In [1]:
```
import pandas as pd
import numpy as np
from typing import List, Callable
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import stats, shapiro, kstest, mannwhitneyu, norm, probplot, boxco
from statsmodels.stats.power import TTestIndPower
from statsmodels.stats.proportion import proportion_effectsize
from wordcloud import WordCloud
import re
import pymc as pm
import arviz as az

from ab_test_methods import NonParametricMethods_2samp, DeltaAndSampleSize
```

In [2]:
```
df = pd.read_csv('data/supermarket_sales.csv')
df
```

Out[2]:

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%
0	750-67-8428	A	Yangon	Member	Female	Health and beauty	74.69	7	26.1415
1	226-31-3081	C	Naypyitaw	Normal	Female	Electronic accessories	15.28	5	3.8200
2	631-41-3108	A	Yangon	Normal	Male	Home and lifestyle	46.33	7	16.2155
3	123-19-1176	A	Yangon	Member	Male	Health and beauty	58.22	8	23.2880
4	373-73-7910	A	Yangon	Normal	Male	Sports and travel	86.31	7	30.2085
...	...	...	...	...	...	...	...	...	...
995	233-67-5758	C	Naypyitaw	Normal	Male	Health and beauty	40.35	1	2.0175
996	303-96-2227	B	Mandalay	Normal	Female	Home and lifestyle	97.38	10	48.6900
997	727-02-1313	A	Yangon	Member	Male	Food and beverages	31.84	1	1.5920
998	347-56-2442	A	Yangon	Normal	Male	Home and lifestyle	65.82	1	3.2910
999	849-09-3807	A	Yangon	Member	Female	Fashion accessories	88.34	7	30.9190

1000 rows × 17 columns

In [3]:
```
print("Количество уникальных значений в столбце gross margin percentage:",len(df['g
```

```
Количество уникальных значений в столбце gross margin percentage: 1
```

Датасет представляет собой записи о продажах в супермаркете. Ниже список столбцов:

```
Invoice ID — Номер счёта/накладной. Уникальный идентификатор каждой
транзакции
Branch — Филиал супермаркета (A, B или C)
```

City – Город, в котором расположен филиал
Customer type – Тип клиента (Member или Normal). "Member" означает, что у клиента есть карта лояльности, «Normal» – обычный клиент без членства
Gender – Пол клиента (Male/Female)
Product line – Категория товара (например, "Health and beauty", "Electronic accessories" и т.д.)
Unit price – Цена за единицу товара (в долларах)
Quantity – Количество единиц товара, купленных в этой транзакции
Tax 5% – Сумма налога (5% от стоимости товаров) в этой транзакции.
Total – Общая сумма чека с учётом налога
Date – Дата покупки (в формате дд/мм/гггг)
Time – Время покупки (в формате чч:мм)
Payment – Способ оплаты (Cash, Credit card, Ewallet)
cogs (Cost of goods sold) – Себестоимость проданных товаров без учёта налога
gross margin percentage – Процент валовой маржи (всегда 4,76% в этом датасете)
gross income – Валовая прибыль по данной транзакции (т.е. разница между выручкой и себестоимостью, фактически это Total - COGS)
Rating – Оценка клиентов по общему впечатлению от покупок/обслуживания по шкале от 1 до 10

In [4]:
```
df.info(memory_usage='Deep')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 17 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Invoice ID               1000 non-null   object 
 1   Branch                   1000 non-null   object 
 2   City                     1000 non-null   object 
 3   Customer type            1000 non-null   object 
 4   Gender                   1000 non-null   object 
 5   Product line             1000 non-null   object 
 6   Unit price               1000 non-null   float64
 7   Quantity                 1000 non-null   int64  
 8   Tax 5%                   1000 non-null   float64
 9   Total                    1000 non-null   float64
 10  Date                     1000 non-null   object 
 11  Time                     1000 non-null   object 
 12  Payment                  1000 non-null   object 
 13  cogs                     1000 non-null   float64
 14  gross margin percentage  1000 non-null   float64
 15  gross income             1000 non-null   float64
 16  Rating                   1000 non-null   float64
dtypes: float64(7), int64(1), object(9)
memory usage: 132.9+ KB
```

In [5]:
```
df[df.columns[0]] = df[df.columns[0]].astype('string')

def set_types(list_n_column:List[int] | np.ndarray, df=df, columns_type:str='catego
    for n in list_n_column:
```

```
        column_name = df.columns[n]
        df[column_name] = df[column_name].astype(columns_type)
```

In [6]:
```
set_types(list_n_column=[1,2,3,4,5,12])
set_types(list_n_column=[7], columns_type='int')
set_types(list_n_column=[6,8,9,13,14,15,16], columns_type='float')
df[df.columns[10]] = pd.to_datetime(df[df.columns[10]])
df[df.columns[11]] = pd.to_datetime(df[df.columns[11]], format='%H:%M').dt.time
```

In [7]:
```
df.info(memory_usage='Deep')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 17 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Invoice ID               1000 non-null   string
 1   Branch                   1000 non-null   category
 2   City                     1000 non-null   category
 3   Customer type            1000 non-null   category
 4   Gender                   1000 non-null   category
 5   Product line             1000 non-null   category
 6   Unit price               1000 non-null   float64
 7   Quantity                 1000 non-null   int32
 8   Tax 5%                   1000 non-null   float64
 9   Total                    1000 non-null   float64
 10  Date                     1000 non-null   datetime64[ns]
 11  Time                     1000 non-null   object
 12  Payment                  1000 non-null   category
 13  cogs                     1000 non-null   float64
 14  gross margin percentage  1000 non-null   float64
 15  gross income             1000 non-null   float64
 16  Rating                   1000 non-null   float64
dtypes: category(6), datetime64[ns](1), float64(7), int32(1), object(1), string(1)
memory usage: 88.9+ KB
```

In [8]:
```
df.describe()
```

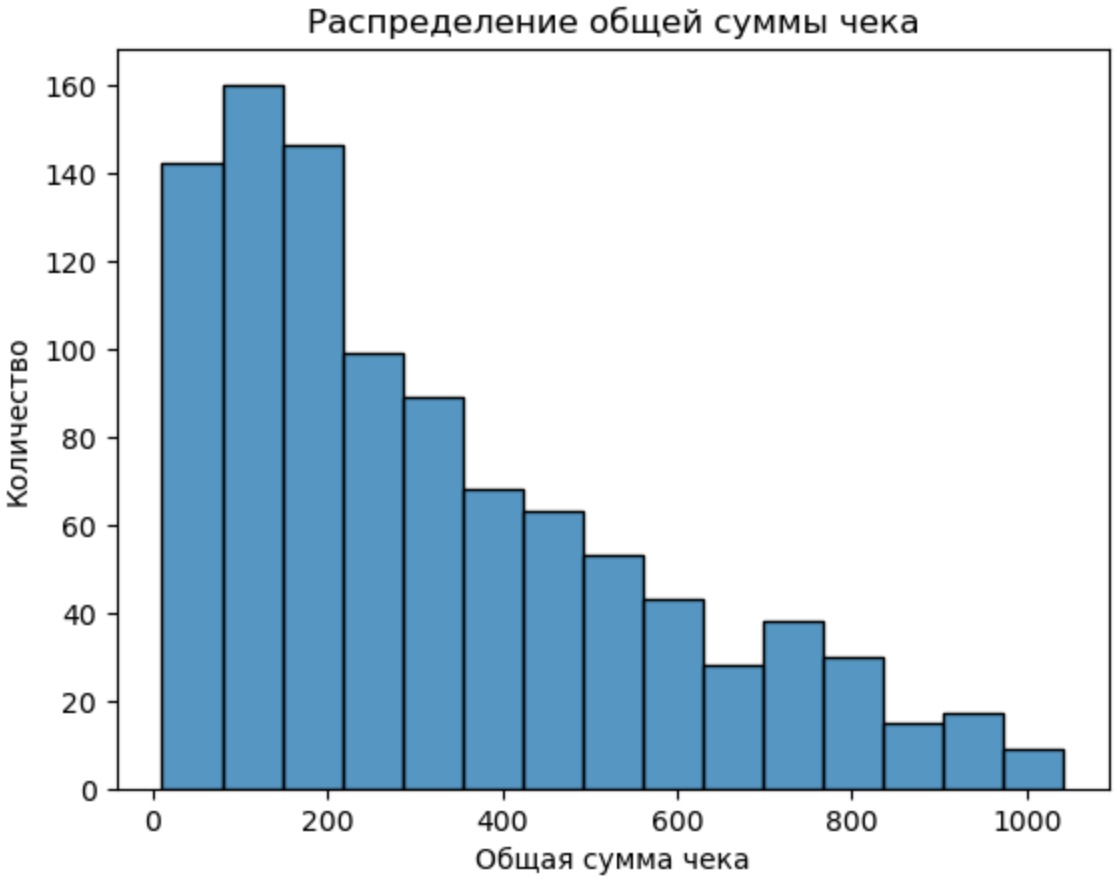
Out[8]:

	Unit price	Quantity	Tax 5%	Total	Date	cogs	gr I
count	1000.000000	1000.000000	1000.000000	1000.000000	1000	1000.00000	1.0
mean	55.672130	5.510000	15.379369	322.966749	2019-02-14 00:05:45.600000	307.58738	4.7
min	10.080000	1.000000	0.508500	10.678500	2019-01-01 00:00:00	10.17000	4.7
25%	32.875000	3.000000	5.924875	124.422375	2019-01-24 00:00:00	118.49750	4.7
50%	55.230000	5.000000	12.088000	253.848000	2019-02-13 00:00:00	241.76000	4.7
75%	77.935000	8.000000	22.445250	471.350250	2019-03-08 00:00:00	448.90500	4.7
max	99.960000	10.000000	49.650000	1042.650000	2019-03-30 00:00:00	993.00000	4.7
std	26.494628	2.923431	11.708825	245.885335	NaN	234.17651	6.

In [9]:
```
df['Total'].median()
```

Out[9]:
```
253.848
```

In [10]:
```
sns.histplot(data=df['Total'])

plt.title('Распределение общей суммы чека')
plt.xlabel('Общая сумма чека')
plt.ylabel('Количество')

plt.show()
```



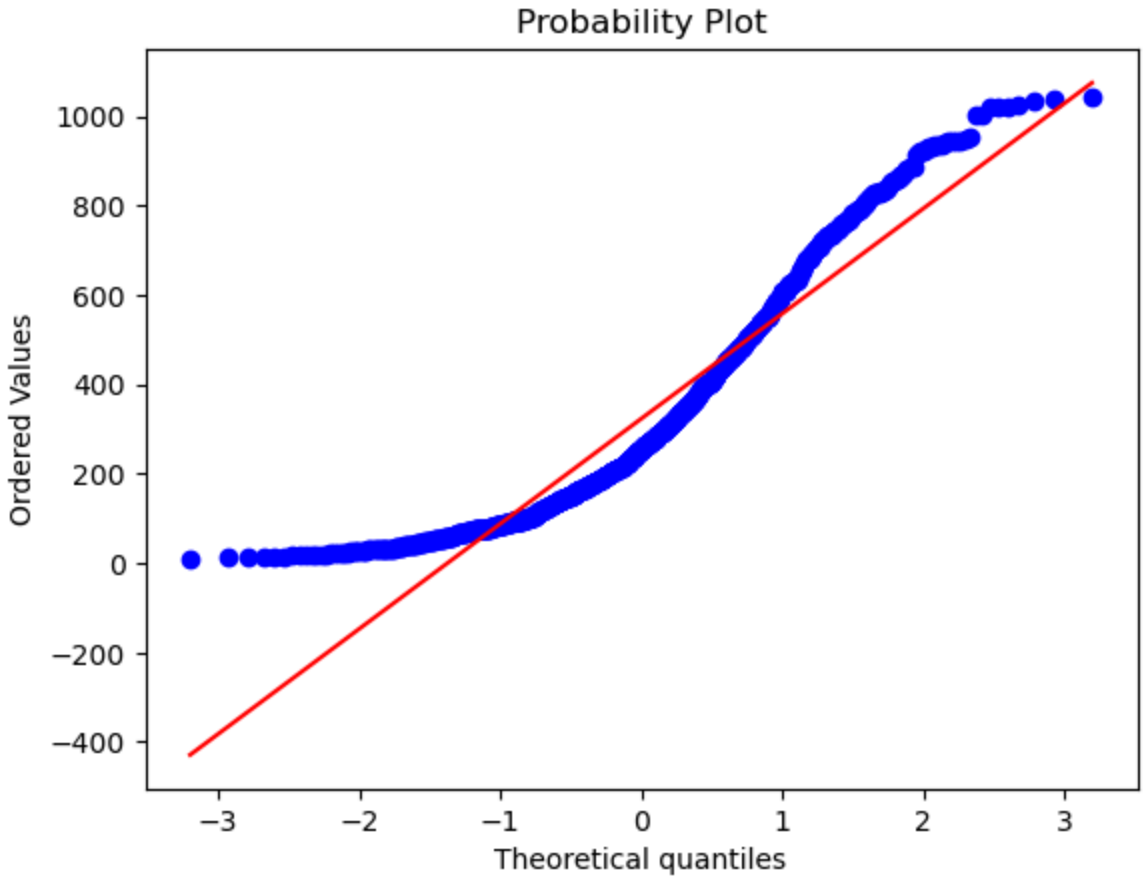
Вероятно, мы видим логнормальное распределение, которое типично для данных о сумме покупок

Проверим, так ли это

In [11]:
```
log_data = np.log(df['Total'])
stat, p = kstest(log_data, 'norm', args=(log_data.mean(), log_data.std()))
print(f'Статистика: {stat}, p-значение: {p}')
```

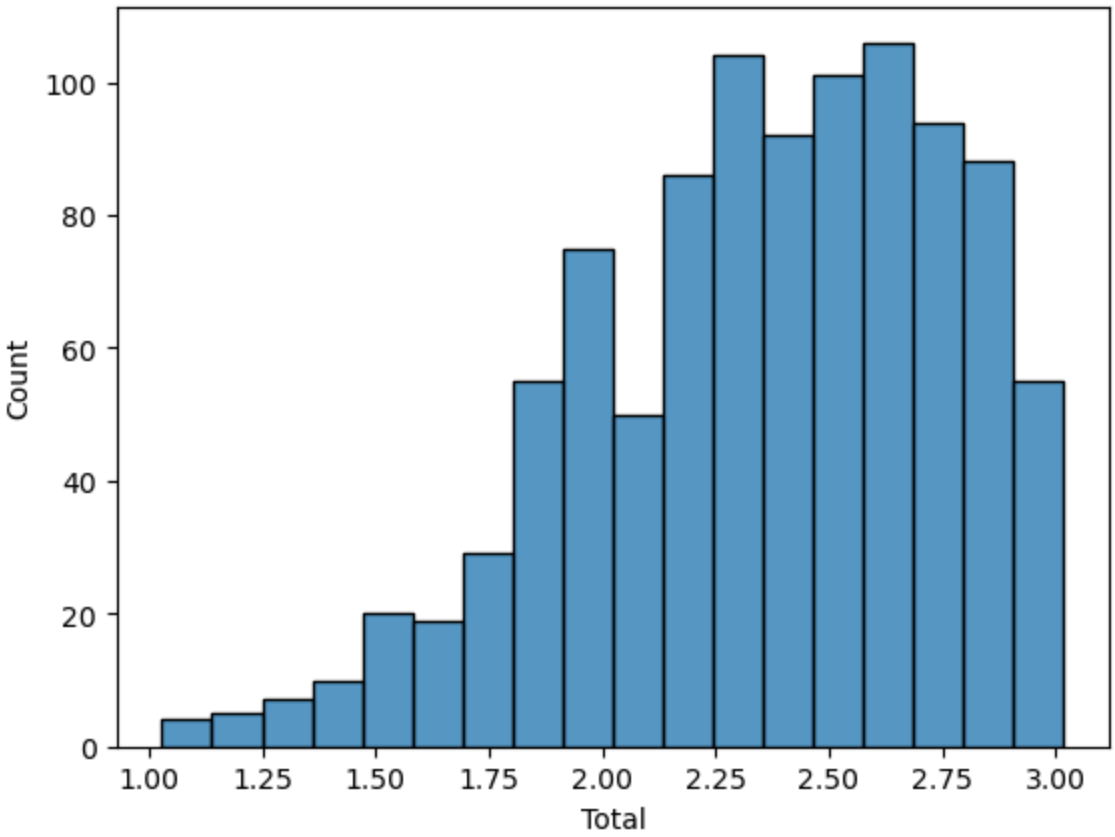
```
Статистика: 0.05610475967542117, p-значение: 0.0035429810604697878
```

In [12]:
```
probplot(df['Total'], plot=plt)
plt.show()
```



Посмотрим как выглядят данные столбца Total, когда мы логарифмируем его

In [13]:
```
sns.histplot(np.log10(df['Total'] + 1e-9))
plt.show()
```



Отклоняем гипотезу о том, что данные логнормально распределены.

*так же можно преобразовать с помощью boxcox (это расширенное лог преобразование с дополнительным коэффициентом). Я это отдельно сделал, p-значение оказалось равно 0.055, но в данном мини-кейсе я буду работать с не трансформированными данными.

Теперь, проводя a/b тестирование мы будем использовать непараметрические тесты, так как параметрические тесты могут справиться хуже (хотя из-за относительно большого размера выборки, результаты могут быть примерно одинаковы)

Проведем имитацию A/B теста, выделим 500 случайных строк для выборок A и B

Выборка A будет контрольной группой, в которой значения "как было", а выборка B - тестовая, которая получает улучшения и мы будем ожидать статистически значимые увеличения в статистиках столбца "Total".

*Я подобрал random_state так, чтобы разница была более велика

In [14]:
```
a_df = df.sample(n=500, random_state=5)
b_df = df.drop(a_df.index)
```

In [15]:
```
print('Среднее значение для выборки A:',a_df['Total'].mean())
print('Среднее значение для выборки B:',b_df['Total'].mean())
print()
```

```
print('Стандартное отклонение для выборки A:',a_df['Total'].std())
print('Стандартное отклонение для выборки B:',b_df['Total'].std())
print()
print('Медиана для выборки A:',a_df['Total'].median())
print('Медиана для выборки B:',b_df['Total'].median())
```

```
Среднее значение для выборки A: 309.334053
Среднее значение для выборки B: 336.599445

Стандартное отклонение для выборки A: 242.22261157914662
Стандартное отклонение для выборки B: 248.990197201507

Медиана для выборки A: 236.2815
Медиана для выборки B: 268.96275
```

In [16]:
```
print('Разница средних значений выборок:',np.mean(a_df['Total'])-np.mean(b_df['Tota
print()
print("Разница медиан выборок:",np.median(a_df['Total'])-np.median(b_df['Total']))
```

```
Разница средних значений выборок: -27.26539200000002

Разница медиан выборок: -32.681250000000034
```

Я создал специальный класс, который можно найти в этой же папке под именем ab_test_methods.py (в docstring класса доступно описание)

Используя его, я буду проводить тест Манна-Уитни, Bootstrap и перестановочный тест

Будем использовать уровень значимости равный 0.05

In [17]:
```
np_2samp = NonParametricMethods_2samp(a_df['Total'],b_df['Total'])
```

Критерий Манна-Уитни

Используется для сравнения двух независимых выборок и определения, есть ли статистически значимая разница между ними (сравнивает ранги значений)

In [18]:
```
np_2samp.m_whitneyu()
```

Out[18]:
```
MannwhitneyuResult(statistic=115941.5, pvalue=0.02365485954411995)
```

Критерий Манна-Уитни показывает p<0.05, на основе чего можно сказать, что статистически выборка A имеет меньшие значения относительно выборки B (с точки зрения рангов)

Bootstrap

Bootstrap для разниц средних значений выборок

In [19]:
```
np_2samp.bootstrap_ci()
```

```
Верхняя граница 95% доверительного интервала разницы mean значений: -1.4164374000000
015
Для подтверждения альтернативной гипотезы, что mean(B) > mean(A), верхняя граница до
лжна быть ниже нуля
```

Out[19]: -1.4164374000000015

Bootstrap для разниц медиан выборок

In [20]:
```
np_2samp.bootstrap_ci(func=np.median)
```

```
Верхняя граница 95% доверительного интервала разницы median значений: 3.050512500000
0013
Для подтверждения альтернативной гипотезы, что median(B) > median(A), верхняя границ
а должна быть ниже нуля
```

Out[20]: 3.0505125000000013

Верхняя граница 95% доверительного интервала разницы средних значений ниже нуля, соответственно, для этого случая H0 отвергается. Верхняя граница разницы медиан - выше, H0 не отвергается.

Перестановочный тест

Перестановочный тест для средних значений

In [21]:
```
np_2samp.perm_test()
```

```
p-значение для разниц mean значений выборок: 0.04123333333333333
Если p-значение меньше заданного ур. значимости, мы принимаем альтернативную гипотез
у о том, что mean(B) > mean(A)
```

Перестановочный тест для медиан

In [22]:
```
np_2samp.perm_test(func=np.median)
```

```
p-значение для разниц median значений выборок: 0.0745
Если p-значение меньше заданного ур. значимости, мы принимаем альтернативную гипотез
у о том, что median(B) > median(A)
```

p-значение для разниц средних значений меньше заданного уровня значимости, соответственно, мы принимаем альтернативную гипотезу

p-значение для разниц медиан больше уровня значимости, в этом случае нельзя отклонить нулевую гипотезу

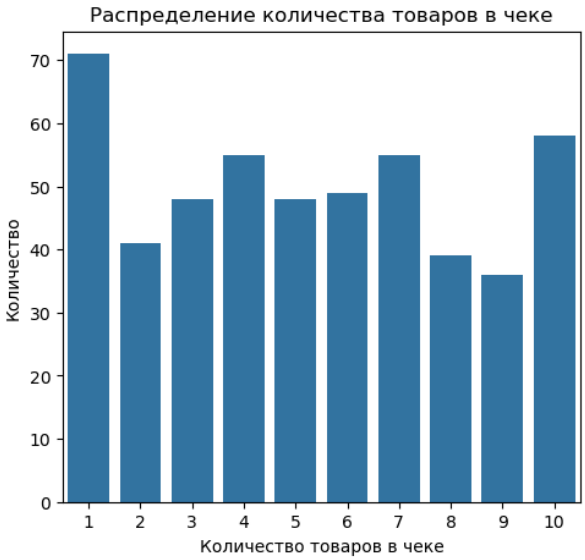
На графике оценки плотности (оцениваент вероятность в каждом интервале на основе данных) можно посмотреть плотности вероятностей значений наших выборок

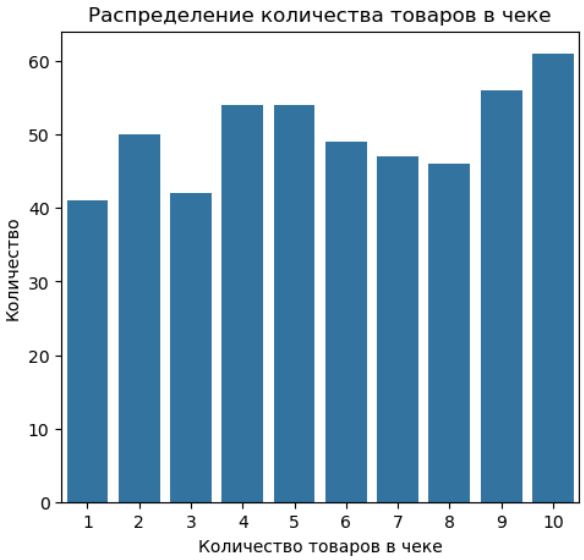
In [23]:
```
sns.kdeplot(a_df['Total'], label='Group A',bw_adjust=0.3, fill=True)
sns.kdeplot(b_df['Total'], label='Group B',bw_adjust=0.3, fill=True)
plt.legend()
plt.title('График оценки плотности')
plt.show()
```

По графику оценки плотности мы видим, что в выборке B данные больше представлены для бОльших значений, чем в выборке A.

Я проверил как распределены различные столбцы для обеих выборок, столбец с количеством товаров в чеке показал интересные результаты:

In [24]:
```
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

sns.barplot(x=a_df['Quantity'].value_counts().index, y=a_df['Quantity'].value_count
axes[0].set_title('Распределение количества товаров в чеке')
axes[0].set_xlabel('Количество товаров в чеке')
axes[0].set_ylabel('Количество')


sns.barplot(x=b_df['Quantity'].value_counts().index, y=b_df['Quantity'].value_count
axes[1].set_title('Распределение количества товаров в чеке')
axes[1].set_xlabel('Количество товаров в чеке')
axes[1].set_ylabel('Количество')

plt.show()
```





Из распределения количества товаров в чеке становится понятно, почему в выборке B среднее и медиана больше, чем в выборке A:

- в выборке A самое часто встречающееся количество товаров в чеке это 1
- в выборке B самое часто встречающееся количество товаров в чеке это 10

Таким образом:

- Тест Манна-Уитни показал статистически значимое различие (с установленным нами уровнем значимости в 0.05) в выборках

- Bootstrap и перестановочный тест для разниц средних значений отвергают нулевую гипотезу, а для медиан - нет

Вероятно, Bootstrap'у и перестановочному тесту не хватает мощности (которая зависит от размера выборки и стандартного отклонения).

Посмотрим, при каких разницах в среднем значении и медиане, мощность будет приблизительно равна 0.8 (при не изменении остальных параметров)

In [25]:
```
%%time
print('Мощность теста при разнице средних значений выборок равной 38:', np_2samp.bo
```

```
Мощность теста при разнице средних значений выборок равной 38: 0.805
CPU times: total: 1min 19s
Wall time: 1min 19s
```

In [26]:
```
%%time
print("Мощность теста при разнице медиан выборок равной 57: ",np_2samp.bootstrap_po
```

```
Мощность теста при разнице медиан выборок равной 57:  0.832
CPU times: total: 2min 58s
Wall time: 2min 58s
```

Как видим, более высокая мощность для тестирования разниц медиан достигается с бОльшим трудом. В 80% тестов интервал указывает на отличие в нужную сторону (в пользу альтернативной гипотезы), лишь при разнице медиан в 57. Для средних значений для этого требуется разница в 38.

Мощность тестов, где разница средних/медиан соответствует реальной разнице (-27 для средних и -32 для медиан), не достигает 80%, что означает недостаточную мощность. Поэтому следует увеличить объем выборки, чтоб результату было можно доверять с большей уверенностью.

Низкая мощность означает, что результаты нестабильны и при повторных экспериментах можно получить и незначимый результат. Таким образом, наблюдаемое статистически значимое различие может быть как правдивым сигналом, так и ложным срабатыванием, обусловленным особенностями распределения

Вычисление необходимого размера выборки для мощности 0.8

In [27]:
```
deltas_samples = DeltaAndSampleSize(a=a_df['Total'], b=b_df['Total'])
```

In [28]:
```
deltas_samples.sample_size(deltas_samples.cohens_delta())
```

Out[28]:
```
1004.2211555115606
```

То есть для проведения A/B тестирования нам нужно примерно 1000 пользователей в группе A и столько же в группе B

Размер эффекта по Клиффу

In [29]:
```
deltas_samples.cliffs_delta()
```

Out[29]:
```
0.072468
```

Получившийся размер эффекта говорит о том, что вероятность того, что случайно выбранное значение из группы B окажется больше случайно выбранного значения из группы A, примерно на 7 процентных пунктов выше, чем вероятность противоположного исхода

Байесовский анализ

In [30]:
```
%%time
group_A = a_df['Total']
group_B = b_df['Total']
```

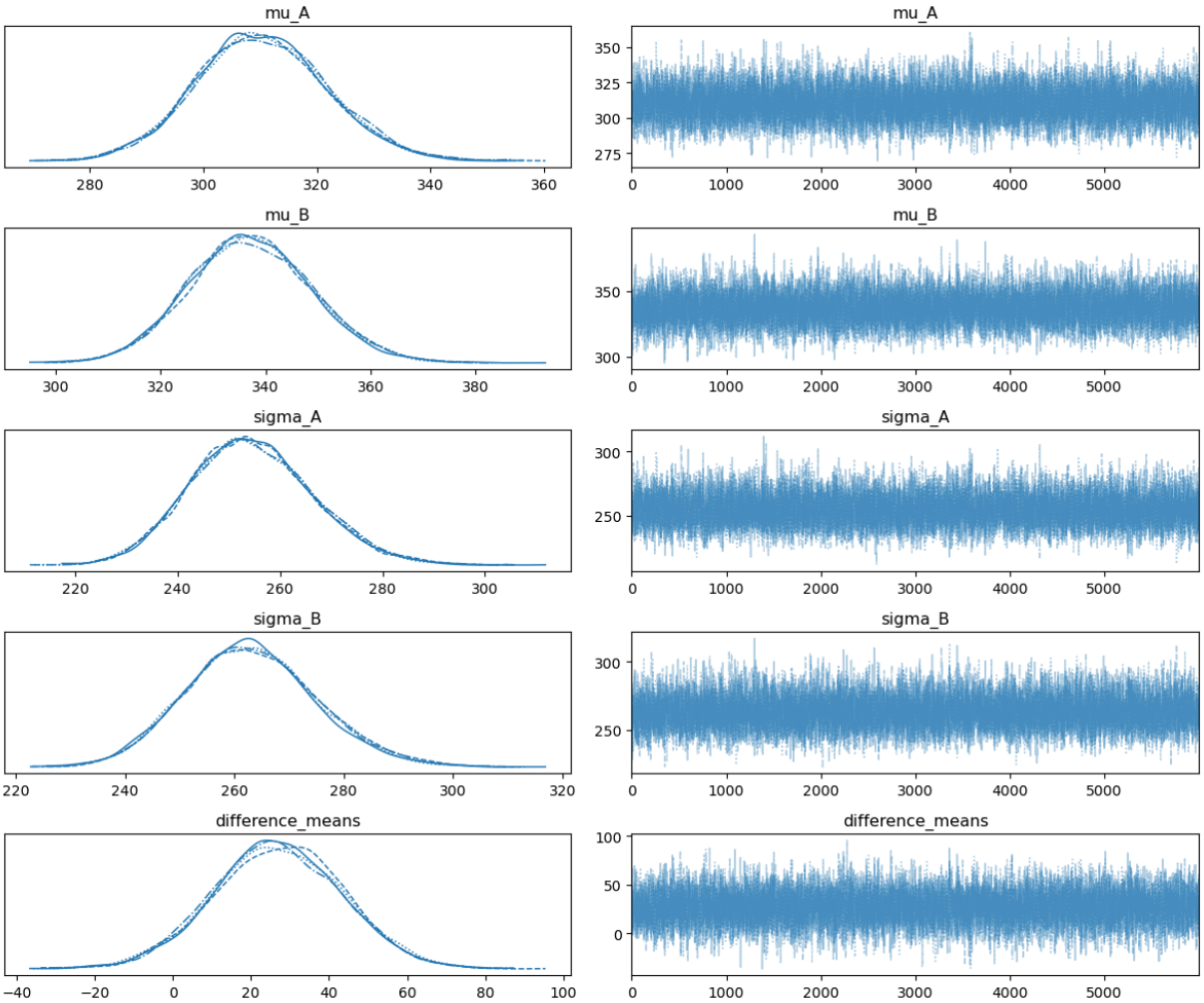
```
mu_prior_mean = 320
mu_prior_sd = 250
sigma_prior_sd = 250

with pm.Model() as model_means:

    mu_A = pm.Normal('mu_A', mu=mu_prior_mean, sigma=mu_prior_sd)
    mu_B = pm.Normal('mu_B', mu=mu_prior_mean, sigma=mu_prior_sd)
    sigma_A = pm.HalfNormal('sigma_A', sigma=sigma_prior_sd)
    sigma_B = pm.HalfNormal('sigma_B', sigma=sigma_prior_sd)

    alpha_A = mu_A**2 / sigma_A**2
    beta_A = mu_A / sigma_A**2
    alpha_B = mu_B**2 / sigma_B**2
    beta_B = mu_B / sigma_B**2

    obs_A = pm.Gamma('obs_A', alpha=alpha_A, beta=beta_A, observed=group_A)
    obs_B = pm.Gamma('obs_B', alpha=alpha_B, beta=beta_B, observed=group_B)


    diff_means = pm.Deterministic('difference_means', mu_B - mu_A)

    trace_means = pm.sample(6000, tune=3000, chains=4, cores=4, target_accept=0.95,
```

```
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [mu_A, mu_B, sigma_A, sigma_B]
Output()
```

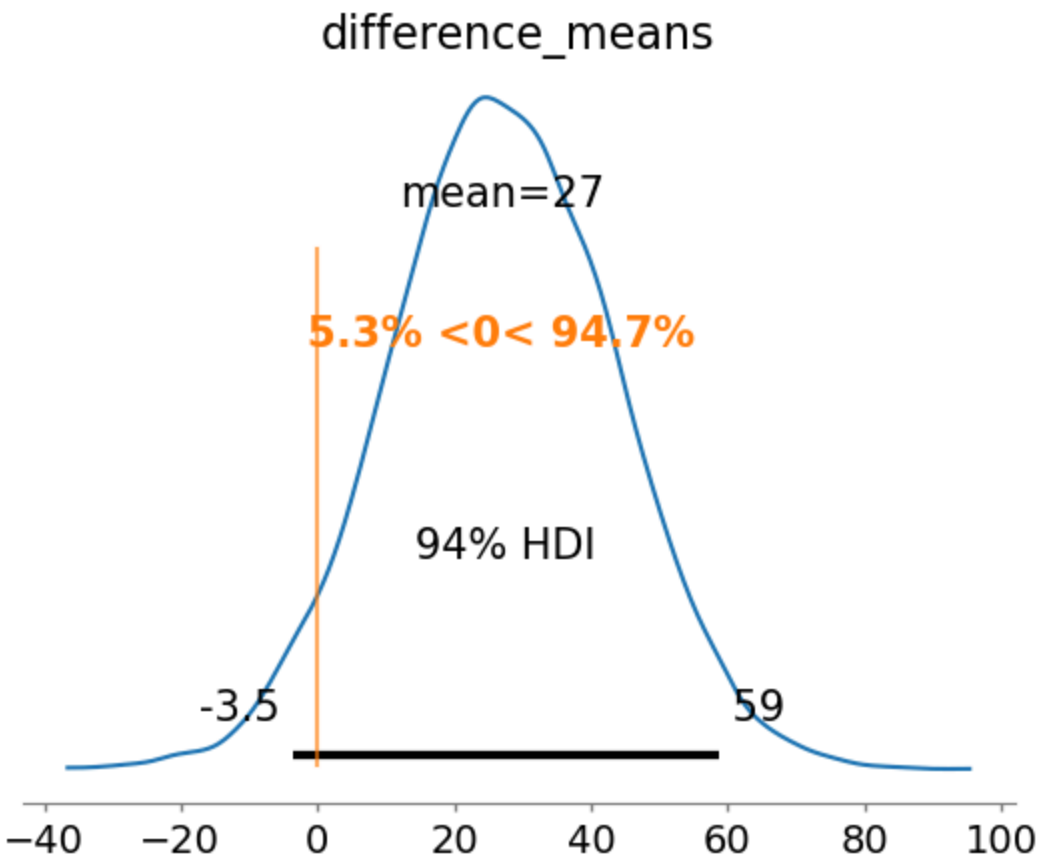
```
Sampling 4 chains for 3_000 tune and 6_000 draw iterations (12_000 + 24_000 draws to
tal) took 35 seconds.
CPU times: total: 9.55 s
Wall time: 42.4 s
```

In [31]:
```
az.plot_trace(trace_means, var_names=['mu_A', 'mu_B', 'sigma_A', 'sigma_B', 'differ
plt.tight_layout()
plt.show()

# Вероятность того, что B > A
posterior_samples = trace_means.posterior.stack(sample=("chain", "draw"))
prob_A_greater_B = (posterior_samples['mu_B'] > posterior_samples['mu_A']).mean().i
print(f"Вероятность того, что среднее B больше среднего A: {prob_A_greater_B:.3f}")

# Расчет среднего прироста и HDI
difference_samples = trace_means.posterior['mu_B'] - trace_means.posterior['mu_A']
mean_increase = difference_samples.mean(('chain', 'draw')).item()
print(f"Средний прирост (B - A): {mean_increase:.3f}")

hdi = az.hdi(difference_samples, hdi_prob=0.95)
print(f"95% HDI: [{hdi['x'].values[0]:.3f}, {hdi['x'].values[1]:.3f}]")

# Визуализация разницы средних
az.plot_posterior(trace_means, var_names=['difference_means'], ref_val=0)
plt.show()
```

```
# Таблица
az.summary(trace_means, var_names=['mu_A', 'mu_B', 'sigma_A', 'sigma_B', 'differenc
```




```
Вероятность того, что среднее B больше среднего A: 0.947
Средний прирост (B - A): 26.888
95% HDI: [-5.172, 59.461]
```



Out[31]:

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_t
mu_A	310.291	11.531	289.355	332.729	0.111	0.079	10807.0	1263
mu_B	337.179	11.764	314.739	358.873	0.113	0.080	10869.0	1208
sigma_A	254.457	12.013	233.162	277.993	0.116	0.082	10923.0	1290
sigma_B	263.141	11.973	240.527	285.342	0.117	0.083	10546.0	1106
difference_means	26.888	16.570	-3.493	58.642	0.159	0.113	10810.0	1202

Апостериорное распределение разницы средних значений между группами B и A показывает, что средняя разница оценивается в 26.888. Хотя большая часть вероятности (94.7%) сосредоточена в области положительных значений разницы (что указывает на более высокое среднее в группе B), важно отметить, что 94% HDI включает отрицательные значения.

Это означает, что, несмотря на наблюдаемую разницу в выборочных средних, данные не предоставляют убедительных доказательств в пользу того, что средние значения группы B больше таковых для группы A.

Выводы:

- Тесты (кроме разниц медиан и байесовского анализа) показывают, что значения выборки B больше значений A

- На самом деле, так как мы брали выборку из одного общего распределения, тесты для разниц медиан и байесовский анализ

показали верные результаты, а остальные совершили ошибку false positive

- В реальной практике нужно было бы проверить, что мы не ухудшаем другие метрики, соответственно их тоже нужно было бы протестировать для групп A и B

- Полученным результатам с трудом можно доверять из-за недостаточной мощности тестов (которая показывает, что результаты нестабильны)

- В реальности в таком случае я бы рекомендовал вносить изменения, если внедрение улучшений (выборка B) является относительно не затратным. Или продолжать тестирование, используя метод многорукого бандита или байесовский анализ

- Стоит помнить, что это имитация A/B теста. В реальности на его проведение влияют время сбора данных, сезонность, случайность выбора и прочие факторы