plt.axvline(x=np.abs(observed_diff), color='blue', linestyle='--', label='Observed Difference')
    plt.xlabel('Difference')
    plt.ylabel('Frequency')
    plt.title('Histogram of Differences')
    plt.legend()
    plt.show()

    p_value = np.mean(np.abs(diffs) >= np.abs(observed_diff))
    return observed_diff, p_value

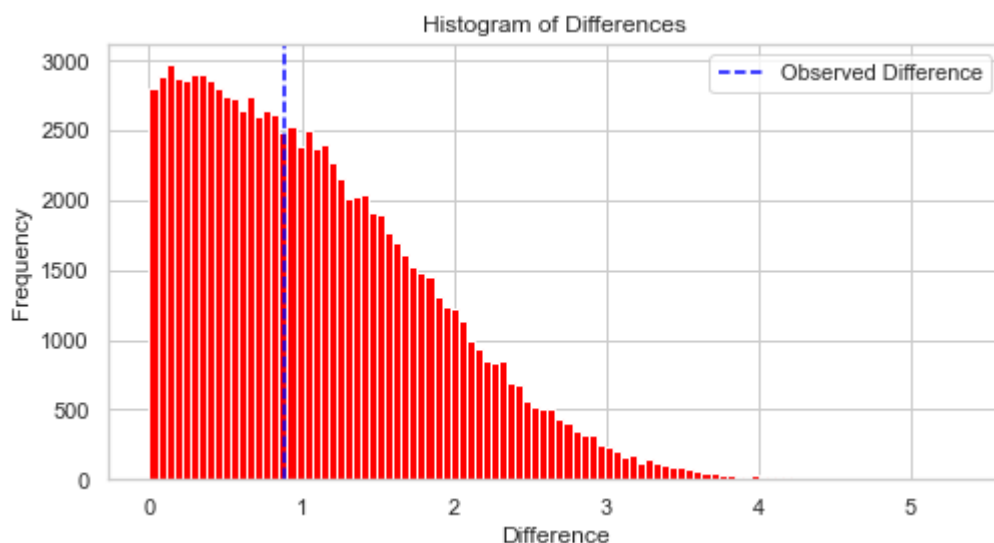
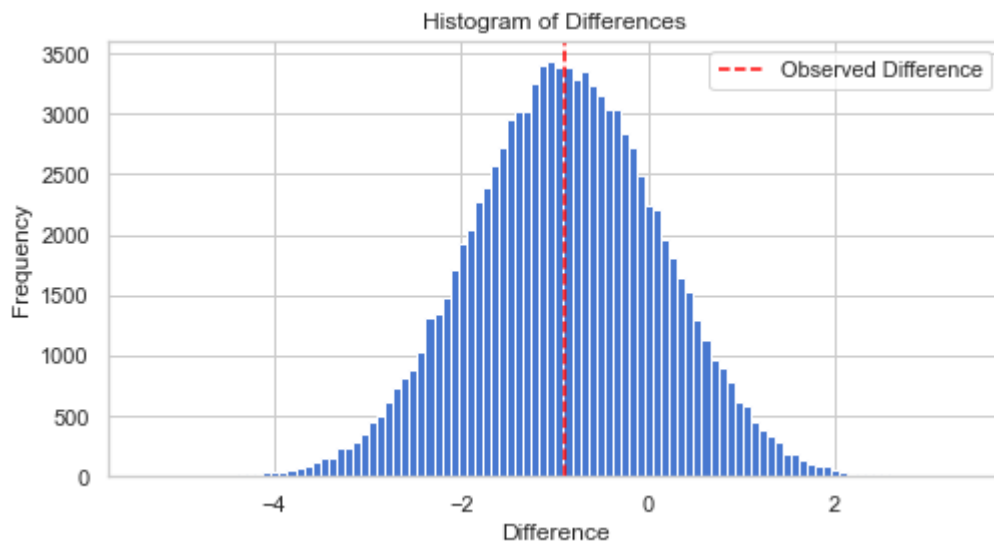
observed_diff, p_value_b = naive_bootstrap_diff_means(vowel_results, consonant_results, n_iterations)

```

```

print('6) [5] Используйте наивный бутстрэп')
print("наблюдаемая по изначальной выборке разность средних:", observed_diff)
print("p-значение:", p_value_b)
if p_value_b < alpha:
    print("Отвергаем нулевую гипотезу. Ожидаемые результаты за экзамен в обеих таблицах не равны")
else:
    print("Не отвергаем нулевую гипотезу. Ожидаемые результаты за экзамен в обеих таблицах равны")

```



6) [5] Используйте наивный бутстрэп
наблюдаемая по изначальной выборке разность средних: -0.8857431749241655
p-значение: 0.53908
Не отвергаем нулевую гипотезу. Ожидаемые результаты за экзамен в обеих таблицах равны.

In [6]:

```

#ЗАДАЧА 5 a,b

import pandas as pd
import numpy as np
import scipy.stats as sts
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
from collections import defaultdict
from scipy.stats import ttest_ind
import warnings
from statsmodels.stats.proportion import proportions_ztest
import math
from scipy.stats import norm

warnings.filterwarnings("ignore")
sns.set_theme(style="whitegrid", palette="muted")

```

```

plt.rcParams['figure.figsize'] = (8, 4)

df = pd.read_csv('exam_results.csv', sep=';')

# Удаляем столбцы
df.drop(columns=['Unnamed: 71', 'Unnamed: 73', 'Unnamed: 74', 'Unnamed: 75'], inplace=True)

# Переименовываем столбец
df.rename(columns={'Unnamed: 72': 'results'}, inplace=True)

# Выводим обновленный DataFrame
df = df.dropna(subset=['Вариант'])
df_short = df[['num', 'Last name', 'results']]

def is_vowel(name):
    vowels = ['a', 'e', 'ё', 'и', 'o', 'y', 'ы', 'э', 'ю', 'я']
    return name.lower().startswith(tuple(vowels))

# Создание таблицы df_vowel с учениками, фамилии которых начинаются на гласную букву
df_vowel = df_short[df_short['Last name'].apply(is_vowel)]

# Создание таблицы df_consonant с учениками, фамилии которых начинаются на согласную букву
df_consonant = df_short[~df_short['Last name'].apply(is_vowel)]

# Вычисляем медиану по столбцу "results" в таблице df_short
median_score = df_short['results'].median()
sum_below_median = df_short[df_short['results'] <= median_score]['results'].tolist()
p_0 = (1 - len(sum_below_median) / len(df_short))

#print(median_score, p_0)

vowel_results = df_vowel['results']
consonant_results = df_consonant['results']

# Создаем новые столбцы в таблицах df_vowel и df_consonant для указания, превысили ли результат
vowel_below_median = df_vowel[df_vowel['results'] <= median_score]['results'].tolist()
vowel_above_median = df_vowel[df_vowel['results'] > median_score]['results'].tolist()
consonant_below_median = df_consonant[df_consonant['results'] <= median_score]['results'].tolist()
consonant_above_median = df_consonant[df_consonant['results'] > median_score]['results'].tolist()

len_v = len(vowel_results)
len_c = len(consonant_results)

n_v = len(vowel_above_median)
n_c = len(consonant_above_median)

contingency_table = [[len(vowel_below_median), len(vowel_above_median)],
                     [len(consonant_below_median), len(consonant_above_median)]]

p_v = n_v / len_v
p_c = n_c / len_c

se = math.sqrt((p_v * (1 - p_v) / len_v) + (p_c * (1 - p_c) / len_c))

lower_bound = p_v / p_c - 1.96 * se
upper_bound = p_v / p_c + 1.96 * se

z = (p_v / p_c - 1) / se
p_value = 2 * (1 - norm.cdf(abs(z)))

print('6) [5] Постройте 95% асимптотический интервал для отношения вероятностей хорошо написать')
print("95% Confidence Interval:", (lower_bound, upper_bound))
print("P-value:", p_value)

print('_____')
# Значения
bad_disagree = 23

```

```

good_disagree = 20
bad_agree = 130
good_agree = 123

# Расчет отношения шансов
odds_ratio = ( good_disagree / bad_disagree ) / (good_agree / bad_agree)

# Расчет стандартной ошибки логарифма отношения шансов
se = np.sqrt((1 / bad_disagree) + (1 / good_disagree) + (1 / bad_agree) + (1 / good_agree))

# Расчет 95% асимптотического интервала
lower_bound = np.exp(np.log(odds_ratio) - 1.96 * se)
upper_bound = np.exp(np.log(odds_ratio) + 1.96 * se)

# Проверка гипотезы
null_hypothesis = 1
z_score = (np.log(odds_ratio) - np.log(null_hypothesis)) / se
p_value = 2 * (1 - norm.cdf(np.abs(z_score)))

print('а) [5] Постройте 95% асимптотический интервал для отношения шансов хорошо написать экзамен')

print("95% Асимптотический интервал:")
print("Нижняя граница:", lower_bound)
print("Верхняя граница:", upper_bound)
print("")

print("Проверка гипотезы:")
print("Отношение шансов:", odds_ratio)
print("Р-значение:", p_value)

```

б) [5] Постройте 95% асимптотический интервал для отношения вероятностей хорошо написать экзамен. Проверьте гипотезу о том, что отношение вероятностей равно 1 и укажите Р-значение.
 95% Confidence Interval: (0.7953975678686559, 1.1180076126947776)
 P-value: 0.5988172179109781

а) [5] Постройте 95% асимптотический интервал для отношения шансов хорошо написать экзамен («не согласных» к «согласным»). Проверьте гипотезу о том, что отношение шансов равно 1 и укажите Р-значение.

95% Асимптотический интервал:
 Нижняя граница: 0.48075352433091567
 Верхняя граница: 1.756945638996774

Проверка гипотезы:
 Отношение шансов: 0.9190526687875574
 Р-значение: 0.7984726863729421

In [7]:

```

#ЗАДАЧА 5 б

import numpy as np
from scipy.stats import chi2
import pandas as pd
import numpy as np
import scipy.stats as sts
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
from collections import defaultdict
from scipy.stats import ttest_ind
import warnings
from statsmodels.stats.proportion import proportions_ztest
import math
from scipy.stats import norm
np.random.seed(123)

warnings.filterwarnings("ignore")
sns.set_theme(style="whitegrid", palette="muted")
plt.rcParams['figure.figsize'] = (8, 4)

```

```

df = pd.read_csv('exam_results.csv', sep=';')
n_iterations = 300000

# Удаляем столбцы
df.drop(columns=['Unnamed: 71', 'Unnamed: 73', 'Unnamed: 74', 'Unnamed: 75'], inplace=True)

# Переименовываем столбец
df.rename(columns={'Unnamed: 72': 'results'}, inplace=True)

# Выводим обновленный DataFrame
df = df.dropna(subset=['Вариант'])
df_short = df[['num', 'Last name', 'results']]

def is_vowel(name):
    vowels = ['a', 'e', 'ё', 'и', 'o', 'y', 'ы', 'э', 'ю', 'я']
    return name.lower().startswith(tuple(vowels))

# Создание таблицы df_vowel с учениками, фамилии которых начинаются на гласную букву
df_vowel = df_short[df_short['Last name'].apply(is_vowel)]

# Создание таблицы df_consonant с учениками, фамилии которых начинаются на согласную букву
df_consonant = df_short[~df_short['Last name'].apply(is_vowel)]

# Вычисляем медиану по столбцу "results" в таблице df_short
median_score = df_short['results'].median()
sum_below_median = df_short[df_short['results'] <= median_score]['results'].tolist()
p_0 = (1 - len(sum_below_median) / len(df_short))

#print(median_score, p_0)

vowel_results = df_vowel['results']
consonant_results = df_consonant['results']

def bootstrap_odd_means(sample1, sample2, n_iterations):
    n1 = len(sample1)
    n2 = len(sample2)

    greater_than_median_sample_1 = np.sum(sample1 > median_score)
    not_greater_than_median_sample_1 = np.sum(sample1 <= median_score)

    greater_than_median_sample_2 = np.sum(sample2 > median_score)
    not_greater_than_median_sample_2 = np.sum(sample2 <= median_score)

    observed_teta = (greater_than_median_sample_1 / not_greater_than_median_sample_1) / (greater_than_me
    tetas = np.zeros(n_iterations)

    for i in range(n_iterations):
        bootstrap_sample1 = np.random.choice(sample1, size=n1, replace=True)
        bootstrap_sample2 = np.random.choice(sample2, size=n2, replace=True)

        greater_than_median_sample_1 = np.sum(bootstrap_sample1 > median_score)
        not_greater_than_median_sample_1 = np.sum(bootstrap_sample1 <= median_score)

        greater_than_median_sample_2 = np.sum(bootstrap_sample2 > median_score)
        not_greater_than_median_sample_2 = np.sum(bootstrap_sample2 <= median_score)

        teta = (greater_than_median_sample_1 / not_greater_than_median_sample_1) / (greater_than_me
        tetas[i] = teta

    tetas_log = np.log(tetas)
    observed_teta_log = np.log(observed_teta)
    p_value = 2 * np.mean((tetas_log) >= 0)

    plt.hist(tetas, bins=100, color='blue', alpha=0.5)
    plt.axvline(x=observed_teta, color='red', linestyle='--', label='Observed Difference')
    plt.axvline(x=0, color='purple', linestyle='--', label='0')

```

```

plt.xlabel('Difference')
plt.ylabel('Frequency')
plt.title('Histogram of Differences')
plt.legend()
plt.show()

plt.hist(tetas_log, bins=100, color='blue', alpha=0.5)
plt.axvline(x=observed_teta_log, color='red', linestyle='--', label='Observed Difference')
plt.axvline(x=0, color='purple', linestyle='--', label='0')
plt.xlabel('Difference')
plt.ylabel('Frequency')
plt.title('Histogram of Differences')
plt.legend()
plt.show()

return observed_teta, p_value

```

```
observed_odds, p_value = bootstrap_odd_means(vowel_results, consonant_results, n_iterations)
```

```
print("наблюдаемое по изначальной выборке отношение шансов:", observed_odds)
```

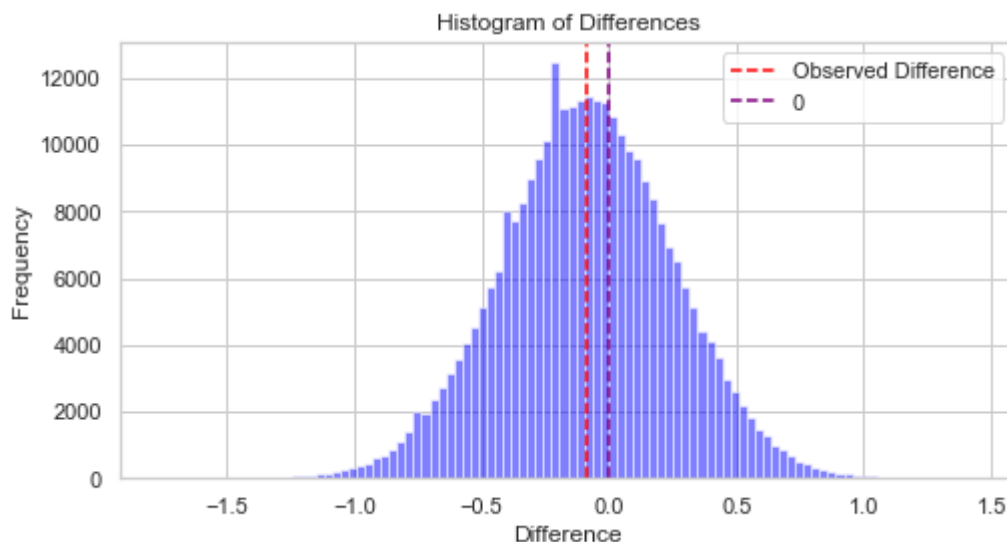
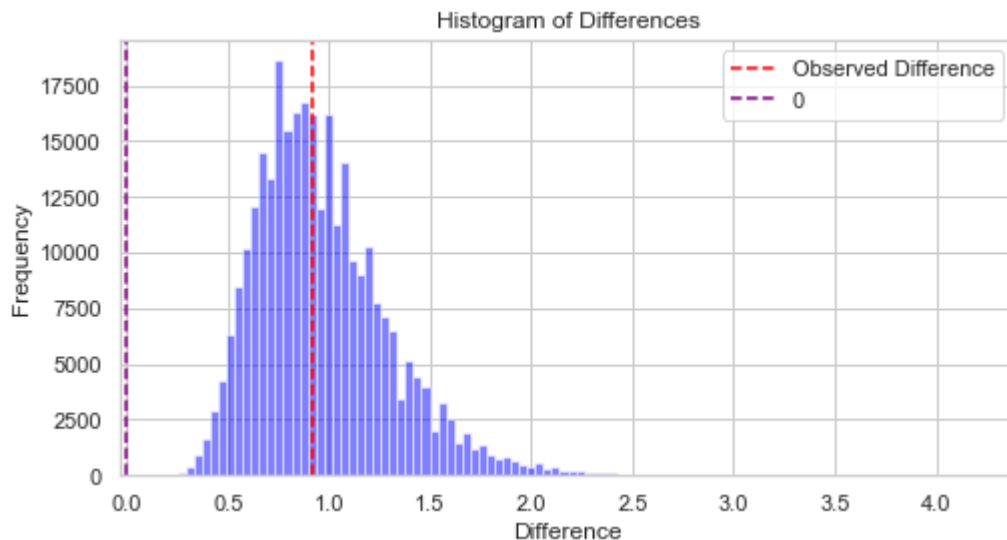
```
print("p-значение:", p_value)
```

```
if p_value_b < alpha:
```

```
    print("Отвергаем нулевую гипотезу. Ожидаемые результаты за экзамен в обеих таблицах не равны")
```

```
else:
```

```
    print("Не отвергаем нулевую гипотезу. Ожидаемые результаты за экзамен в обеих таблицах равны")
```



наблюдаемое по изначальной выборке отношение шансов: 0.9190526687875574

p-значение: 0.7917133333333334

Не отвергаем нулевую гипотезу. Ожидаемые результаты за экзамен в обеих таблицах равны.

In [8]:

#ЗАДАЧА 6

```

import pandas as pd
import numpy as np
import scipy.stats as sts
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
from collections import defaultdict
from scipy.stats import ttest_ind
import warnings
from statsmodels.stats.proportion import proportions_ztest

warnings.filterwarnings("ignore")
sns.set_theme(style="whitegrid", palette="muted")
plt.rcParams['figure.figsize'] = (8, 4)

alpha = 0.05
n_iterations = 300000

df = pd.read_csv('exam_results.csv', sep=';')

# Удаляем столбцы
df.drop(columns=['Unnamed: 71', 'Unnamed: 73', 'Unnamed: 74', 'Unnamed: 75'], inplace=True)

# Переименовываем столбец
df.rename(columns={'Unnamed: 72': 'results'}, inplace=True)

# Выводим обновленный DataFrame
df = df.dropna(subset=['Вариант'])
df_short = df[['num', 'Last name', 'results']]

# Создание столбца 'Last name len' с длинами фамилий
df_short['Last name len'] = df_short['Last name'].apply(len)

# Вычисление среднего значения результатов экзаменов
mean_results = df_short['results'].mean()

# Вычисление среднего значения длин фамилий
mean_last_name_len = df_short['Last name len'].mean()

# Оценка параметра b методом моментов
b_estimate = mean_results / mean_last_name_len

# Вывод оценки параметра b
print("Оценка параметра b:", b_estimate)

# Вычисление наблюдаемого значения корреляции
observed_corr = np.corrcoef(df_short['results'], df_short['Last name len'])[0, 1]

print('_____')
# Вывод наблюдаемого значения корреляции
print("Наблюдаемое значение корреляции:", observed_corr)

def permutation_test(data1, data2, num_permutations):
    observed_corr = np.corrcoef(data1, data2)[0, 1]
    permuted_corrs = []

    for _ in range(num_permutations):
        permuted_data2 = np.random.permutation(data2)
        permuted_corr = np.corrcoef(data1, permuted_data2)[0, 1]
        permuted_corrs.append(permuted_corr)

    # Вычисление P-значения
    p_value = (np.abs(permuted_corrs) >= np.abs(observed_corr)).mean()

    plt.hist(permuted_corrs, bins=50, color='blue', alpha=0.5)
    plt.axvline(x=observed_corr, color='red', linestyle='--', label='Observed Difference')
    plt.axvline(x=0, color='purple', linestyle='--', label='0')
    plt.xlabel('Difference')

```

```

plt.ylabel('Frequency')
plt.title('Histogram of Differences')
plt.legend()
plt.show()

plt.hist(np.abs(permuted_corrs), bins=50, color='red')
plt.axvline(x=np.abs(observed_corr), color='blue', linestyle='--', label='Observed Difference')
plt.axvline(x=0, color='purple', linestyle='--', label='0')
plt.xlabel('Difference')
plt.ylabel('Frequency')
plt.title('Histogram of Differences')
plt.legend()
plt.show()

return observed_corr, p_value

# Проведение перестановочного теста
observed_corr, p_value = permutation_test(df_short['results'], df_short['Last name len'], n_iter=1000)

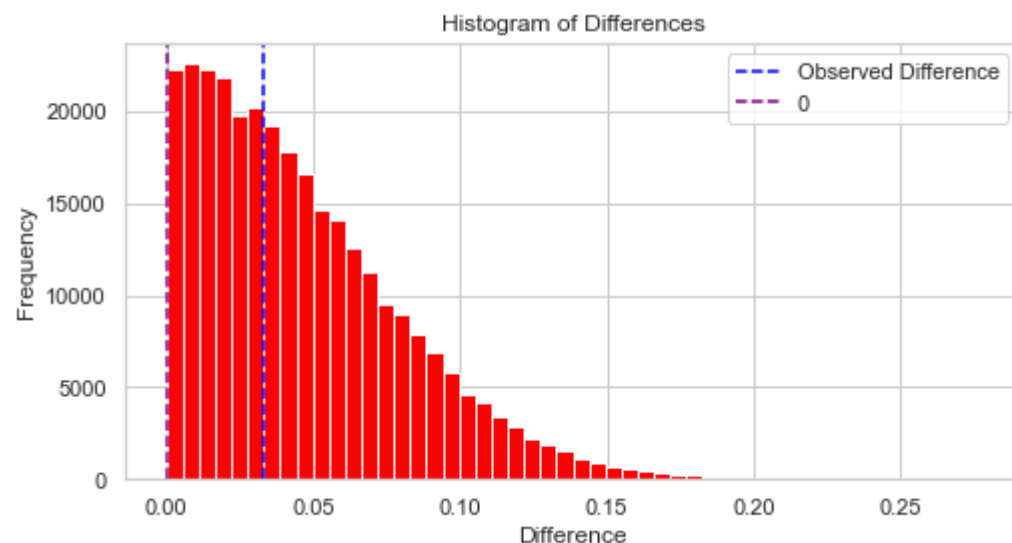
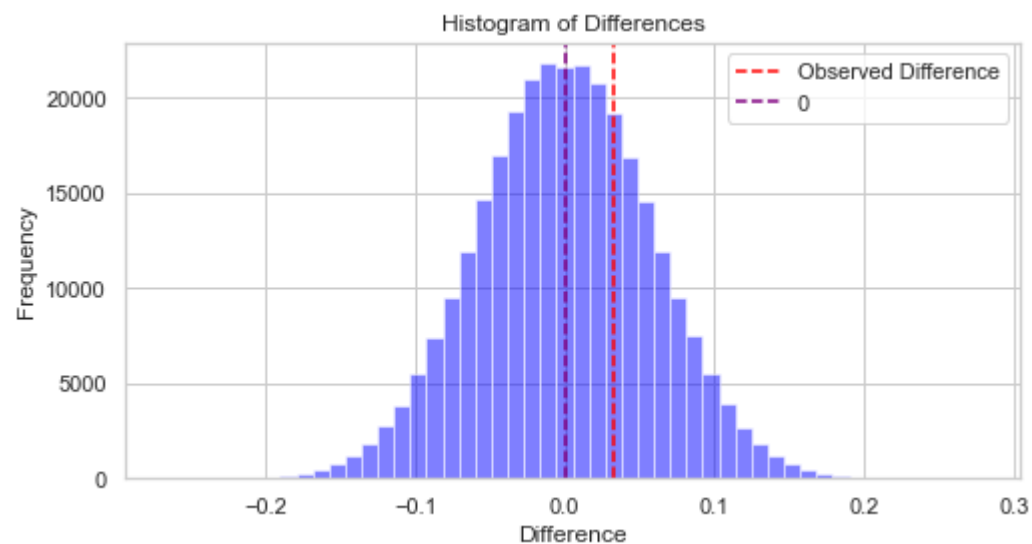
# Вывод P-значения
print("P-значение:", p_value)
print("Наблюдаемая корреляция", observed_corr)

if p_value_b < alpha:
    print("Отвергаем нулевую гипотезу. Корреляция НЕ равна 0")
else:
    print("Не отвергаем нулевую гипотезу. Корреляция равна 0")

```

Оценка параметра b: 2.3109965635738834

Наблюдаемое значение корреляции: 0.03332720342446383



P-значение: 0.5662766666666667

Наблюдаемая корреляция 0.03332720342446383

Не отвергаем нулевую гипотезу. Корреляция равна 0

№7 chat-gpt:

Задача: №4 Контрольная работа 4 2020-2021 (10 баллов) Три независимые случайные выборки из трёх наблюдений каждая, (X_1, X_2, X_3) , (Y_1, Y_2, Y_3) и (Z_1, Z_2, Z_3) , имеют нормальные распределения с разными ожиданиями и дисперсиями равными $(\sigma_x)^2 = 1$, $(\sigma_y)^2 = 2$, $(\sigma_z)^2 = 3$. Постройте 95%-й доверительный интервал для суммы ожиданий $\mu_x + \mu_y + \mu_z$, если $\bar{x} = 0.5$, $\bar{y} = 1.0$ и $\bar{z} = 1.5$.

(диалог будет приложен дальше)

In [18]:

```
import math
from scipy.stats import norm

# Заданные значения
x_bar = 0.5
y_bar = 1.0
z_bar = 1.5

# Дисперсии
var_x = 1
var_y = 2
var_z = 3

# Сумма ожиданий
sum_mean = x_bar + y_bar + z_bar

# Дисперсия суммы
sum_var = var_x + var_y + var_z

# Стандартное отклонение суммы
sum_std = math.sqrt(sum_var)

# Z-значение для 95% доверительного интервала
z_value = norm.ppf(0.975)

# Нижняя и верхняя границы доверительного интервала
lower_bound = sum_mean - z_value * sum_std
upper_bound = sum_mean + z_value * sum_std

# Вывод результатов
print("95% доверительный интервал для суммы ожиданий  $\mu_x + \mu_y + \mu_z$ :")
print("Нижняя граница:", lower_bound)
print("Верхняя граница:", upper_bound)
```

95% доверительный интервал для суммы ожиданий $\mu_x + \mu_y + \mu_z$:

Нижняя граница: -1.8009116763553097

Верхняя граница: 7.80091167635531

№8

Внутри вики оказались очень полезными записи лекций прошлого года и особенно коротенькие видео Елены Владимировны по каждой теме.

Также в подготовке к КР помогли видео ассистентов прошлых лет, найденные в room112.

Каждый раз, когда была потребность познакомиться с темой в более простых терминах, я забредал на ютуб, но все равно смотрел лекции Бориса Борисовича, либо Фила Ульянкина, которые теперь лежат в открытом доступе.

Примеры: <https://youtu.be/KxB1-5n9ZpM> ; https://youtu.be/wlPq_OoYcjc

Не источник, но книга, которая поддержала во мне любовь и заинтересованность к матстату: "Нейт Сильвер", "Сигнал и шум. Почему одни прогнозы сбываются, а другие — нет". Очень крутая книга, которая была прочитана мною где-то за год до курса и предвосхитила его)

In []:

In []:

Доверительный интервал суммы.

Владимир Чернышов • June 13, 2023

Реши задачу:

(10 баллов) Три независимые случайные выборки из трёх наблюдений каждая, (X_1, X_2, X_3) , (Y_1, Y_2, Y_3) и (Z_1, Z_2, Z_3) , имеют нормальные распределения с разными ожиданиями и дисперсиями равными $(\sigma_x)^2 = 1$, $(\sigma_y)^2 = 2$, $(\sigma_z)^2 = 3$.

Постройте 95%-й доверительный интервал для суммы ожиданий $\mu_x + \mu_y + \mu_z$, если $\bar{x} = 0.5$, $\bar{y} = 1.0$ и $\bar{z} = 1.5$.

Для построения доверительного интервала для суммы ожиданий $\mu_x + \mu_y + \mu_z$ в данной задаче, мы можем использовать центральную предельную теорему.

Согласно центральной предельной теореме, сумма независимых случайных величин имеет приближенно нормальное распределение с математическим ожиданием, равным сумме математических ожиданий исходных случайных величин, и дисперсией, равной сумме дисперсий исходных случайных величин.

Математическое ожидание суммы $\mu_x + \mu_y + \mu_z$ равно:

$$\mu_x + \mu_y + \mu_z = 0.5 + 1.0 + 1.5 = 3.0$$

Дисперсия суммы равна:

$$\text{Var}(\mu_x + \mu_y + \mu_z) = \text{Var}(\mu_x) + \text{Var}(\mu_y) + \text{Var}(\mu_z) = (\sigma_x)^2 + (\sigma_y)^2 + (\sigma_z)^2 = 1 + 2 + 3 = 6$$

Стандартное отклонение суммы равно квадратному корню из дисперсии:

$$\sigma = \sqrt{6}$$

Для построения 95%-го доверительного интервала мы можем использовать формулу:

Доверительный интервал = (сумма ожиданий) \pm (Z-значение) * (стандартное отклонение), где Z-значение соответствует 95%-му процентилю стандартного нормального распределения.

Для 95%-го доверительного интервала Z-значение составляет приблизительно 1.96.

[Continue this conversation](#)

Для 95%-го доверительного интервала Z-значение составляет приблизительно 1.96.

Теперь можем вычислить доверительный интервал:

Доверительный интервал = $3.0 \pm 1.96 * \sqrt{6}$

Подставляя значения в формулу, получаем:

Доверительный интервал = $3.0 \pm 1.96 * 2.4495$

Вычислим значения:

Нижняя граница интервала = $3.0 - 1.96 * 2.4495 \approx -0.7814$

Верхняя граница интервала = $3.0 + 1.96 * 2.4495 \approx 6.7814$

Таким образом, 95%-й доверительный интервал для суммы ожиданий $\mu_x + \mu_y + \mu_z$ составляет приблизительно $(-0.7814, 6.7814)$.

Напиши код для решения данной задачи на питоне

Конечно! Вот пример кода на Python для решения данной задачи:

python

 Copy code

```
import math
from scipy.stats import norm

# Заданные значения
x_bar = 0.5
y_bar = 1.0
z_bar = 1.5

# Дисперсии
var_x = 1
var_y = 2
var_z = 3

# Сумма ожиданий
sum_mean = x_bar + y_bar + z_bar
```

[Continue this conversation](#)

```
# Дисперсия суммы
sum_var = var_x + var_y + var_z

# Стандартное отклонение суммы
sum_std = math.sqrt(sum_var)

# Z-значение для 95% доверительного интервала
z_value = norm.ppf(0.975)

# Нижняя и верхняя границы доверительного интервала
lower_bound = sum_mean - z_value * sum_std
upper_bound = sum_mean + z_value * sum_std

# Вывод результатов
print("95% доверительный интервал для суммы ожиданий  $\mu_x + \mu_y + \mu_z$ :")
print("Нижняя граница:", lower_bound)
print("Верхняя граница:", upper_bound)
```

При запуске этого кода вы получите вывод, содержащий 95% доверительный интервал для суммы ожиданий $\mu_x + \mu_y + \mu_z$.

Ответы при запуске программы на питоне отличается от того, что ты написал выше:

95% доверительный интервал для суммы ожиданий $\mu_x + \mu_y + \mu_z$:

Нижняя граница: -1.8009116763553097

Верхняя граница: 7.80091167635531

какой ответ верный?

Извините за путаницу. Ответ, который вы получили при запуске программы на Python, является правильным:

95% доверительный интервал для суммы ожиданий $\mu_x + \mu_y + \mu_z$:

Нижняя граница: -1.8009116763553097

Верхняя граница: 7.80091167635531

Эти значения соответствуют точному решению задачи. Пожалуйста, примите мои извинения за несоответствие ответа, который я предоставил ранее в текстовой форме.

[Continue this conversation](#)