Описание проекта: исследование интернет-магазина «Подарочек»

В качестве тестового задания исследуйте данные онлайн-магазина подарков.

Магазин зачастую отправляет товары по почте, работает как с оптовыми, так и с розничными покупателями.

Вы должны применить Python для анализа данных и показать, что можете самостоятельно решить задачу по исследованию данных, сегментации клиентов и проверке статистических гипотез.

Описание данных

Основной датасет (df)

- entry_date — дата записи;
- order_id — идентификационный номер заказа;
- customer_id — идентификационный номер клиента;
- quantity — количество;
- price — цена;
- name_clust — автоматически присвоенная группа записи на основе названия;
- entry_id — идентификационный номер записи;
- country_id — идентификационный номер страны.

Текстовое описание записей (df_text)

- entry_id — идентификационный номер записи;
- entry — запись.

Датасеты содержит данные, которые несут в себе информацию о клиентах онлайн магазина Подарочек

- основной датасет содержит информацию о заказах, включая дату, идентификаторы заказов и клиентов, количество товаров, их цену
- текстовый датасет дополняет основной датасет описаниями записей

Цель работы

- Необходимо исследовать данные онлайн-магазина подарков с целью анализа поведения клиентов, сегментации их на основе различных характеристик и проверки статистических гипотез.
- Данный проект поможет как понять исследуемых клиентов,так и улучшить взаимодействие компании с ними, что приведет к повышению эффективность бизнесса компании.

План работы:

- Шаг 1. Получение, осмотр и объединение данных
  - Загрузка данных из csv-файлов в датафреймы.
  - Изучить общую информацию о датафреймах.
  - Проверить наличие пропусков и принять решение о их заполнении.
  - Проверить наличие дубликатов и принять решение о их удалении.
  - Привести типы данных в каждом столбце к нужным форматам.
  - Проверить соответствие идентификационных номеров.
  - Объединить информацию из всех датафреймов в один.
- Шаг 2. Предобработка и начало исследовательского анализа
  - Найдите выбросы и аномальные значения в столбцах price и quantity, рассчитайте сумму стоимости каждой товарной позиции в датасете, примите и реализуйте решение о сохранении или отбрасывании подозрительных данных.
  - Изучите столбцы order_id, customer_id, name_clust, entry_id и country_id.
  - Изучите полноту данных, анализируя время записей. Посчитайте по месяцам количество дней, в которых не было продаж. Выберите период для анализа, содержащий основную часть данных, и далее работайте только с актуальными данными.
- Шаг 3. Расчёт метрик
  - Оцените по часам и дням недели количество заказов и количество уникальных покупателей. Постройте графики и сделайте вывод о наличии цикличности в покупательской активности.
  - Рассчитайте по месяцам среднюю выручку с клиента в день и количество уникальных покупателей. Сделайте вывод о наличии или отсутствии сезонности, если это возможно.
  - Рассчитайте стики-фактор за второй и третий квартал 2019 года.
  - Составьте профиль каждого клиента, включите в профиль количество заказов, дату первого и последнего заказа, общую сумму всех заказов, среднюю цену заказа, а также другие показатели по вашему выбору.
  - Разделите клиентов на возвратных и нет по признаку наличия повторных покупок, для каждой из групп на основе профилей клиентов (когда это возможно) рассчитайте средние показатели и оцените их.
- Шаг 4. Проведение RFM-сегментацию клиентов
  - разделите клиентов на группы по методике RFM;
  - оцените получившиеся группы, найдите похожие и различающиеся;

- сформулируйте рекомендации для бизнеса по взаимодействию с сегментами, сопроводив их подходящими графиками и таблицами.

- Шаг 5.Проверка статистических гипотез
  - Сравните доли возвратных и невозвратных клиентов за второй и третий квартал 2019 года при помощи подходящего статистического теста.
  - Сравните средние чеки в странах с country_id, равному 3, 6 и 24. На основе статистических тестов сделайте вывод о том, отличаются ли средние чеки в этих странах или нет.
  - Сформулируйте собственную гипотезу и проверьте её.
- Шаг 6. Выводы по проекту
  - Опишите полученные результаты и зафиксируйте итоговый вывод проведённого исследования.

Шаг 1. Получение, осмотр и объединение данных

1. Загрузка данные из csv-файлов в датафреймы.

In [70]:
```
import pandas as pd
df = pd.read_csv('https://code.s3.yandex.net/datasets/gift.csv')
df
```

Out[70]:

	entry_date	order_id	customer_id	quantity	price	name_clust	entry_id	country_id
0	12/01/2018 08:26	3031	2150	6	339	740	891	28
1	12/01/2018 08:26	3031	2150	8	275	132	1596	28
2	12/01/2018 08:26	3031	2150	6	339	197	166	28
3	12/01/2018 08:26	3031	2150	2	765	767	1810	28
4	12/01/2018 08:26	3031	2150	6	425	383	2585	28
...	...	...	...	...	...	...	...	...
356935	12/09/2019 12:50	48253	7320	12	85	556	2723	5
356936	12/09/2019 12:50	48253	7320	6	210	144	1246	5
356937	12/09/2019 12:50	48253	7320	4	415	144	1241	5
356938	12/09/2019 12:50	48253	7320	4	415	114	2811	5
356939	12/09/2019 12:50	48253	7320	3	495	785	1818	5

356940 rows × 8 columns

In [71]:
```
df_text = pd.read_csv('https://code.s3.yandex.net/python-for-analytics/gift_entry.csv', sep=';', error_bad_lines=False, warn_bad_lines=True
df_text
```

Out[71]:

	entry_id	entry
0	0	NaN
1	1	10-цветная ручка Spaceboy
2	2	12 карандашей, черепа
3	3	12 карандашей, высокий тюбик, лесной массив
4	4	12 карандашей, маленький тюбик с черепом
...	...	...
2912	2912	яйцо с подвесным украшением из слоновой кости
2913	2913	янтарное массивное колье из стекла+бусины
2914	2914	яркие голубые ленты
2915	2915	ящик для хранения большой, черепа
2916	2916	ящик для хранения маленький, черепа

2917 rows × 2 columns

2. Изучите общую информацию о датафреймах.

In [72]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 356940 entries, 0 to 356939
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype 
---  ------       --------------   ----- 
 0   entry_date   356940 non-null  object
 1   order_id     356940 non-null  int64 
 2   customer_id  356940 non-null  int64 
 3   quantity     356940 non-null  int64 
 4   price        356940 non-null  int64 
 5   name_clust   356940 non-null  int64 
 6   entry_id     356940 non-null  int64 
 7   country_id   356940 non-null  int64 
dtypes: int64(7), object(1)
memory usage: 21.8+ MB
```

- в основном датафрейме нет пропусков данных
- стобец entry_date имеет тип object (строки), в дальнейшим преобразуем в тип данных datetime
- остальные стообцы имеют тип int64, что соответствует числовым значениям

In [73]:
```
df_text.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2917 entries, 0 to 2916
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   entry_id  2917 non-null   int64 
 1   entry     2916 non-null   object
dtypes: int64(1), object(1)
memory usage: 45.7+ KB
```

- в текстовом датафрейме есть пропуск в entry - в дальнешим заполним пропуски значением 'Unknown'
- столбец entry_id имеет тип int64, что соответствует числовым значениям идентификаторов.
- столбец entry имеет тип object, что соответствует текстовым значениям записей.

3. Проверьте наличие пропусков, примите решение о заполнении.

In [74]:
```
df.isnull().sum()
```

Out[74]:
```
entry_date     0
order_id       0
customer_id    0
quantity       0
price          0
name_clust     0
entry_id       0
country_id     0
dtype: int64
```

- продублируем процесс, хотя еще в info было показано,что пропуско в этом файле нет

In [75]:
```
df_text.isnull().sum()
```

Out[75]:
```
entry_id    0
entry       1
dtype: int64
```

In [76]:
```
df_text['entry'].fillna('Unknown', inplace=True)
df_text.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2917 entries, 0 to 2916
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   entry_id  2917 non-null   int64 
 1   entry     2917 non-null   object
dtypes: int64(1), object(1)
memory usage: 45.7+ KB
```

данные в столбце entry не содержат пропусков

4. Проверьте наличие дубликатов, примите решение об удалении

In [77]:
```
df.shape
```

Out[77]:
```
(356940, 8)
```

In [78]:
```
df_text.shape
```

Out[78]:
```
(2917, 2)
```

In [79]:
```
# Рассчитаем количество дубликатов
duplicates_df = df.duplicated().sum()
```

```
print(f"Количество дубликатов в основном файле: {duplicates_df}")
print(f"Процент дубликатов в основном файле: {(duplicates_df / df.shape[0]) *100:.2f}%")
```

```
Количество дубликатов в основном файле: 3573
Процент дубликатов в основном файле: 1.00%
```

In [80]:
```
# Рассчитаем количество дубликатов
duplicates_df_text = df_text.duplicated().sum()
print(f"Количество дубликатов в текстовом файле: {duplicates_df_text}")
print(f"Процент дубликатов в текстовом файле: {(duplicates_df_text / df_text.shape[0]) *100:.2f}%")
```

```
Количество дубликатов в текстовом файле: 0
Процент дубликатов в текстовом файле: 0.00%
```

- таким образом дупликаты встречаются только в текстовом файле и их объём составляет 1 процент
- их решено удалить

In [81]:
```
df=df.drop_duplicates()
```

In [82]:
```
# Рассчитаем количество дубликатов
duplicates_df = df.duplicated().sum()
print(f"Количество дубликатов в основном файле: {duplicates_df}")
```

```
Количество дубликатов в основном файле: 0
```

5. Рассмотрите типы данных в каждом столбце, приведите типы (если нужно)

In [83]:
```
# Проверка типов данных в основном датафрейме
print(df.dtypes)
```

```
entry_date     object
order_id        int64
customer_id     int64
quantity        int64
price           int64
name_clust      int64
entry_id        int64
country_id      int64
dtype: object
```

In [84]:
```
# Проверка типов данных в текстовом датафрейме
print(df_text.dtypes)
```

```
entry_id     int64
entry       object
dtype: object
```

- в основном датафрейме необходимо в столбце entry_date в тип datetime
- в текстовом датафрейме типы данных корректны

In [86]:
```
# Преобразование столбца entry_date в тип datetime
df['entry_date'] = pd.to_datetime(df['entry_date'])

# Проверка типов данных после преобразования
print(df.dtypes)
```

```
entry_date     datetime64[ns]
order_id                int64
customer_id             int64
quantity                int64
price                   int64
name_clust              int64
entry_id                int64
country_id              int64
dtype: object
/tmp/ipykernel_31/2307869771.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['entry_date'] = pd.to_datetime(df['entry_date'])
```

In [87]:
```
# Преобразование столбца entry_date в тип datetime с использованием .loc
df.loc[:, 'entry_date'] = pd.to_datetime(df['entry_date'])

# Проверка типов данных после преобразования
print(df.dtypes)
```

```
entry_date     datetime64[ns]
order_id                int64
customer_id             int64
quantity                int64
price                   int64
name_clust              int64
entry_id                int64
country_id              int64
dtype: object
```

6. Проверьте соответствие идентификационных номеров.

In [88]:
```
# Проверка соответствия идентификационных номеров
print(df['entry_id'].isin(df_text['entry_id']).all())
```

```
True
```

In [89]:
```
print(df['entry_id'].nunique())
print(df_text['entry_id'].nunique())
```

```
2917
2917
```

Количество уникальных идентификаторов entry_id в обоих датафреймах совпадает и составляет 2917.

7. Объедините информацию из всех датафреймов в один.

In [90]:
```
# Объединение информации из обоих датафреймов
comb_data = pd.merge(df, df_text, on='entry_id', how='left')

# Проверка первых строк объединенного датафрейма
display(comb_data.head())
```

	entry_date	order_id	customer_id	quantity	price	name_clust	entry_id	country_id	entry
0	2018-12-01 08:26:00	3031	2150	6	339	740	891	28	белый металлический фонарь
1	2018-12-01 08:26:00	3031	2150	8	275	132	1596	28	кремовая вешалка в форме сердечек Купидона
2	2018-12-01 08:26:00	3031	2150	6	339	197	166	28	Вязаная грелка с флагом Союза
3	2018-12-01 08:26:00	3031	2150	2	765	767	1810	28	набор 7 скворечников для бабушек
4	2018-12-01 08:26:00	3031	2150	6	425	383	2585	28	стеклянный матовый держатель в форме звезды

In [91]:
```
# Проверка типов данных и информации о датафрейме
print(comb_data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 353367 entries, 0 to 353366
Data columns (total 9 columns):
 #   Column       Non-Null Count   Dtype         
---  ------       --------------   -----         
 0   entry_date   353367 non-null  datetime64[ns]
 1   order_id     353367 non-null  int64         
 2   customer_id  353367 non-null  int64         
 3   quantity     353367 non-null  int64         
 4   price        353367 non-null  int64         
 5   name_clust   353367 non-null  int64         
 6   entry_id     353367 non-null  int64         
 7   country_id   353367 non-null  int64         
 8   entry        353367 non-null  object        
dtypes: datetime64[ns](1), int64(7), object(1)
memory usage: 27.0+ MB
None
```

- в объединённой таблице пропусков нет
- форматы правильны
- перед предобработкой проверим дубликаты

In [92]:
```
# Рассчитаем количество дубликатов
duplicates_comb_data = comb_data.duplicated().sum()
print(f"Количество дубликатов в объединённом датафрейме: {duplicates_comb_data}")
```

```
Количество дубликатов в объединённом датафрейме: 0
```

In [93]:
```
comb_data.describe()
```

Out[93]:

	order_id	customer_id	quantity	price	name_clust	entry_id	country_id
count	353367.000000	353367.000000	353367.000000	3.533670e+05	353367.000000	353367.000000	353367.000000
mean	26663.279831	3479.992538	10.218348	4.016779e+02	468.644602	1517.811349	26.741045
std	13368.784949	2549.306356	147.510432	5.084618e+03	259.160574	833.700318	4.998306
min	3031.000000	-1.000000	-9600.000000	-1.106206e+06	0.000000	0.000000	0.000000
25%	14832.000000	-1.000000	1.000000	1.250000e+02	242.000000	875.000000	28.000000
50%	27316.000000	3630.000000	3.000000	2.080000e+02	448.000000	1558.000000	28.000000
75%	38445.000000	5633.000000	10.000000	4.130000e+02	702.000000	2223.000000	28.000000
max	48253.000000	7653.000000	80995.000000	1.354133e+06	929.000000	2916.000000	29.000000

Выводы по шагу 1

- Основной датафрейм содержит данные без пропусков, однако столбец entry_date преобразован в тип datetime.
- В текстовом датафрейме есть один пропуск в столбце entry.Был заполнен значением 'Unknown'.
- Дубликаты присуствуют только в основном датафрейме. Они были удалены
- Все идентификаторы в основном датафрейме имеют соответствия в текстовом.

- Создан объединенный датафрейм comb_data, который содержит всю информацию из обоих исходных датафреймов:

*** к содержанию**

Шаг 2. Предобработка и начало исследовательского анализа

1. Найдите выбросы и аномальные значения в столбцах price и quantity

- рассчитайте сумму стоимости каждой товарной позиции в датасете
- примите и реализуйте решение о сохранении или отбрасывании подозрительных данных.

начнём с метода describe, кооторый хорошо показывают основную статистику по набору в столбце

In [94]:
```
comb_data.shape
```

Out[94]:
```
(353367, 9)
```

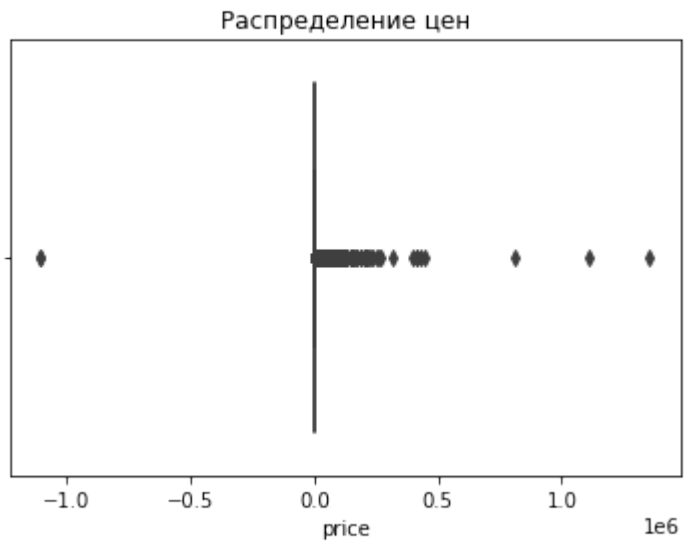
In [95]:
```
# Основные статистические показатели для 'price'
price_stats = comb_data['price'].describe()
print(price_stats)
print('')
# Основные статистические показатели для 'quantity'
quantity_stats = comb_data['quantity'].describe()
print(quantity_stats)
```

```
count    3.533670e+05
mean     4.016779e+02
std      5.084618e+03
min     -1.106206e+06
25%      1.250000e+02
50%      2.080000e+02
75%      4.130000e+02
max      1.354133e+06
Name: price, dtype: float64

count    353367.000000
mean         10.218348
std         147.510432
min       -9600.000000
25%           1.000000
50%           3.000000
75%          10.000000
max       80995.000000
Name: quantity, dtype: float64
```

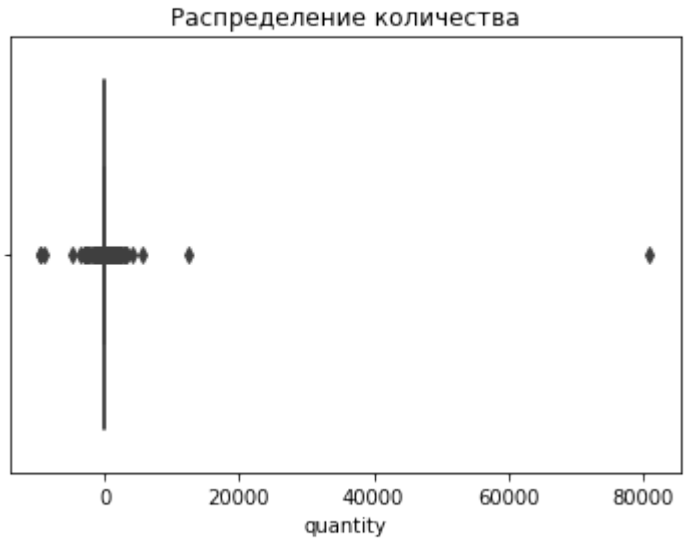
- присуствуют отрицательные значения и сильно большие положительные значения, что безусловно указывает на выбросы и аномалии.

теперь визуализируем колонки price и quantity

In [96]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.boxplot(x=comb_data['price'])
plt.title('Распределение цен')
plt.show()
```



In [97]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.boxplot(x=comb_data['quantity'])
plt.title('Распределение количества')
plt.show()
```



выделим выбросы и фильруем

In [98]:
```
# Выявление выбросов для 'price'
Q1_price = comb_data['price'].quantile(0.25)
Q3_price = comb_data['price'].quantile(0.75)
IQR_price = Q3_price - Q1_price
lb_price = Q1_price - 1.5 * IQR_price
upb_price = Q3_price + 1.5 * IQR_price
# Фильтрация выбросов для 'price'
price_lb_upb = comb_data[(comb_data['price'] < lb_price) | (comb_data['price'] > upb_price)]

print(f'Количество выбросов в столбце price: {price_lb_upb.shape[0]}')
print(f'процент от всех данных: {(price_lb_upb.shape[0]/comb_data.shape[0])*100}')
```

```
Количество выбросов в столбце price: 26875
процент от всех данных: 7.60540740929402
```

In [99]:
```
# Выявление выбросов для 'quantity'
Q1_quantity = comb_data['quantity'].quantile(0.25)
Q3_quantity = comb_data['quantity'].quantile(0.75)
IQR_quantity = Q3_quantity - Q1_quantity
lb_quantity = Q1_quantity - 1.5 * IQR_quantity
upb_quantity = Q3_quantity + 1.5 * IQR_quantity
# Фильтрация выбросов для 'quantity'
quantity_lb_upb = comb_data[(comb_data['quantity'] < lb_quantity) | (comb_data['quantity'] > upb_quantity)]

print(f'Количество выбросов в столбце quantity: {quantity_lb_upb.shape[0]}')
print(f'процент от всех данных: {(quantity_lb_upb.shape[0]/comb_data.shape[0])*100}')
```

```
Количество выбросов в столбце quantity: 39365
процент от всех данных: 11.139976285278479
```

In [100…
```
# Фильтрация данных без выбросов
filtered_data = comb_data[
    (comb_data['price'] >= lb_price) & (comb_data['price'] <= upb_price) &
    (comb_data['quantity'] >= lb_quantity) & (comb_data['quantity'] <= upb_quantity)
]

print(f'Размер датафрейма после удаления выбросов: {filtered_data.shape}')

# Рассчитаем уменьшение количества строк
print(f'Количество строк уменьшилось на: {(1 - (filtered_data.shape[0] / comb_data.shape[0])) * 100}%')
```

```
Размер датафрейма после удаления выбросов: (287428, 9)
Количество строк уменьшилось на: 18.660203131588403%
```

далее работаем с filtered_data

рассчитаем сумму стоимости каждой товарной позиции в датасете

In [103…
```
# Рассчитаем общую стоимость каждой товарной позиции
filtered_data['total_cost'] = filtered_data['quantity'] * filtered_data['price']
filtered_data.head()
```

```
/tmp/ipykernel_31/1607913489.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  filtered_data['total_cost'] = filtered_data['quantity'] * filtered_data['price']
```

Out[103…

	entry_date	order_id	customer_id	quantity	price	name_clust	entry_id	country_id	entry	total_cost
0	2018-12-01 08:26:00	3031	2150	6	339	740	891	28	белый металлический фонарь	2034
1	2018-12-01 08:26:00	3031	2150	8	275	132	1596	28	кремовая вешалка в форме сердечек Купидона	2200
2	2018-12-01 08:26:00	3031	2150	6	339	197	166	28	Вязаная грелка с флагом Союза	2034
3	2018-12-01 08:26:00	3031	2150	2	765	767	1810	28	набор 7 скворечников для бабушек	1530
4	2018-12-01 08:26:00	3031	2150	6	425	383	2585	28	стеклянный матовый держатель в форме звезды	2550

In [106…

```
# Рассчитаем общую стоимость каждой товарной позиции с использованием .loc
filtered_data.loc[:, 'total_cost'] = filtered_data['quantity'] * filtered_data['price']

# Проверка первых строк данных после преобразования
filtered_data.head()
```

```
/opt/conda/lib/python3.9/site-packages/pandas/core/indexing.py:1676: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_column(ilocs[0], value, pi)
```

Out[106…

	entry_date	order_id	customer_id	quantity	price	name_clust	entry_id	country_id	entry	total_cost
0	2018-12-01 08:26:00	3031	2150	6	339	740	891	28	белый металлический фонарь	2034
1	2018-12-01 08:26:00	3031	2150	8	275	132	1596	28	кремовая вешалка в форме сердечек Купидона	2200
2	2018-12-01 08:26:00	3031	2150	6	339	197	166	28	Вязаная грелка с флагом Союза	2034
3	2018-12-01 08:26:00	3031	2150	2	765	767	1810	28	набор 7 скворечников для бабушек	1530
4	2018-12-01 08:26:00	3031	2150	6	425	383	2585	28	стеклянный матовый держатель в форме звезды	2550

применим еще раз метод describe

In [107…

```
# Основные статистические показатели для 'price'
print(filtered_data['price'].describe())
print('')
# Основные статистические показатели для 'quantity'
print(filtered_data['quantity'].describe())
```

```
count    287428.000000
mean        256.170307
std         183.019977
min           0.000000
25%         125.000000
50%         208.000000
75%         375.000000
max         833.000000
Name: price, dtype: float64

count    287428.000000
mean          4.925637
std           4.528230
min         -12.000000
25%           1.000000
50%           3.000000
75%           8.000000
max          23.000000
Name: quantity, dtype: float64
```

min -12.000000

- нужно дополнительно еще отфильтровать quantity

In [108…

```
# Фильтрация отрицательных значений в quantity
filtered_data = filtered_data[filtered_data['quantity'] > 0]

print(f'Размер датафрейма после удаления отрицательных значений в quantity: {filtered_data.shape}')
print(filtered_data['quantity'].describe())
```

```
Размер датафрейма после удаления отрицательных значений в quantity: (287044, 10)
count    287044.000000
mean          4.938887
std           4.515071
min           1.000000
25%           1.000000
50%           3.000000
75%           8.000000
max          23.000000
Name: quantity, dtype: float64
```

2. Изучите столбцы order_id, customer_id, name_clust, entry_id и country_id.

In [109…
```
# Рассчитаем общую стоимость каждой товарной позиции
filtered_data['total_cost'] = filtered_data['quantity'] * filtered_data['price']
filtered_data.head()
```

Out[109…

	entry_date	order_id	customer_id	quantity	price	name_clust	entry_id	country_id	entry	total_cost
0	2018-12-01 08:26:00	3031	2150	6	339	740	891	28	белый металлический фонарь	2034
1	2018-12-01 08:26:00	3031	2150	8	275	132	1596	28	кремовая вешалка в форме сердечек Купидона	2200
2	2018-12-01 08:26:00	3031	2150	6	339	197	166	28	Вязаная грелка с флагом Союза	2034
3	2018-12-01 08:26:00	3031	2150	2	765	767	1810	28	набор 7 скворечников для бабушек	1530
4	2018-12-01 08:26:00	3031	2150	6	425	383	2585	28	стеклянный матовый держатель в форме звезды	2550

Изучите столбцы order_id, customer_id, name_clust, entry_id и country_id.

In [110…
```
filtered_data[['order_id','customer_id','entry_id','country_id']].describe()
```

Out[110…

	order_id	customer_id	entry_id	country_id
count	287044.000000	287044.000000	287044.000000	287044.000000
mean	26840.026055	3357.074431	1510.617020	26.966601
std	13422.453882	2544.366348	826.781953	4.577417
min	3031.000000	-1.000000	0.000000	0.000000
25%	14876.000000	-1.000000	877.000000	28.000000
50%	27490.000000	3451.000000	1538.000000	28.000000
75%	38769.000000	5502.000000	2194.000000	28.000000
max	48253.000000	7653.000000	2916.000000	29.000000

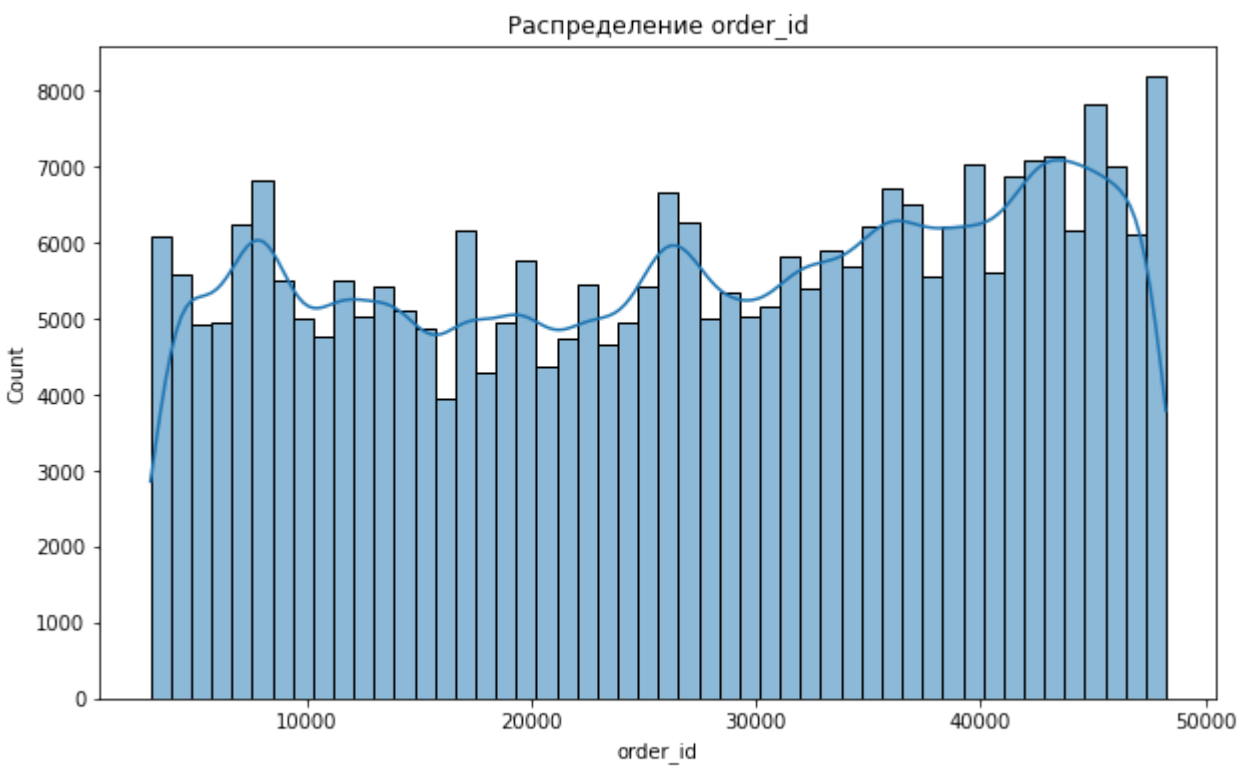
выделяются аозможные ошибки

- customer_id=-1
- entry_id=0
- country_id=0

In [111…
```
filtered_data['customer_id'].value_counts()
```

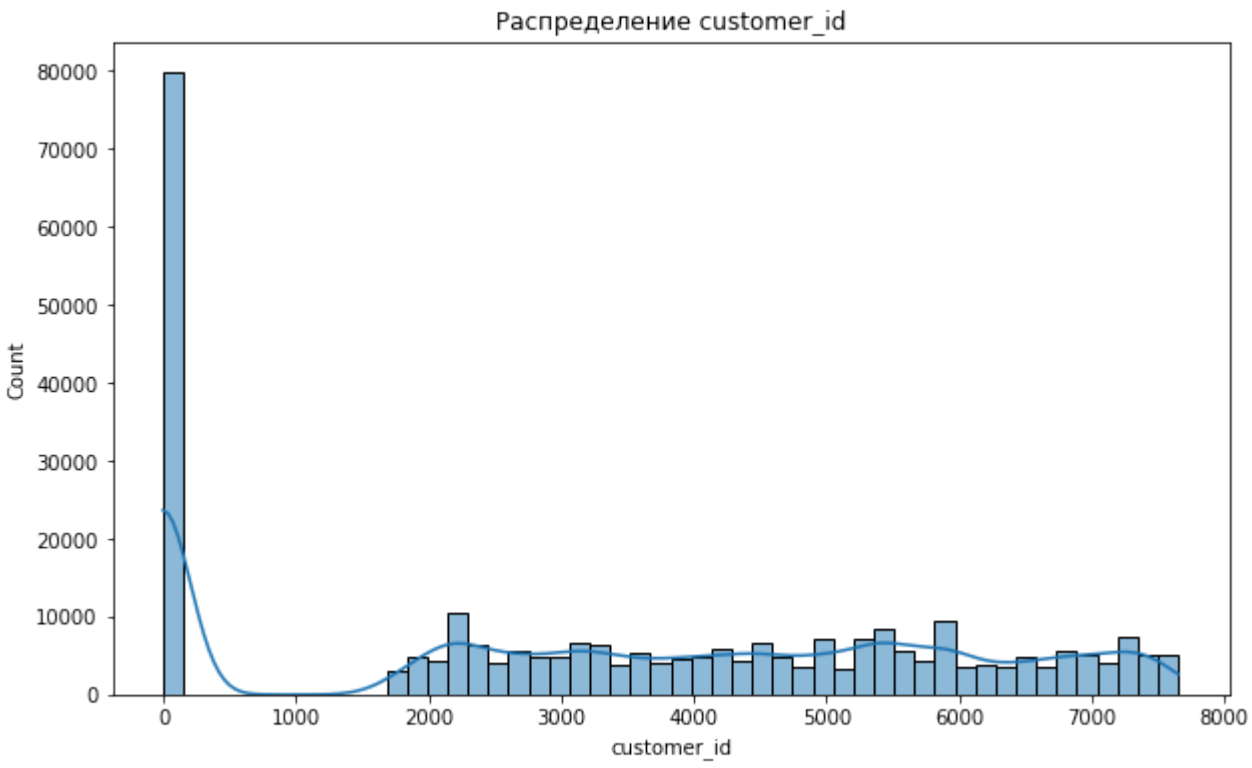
Out[111…
```
-1       79796
 2159     4899
 5904     3181
 7252     2773
 5394     1723
         ...
 4639        1
 6783        1
 3613        1
 7032        1
 3922        1
Name: customer_id, Length: 4074, dtype: int64
```

In [112…
```
import seaborn as sns
import matplotlib.pyplot as plt

# Гистограмма для 'order_id'
plt.figure(figsize=(10, 6))
sns.histplot(filtered_data['order_id'], bins=50, kde=True)
plt.title('Распределение order_id')
plt.show()
```



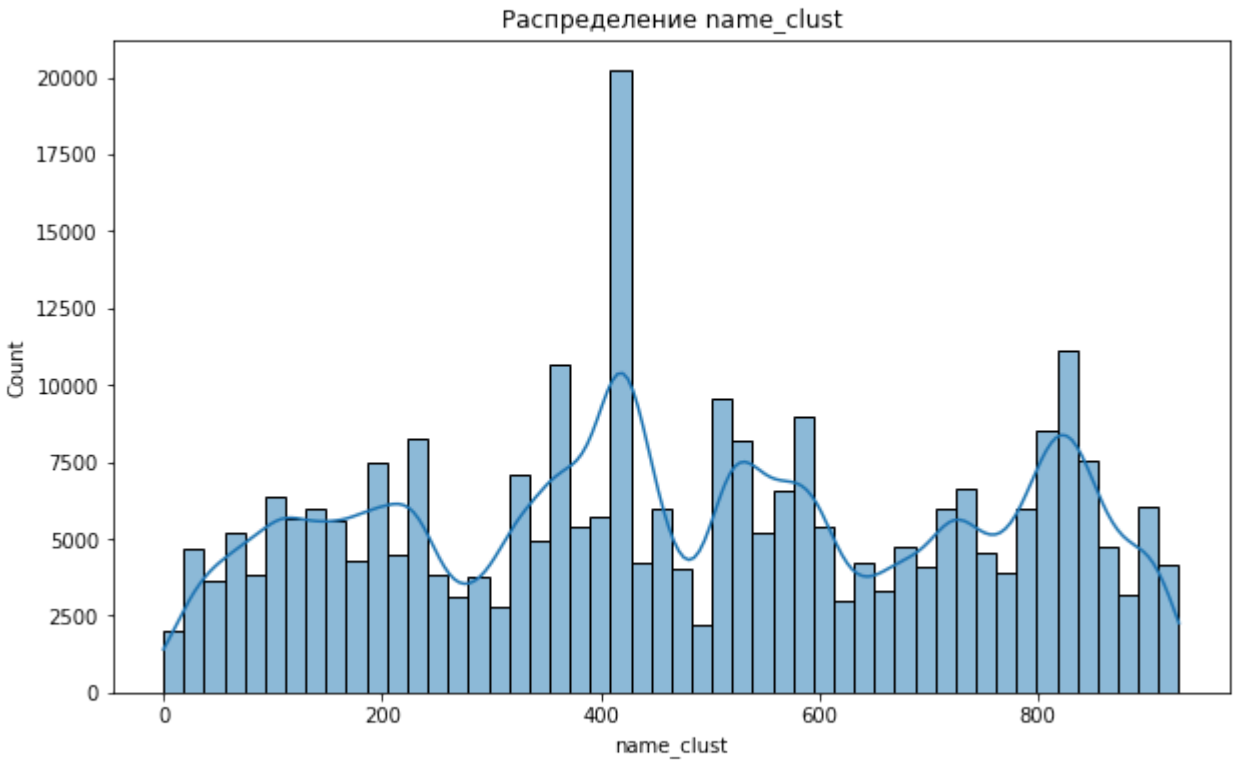
In [113…

```python
import seaborn as sns
import matplotlib.pyplot as plt

# Гистограмма для 'customer_id'
plt.figure(figsize=(10, 6))
sns.histplot(filtered_data['customer_id'], bins=50, kde=True)
plt.title('Распределение customer_id')
plt.show()
```



In [114…

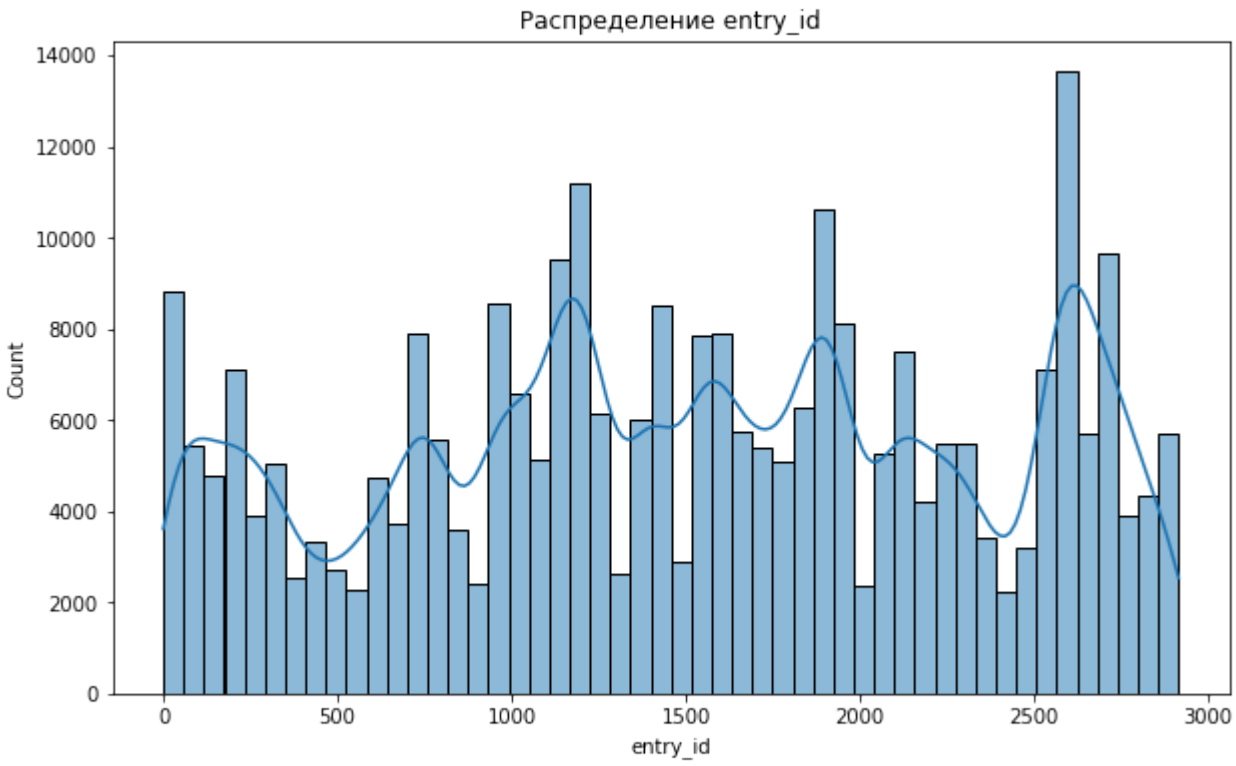
```python
import seaborn as sns
import matplotlib.pyplot as plt

# Гистограмма для 'name_clust'
plt.figure(figsize=(10, 6))
sns.histplot(filtered_data['name_clust'], bins=50, kde=True)
plt.title('Распределение name_clust')
plt.show()
```



In [115…

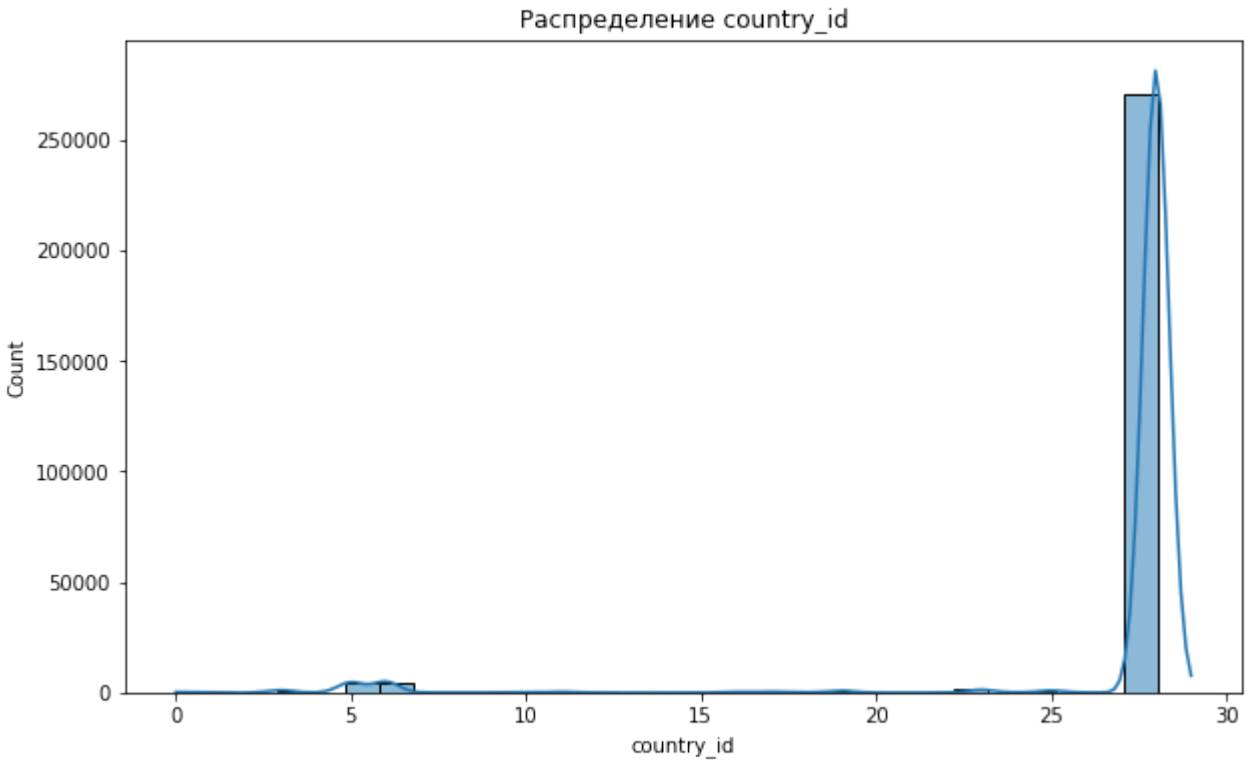
```python
import seaborn as sns
import matplotlib.pyplot as plt

# Гистограмма для 'entry_id'
```

```
plt.figure(figsize=(10, 6))
sns.histplot(filtered_data['entry_id'], bins=50, kde=True)
plt.title('Распределение entry_id')
plt.show()
```



In [116…

```
import seaborn as sns
import matplotlib.pyplot as plt


# Гистограмма для 'country_id'
plt.figure(figsize=(10, 6))
sns.histplot(filtered_data['country_id'], bins=30, kde=True)
plt.title('Распределение country_id')
plt.show()
```



- значений -1 customer_idслишком много
- решено их не трогать

3. Изучите полноту данных, анализируя время записей. Посчитайте по месяцам количество дней, в которых не было продаж. Выберите период для анализа, содержащий основную часть данных, и далее работайте только с актуальными данными.

In [117…

```
filtered_data.head()
```

Out[117…

	entry_date	order_id	customer_id	quantity	price	name_clust	entry_id	country_id	entry	total_cost
0	2018-12-01 08:26:00	3031	2150	6	339	740	891	28	белый металлический фонарь	2034
1	2018-12-01 08:26:00	3031	2150	8	275	132	1596	28	кремовая вешалка в форме сердечек Купидона	2200
2	2018-12-01 08:26:00	3031	2150	6	339	197	166	28	Вязаная грелка с флагом Союза	2034
3	2018-12-01 08:26:00	3031	2150	2	765	767	1810	28	набор 7 скворечников для бабушек	1530
4	2018-12-01 08:26:00	3031	2150	6	425	383	2585	28	стеклянный матовый держатель в форме звезды	2550

In [118…

```
# Добавим столбцы с годом и месяцем
filtered_data['year'] = filtered_data['entry_date'].dt.year
```

```
filtered_data['month'] = filtered_data['entry_date'].dt.month
filtered_data.head()
```

Out[118…

	entry_date	order_id	customer_id	quantity	price	name_clust	entry_id	country_id	entry	total_cost	year	month
0	2018-12-01 08:26:00	3031	2150	6	339	740	891	28	белый металлический фонарь	2034	2018	12
1	2018-12-01 08:26:00	3031	2150	8	275	132	1596	28	кремовая вешалка в форме сердечек Купидона	2200	2018	12
2	2018-12-01 08:26:00	3031	2150	6	339	197	166	28	Вязаная грелка с флагом Союза	2034	2018	12
3	2018-12-01 08:26:00	3031	2150	2	765	767	1810	28	набор 7 скворечников для бабушек	1530	2018	12
4	2018-12-01 08:26:00	3031	2150	6	425	383	2585	28	стеклянный матовый держатель в форме звезды	2550	2018	12

In [119…
```
filtered_data['year'].value_counts()
```

Out[119…
```
2019    265411
2018     21633
Name: year, dtype: int64
```

In [120…
```
filtered_data['month'].value_counts()
```

Out[120…
```
11    46268
12    35590
10    32300
9     26095
7     21431
5     19668
6     19518
1     18759
3     18744
8     18695
4     15753
2     14223
Name: month, dtype: int64
```

Подсчитать количество дней в каждом месяце, в которые не было продаж, на основе временных данных.

In [121…
```
# Группируем данные по годам, месяцам и дням
sales_by_day = filtered_data.groupby(['year', 'month', filtered_data['entry_date'].dt.day]).size().reset_index(name='counts')
sales_by_day
```

Out[121…

	year	month	entry_date	counts
0	2018	12	1	1600
1	2018	12	2	1013
2	2018	12	3	1091
3	2018	12	5	1496
4	2018	12	6	2084
…	…	…	…	…
300	2019	12	5	2974
301	2019	12	6	1868
302	2019	12	7	1077
303	2019	12	8	2801
304	2019	12	9	921

305 rows × 4 columns

In [122…
```
# Подсчитываем количество дней с продажами в каждом месяце
days_with_sales = sales_by_day.groupby(['year', 'month'])['counts'].count().reset_index(name='days_with_sales')
days_with_sales
```

Out[122…

	year	month	days_with_sales
0	2018	12	20
1	2019	1	24
2	2019	2	24
3	2019	3	27
4	2019	4	21
5	2019	5	25
6	2019	6	26
7	2019	7	26
8	2019	8	26
9	2019	9	26
10	2019	10	26
11	2019	11	26
12	2019	12	8

In [123…

```
# Подсчитываем общее количество дней в каждом месяце
days_in_month = sales_by_day.groupby(['year', 'month'])['entry_date'].apply(lambda x: x.max()).reset_index(name='days_in_month')
days_in_month
```

Out[123…

	year	month	days_in_month
0	2018	12	23
1	2019	1	31
2	2019	2	28
3	2019	3	31
4	2019	4	28
5	2019	5	31
6	2019	6	30
7	2019	7	31
8	2019	8	31
9	2019	9	30
10	2019	10	31
11	2019	11	30
12	2019	12	9

In [124…

```
# Объединяем данные для получения итогового результата
sales_analysis = pd.merge(days_in_month, days_with_sales, on=['year', 'month'])
sales_analysis['days_without_sales'] = sales_analysis['days_in_month'] - sales_analysis['days_with_sales']
print(sales_analysis)
```

```
    year  month  days_in_month  days_with_sales  days_without_sales
0   2018     12             23               20                   3
1   2019      1             31               24                   7
2   2019      2             28               24                   4
3   2019      3             31               27                   4
4   2019      4             28               21                   7
5   2019      5             31               25                   6
6   2019      6             30               26                   4
7   2019      7             31               26                   5
8   2019      8             31               26                   5
9   2019      9             30               26                   4
10  2019     10             31               26                   5
11  2019     11             30               26                   4
12  2019     12              9                8                   1
```

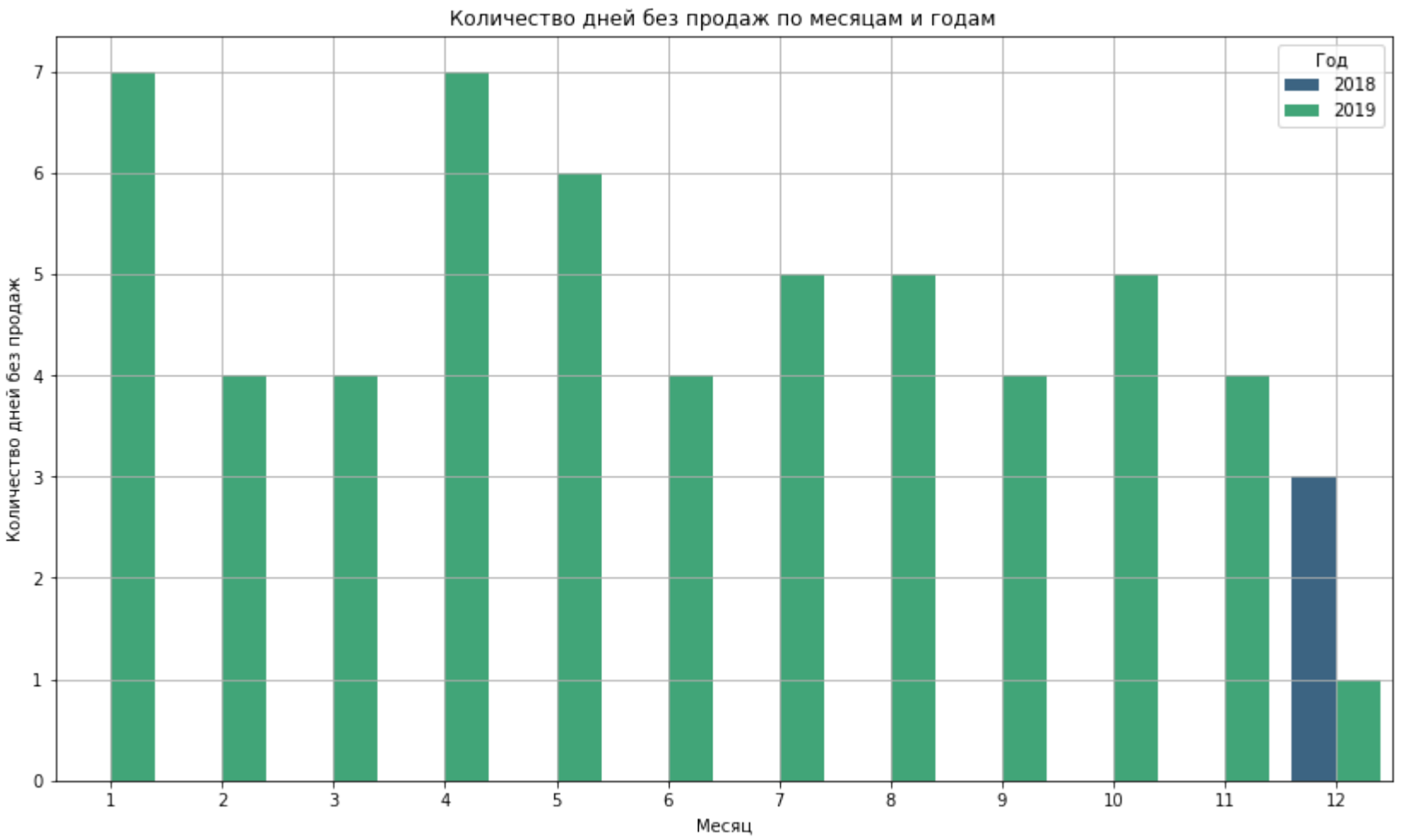
- определили количество дней без продаж для каждого месяца.
- в дальнейшем может понадобится для выявления периода например низкой активности клиента

визуализируем дни без продаж по месяцаи

In [125…

```
import matplotlib.pyplot as plt
import seaborn as sns

# Построение графика
plt.figure(figsize=(14, 8))
sns.barplot(data=sales_analysis, x='month', y='days_without_sales', hue='year', palette='viridis')

# Настройки графика
plt.title('Количество дней без продаж по месяцам и годам')
plt.xlabel('Месяц')
```

```
plt.ylabel('Количество дней без продаж')
plt.legend(title='Год')
plt.grid(True)

# Показать график
plt.show()
```



выводы по граифку

- в декабре 2018 и 2019 годов минимум дней без продаж. Объяснимо самим типом месяца для продаж
- максимум в январе и марте

Выбор периода для анализа

- ределим по годам и мемяцам наибольшее количество записей

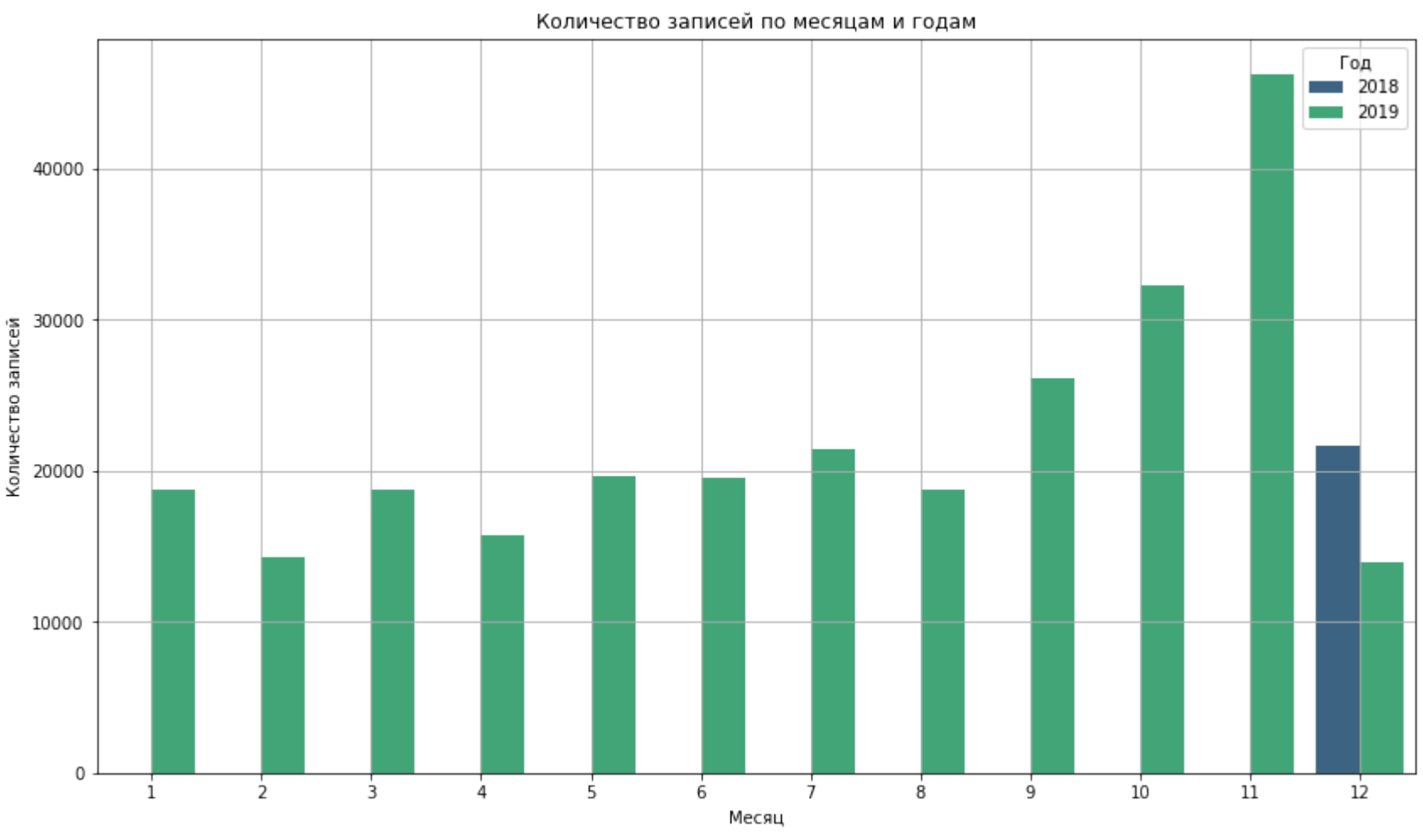
In [126…

```
# Подсчет количества записей по годам и месяцам
records_per_year_month = filtered_data.groupby(['year', 'month']).size().reset_index(name='counts')
print(records_per_year_month)
```

```
    year  month  counts
0   2018     12   21633
1   2019      1   18759
2   2019      2   14223
3   2019      3   18744
4   2019      4   15753
5   2019      5   19668
6   2019      6   19518
7   2019      7   21431
8   2019      8   18695
9   2019      9   26095
10  2019     10   32300
11  2019     11   46268
12  2019     12   13957
```

In [127…

```
import matplotlib.pyplot as plt
import seaborn as sns

# Построение графика
plt.figure(figsize=(14, 8))
sns.barplot(data=records_per_year_month, x='month', y='counts', hue='year', palette='viridis')

# Настройки графика
plt.title('Количество записей по месяцам и годам')
plt.xlabel('Месяц')
plt.ylabel('Количество записей')
plt.legend(title='Год')
plt.grid(True)

# Показать график
plt.show()
```



- наибольшее количество записей приходится на ноябрь 2019 года (46,268 записей)и октябрь 2019 года (32,300 записей)
- декабрь 2018 года (21,633 записей) и большинство месяцев 2019 года имеют значительное количество записей,приближенное к 20 тысячаи, что указывает на активные периоды продаж

Для анализа предлагается брать годовой полный цикл вместе с сезооными колебаниями - ч декабря 2019 по ноябрь 2019

In [128…
```
# Фильтрация данных за выбранный период (с декабря 2018 года по ноябрь 2019 года)
analysis_period = filtered_data[(filtered_data['year'] == 2018) & (filtered_data['month'] == 12) |
                                ((filtered_data['year'] == 2019) & (filtered_data['month'] <= 11))]
print(f'Размер исходного датафрейма для анализа: {filtered_data.shape}')
print(f'Размер датафрейма для анализа: {analysis_period.shape}')
```

```
Размер исходного датафрейма для анализа: (287044, 12)
Размер датафрейма для анализа: (273087, 12)
```

In [129…
```
analysis_period.head()
```

Out[129…

	entry_date	order_id	customer_id	quantity	price	name_clust	entry_id	country_id	entry	total_cost	year	month
0	2018-12-01 08:26:00	3031	2150	6	339	740	891	28	белый металлический фонарь	2034	2018	12
1	2018-12-01 08:26:00	3031	2150	8	275	132	1596	28	кремовая вешалка в форме сердечек Купидона	2200	2018	12
2	2018-12-01 08:26:00	3031	2150	6	339	197	166	28	Вязаная грелка с флагом Союза	2034	2018	12
3	2018-12-01 08:26:00	3031	2150	2	765	767	1810	28	набор 7 скворечников для бабушек	1530	2018	12
4	2018-12-01 08:26:00	3031	2150	6	425	383	2585	28	стеклянный матовый держатель в форме звезды	2550	2018	12

In [130…
```
analysis_period.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 273087 entries, 0 to 336642
Data columns (total 12 columns):
 #   Column       Non-Null Count   Dtype         
---  ------       --------------   -----         
 0   entry_date   273087 non-null  datetime64[ns]
 1   order_id     273087 non-null  int64         
 2   customer_id  273087 non-null  int64         
 3   quantity     273087 non-null  int64         
 4   price        273087 non-null  int64         
 5   name_clust   273087 non-null  int64         
 6   entry_id     273087 non-null  int64         
 7   country_id   273087 non-null  int64         
 8   entry        273087 non-null  object        
 9   total_cost   273087 non-null  int64         
 10  year         273087 non-null  int64         
 11  month        273087 non-null  int64         
dtypes: datetime64[ns](1), int64(10), object(1)
memory usage: 27.1+ MB
```

In [131…
```
comb_data.describe()
```

Out[131…

	order_id	customer_id	quantity	price	name_clust	entry_id	country_id
count	353367.000000	353367.000000	353367.000000	3.533670e+05	353367.000000	353367.000000	353367.000000
mean	26663.279831	3479.992538	10.218348	4.016779e+02	468.644602	1517.811349	26.741045
std	13368.784949	2549.306356	147.510432	5.084618e+03	259.160574	833.700318	4.998306
min	3031.000000	-1.000000	-9600.000000	-1.106206e+06	0.000000	0.000000	0.000000
25%	14832.000000	-1.000000	1.000000	1.250000e+02	242.000000	875.000000	28.000000
50%	27316.000000	3630.000000	3.000000	2.080000e+02	448.000000	1558.000000	28.000000
75%	38445.000000	5633.000000	10.000000	4.130000e+02	702.000000	2223.000000	28.000000
max	48253.000000	7653.000000	80995.000000	1.354133e+06	929.000000	2916.000000	29.000000

In [132…
```
filtered_data.describe()
```

Out[132…

	order_id	customer_id	quantity	price	name_clust	entry_id	country_id	total_cost	year	
count	287044.000000	287044.000000	287044.000000	287044.000000	287044.000000	287044.000000	287044.000000	287044.000000	287044.000000	287044.(
mean	26840.026055	3357.074431	4.938887	256.513005	478.076413	1510.617020	26.966601	1023.904698	2018.924635	7.5
std	13422.453882	2544.366348	4.515071	182.902203	255.922840	826.781953	4.577417	1087.447821	0.263979	3.5
min	3031.000000	-1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	2018.000000	1.(
25%	14876.000000	-1.000000	1.000000	125.000000	267.000000	877.000000	28.000000	295.000000	2019.000000	5.(
50%	27490.000000	3451.000000	3.000000	208.000000	456.000000	1538.000000	28.000000	663.000000	2019.000000	8.(
75%	38769.000000	5502.000000	8.000000	375.000000	709.000000	2194.000000	28.000000	1500.000000	2019.000000	11.(
max	48253.000000	7653.000000	23.000000	833.000000	929.000000	2916.000000	29.000000	19159.000000	2019.000000	12.(

In [133…
```
analysis_period.describe()
```

Out[133…

	order_id	customer_id	quantity	price	name_clust	entry_id	country_id	total_cost	year	
count	273087.000000	273087.000000	273087.000000	273087.000000	273087.000000	273087.000000	273087.000000	273087.000000	273087.000000	273087.(
mean	25785.911006	3370.836184	4.955651	256.725505	478.653371	1511.513455	26.951418	1027.393398	2018.920783	7.3
std	12903.763648	2540.265546	4.519364	182.571692	256.126875	826.583183	4.609106	1085.090312	0.270077	3.4
min	3031.000000	-1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	2018.000000	1.(
25%	14316.000000	-1.000000	1.000000	125.000000	267.000000	877.000000	28.000000	298.000000	2019.000000	5.(
50%	26482.000000	3473.000000	3.000000	208.000000	456.000000	1538.000000	28.000000	672.000000	2019.000000	8.(
75%	37165.000000	5502.000000	8.000000	375.000000	709.000000	2195.000000	28.000000	1500.000000	2019.000000	10.(
max	46551.000000	7653.000000	23.000000	833.000000	929.000000	2916.000000	29.000000	19159.000000	2019.000000	12.(

Выводы по шагу 2

- Были обнаружены выбросы и аномалии в столбцах price и quantity. Данные были отфилтиованы
- В индефикаторов customer_id выделяется значение -1. Возможно, это аномалия, но решил оставить
- Интререс вызывает преобладание country_id= 28- некий основной регион продаж
- Основной массив данных охватывает период с декабря 2018 по декабрь 2019:
  - Пиковые месяцы: ноябрь 2019 , октябрь 2019
  - Минимальные продажи: февраль 2019
- Количество дней без продаж варьируется по месяцам:
  - Максимум пропусков в январе и апреле 2019
  - Минимум пропусков в декабре обоих лет(повышенную активность в предновогодний период)
- Выбран период с декабря 2018 по ноябрь 2019 (полный годовой цикл)
- Данные структурированы по годам и месяцам для дальнейшего анализа

*** к содержанию**

Шаг 3. Расчёт метрик

- работаем с датафреймом analysis_period

1. Оцените по часам и дням недели количество заказов и количество уникальных покупателей. Постройте графики и сделайте вывод о наличии цикличности в покупательской активности.

In [136…
```
# Создание столбцов для часов и дней недели с использованием .loc
# Создание столбцов для часов и дней недели
analysis_period['hour'] = analysis_period['entry_date'].dt.hour
analysis_period['day_of_week'] = analysis_period['entry_date'].dt.dayofweek

analysis_period.head()
```

```
/tmp/ipykernel_31/559917193.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  analysis_period['hour'] = analysis_period['entry_date'].dt.hour
/tmp/ipykernel_31/559917193.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  analysis_period['day_of_week'] = analysis_period['entry_date'].dt.dayofweek
```

Out[136…

	entry_date	order_id	customer_id	quantity	price	name_clust	entry_id	country_id	entry	total_cost	year	month	hour	day_of_week
0	2018-12-01 08:26:00	3031	2150	6	339	740	891	28	белый металлический фонарь	2034	2018	12	8	5
1	2018-12-01 08:26:00	3031	2150	8	275	132	1596	28	кремовая вешалка в форме сердечек Купидона	2200	2018	12	8	5
2	2018-12-01 08:26:00	3031	2150	6	339	197	166	28	Вязаная грелка с флагом Союза	2034	2018	12	8	5
3	2018-12-01 08:26:00	3031	2150	2	765	767	1810	28	набор 7 скворечников для бабушек	1530	2018	12	8	5
4	2018-12-01 08:26:00	3031	2150	6	425	383	2585	28	стеклянный матовый держатель в форме звезды	2550	2018	12	8	5

In [137…
```
# Создание столбцов для часов и дней недели с использованием .loc
analysis_period.loc[:, 'hour'] = analysis_period['entry_date'].dt.hour
analysis_period.loc[:, 'day_of_week'] = analysis_period['entry_date'].dt.dayofweek

# Проверка первых строк данных после преобразования
analysis_period.head()
```

```
/opt/conda/lib/python3.9/site-packages/pandas/core/indexing.py:1676: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_column(ilocs[0], value, pi)
```

Out[137…

	entry_date	order_id	customer_id	quantity	price	name_clust	entry_id	country_id	entry	total_cost	year	month	hour	day_of_week
0	2018-12-01 08:26:00	3031	2150	6	339	740	891	28	белый металлический фонарь	2034	2018	12	8	5
1	2018-12-01 08:26:00	3031	2150	8	275	132	1596	28	кремовая вешалка в форме сердечек Купидона	2200	2018	12	8	5
2	2018-12-01 08:26:00	3031	2150	6	339	197	166	28	Вязаная грелка с флагом Союза	2034	2018	12	8	5
3	2018-12-01 08:26:00	3031	2150	2	765	767	1810	28	набор 7 скворечников для бабушек	1530	2018	12	8	5
4	2018-12-01 08:26:00	3031	2150	6	425	383	2585	28	стеклянный матовый держатель в форме звезды	2550	2018	12	8	5

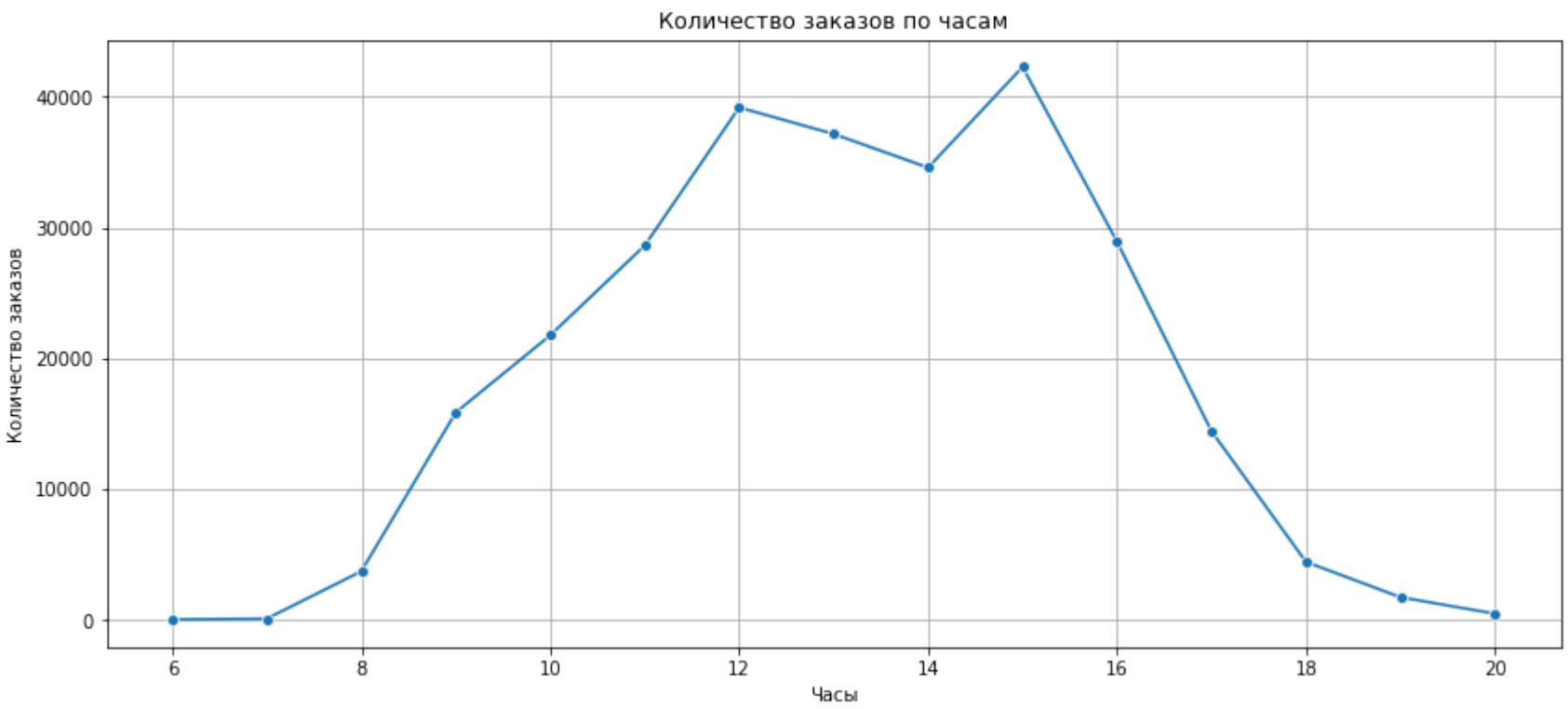
In [138…
```
# Подсчет количества заказов по часам
orders_by_hour = analysis_period.groupby('hour').size().reset_index(name='order_count')

# Подсчет количества уникальных покупателей по часам
unique_customers_by_hour = analysis_period.groupby('hour')['customer_id'].nunique().reset_index(name='unique_customers')
```

```
# Объединение данных
hourly_data = pd.merge(orders_by_hour, unique_customers_by_hour, on='hour')

print(hourly_data)
```

```
    hour  order_count  unique_customers
0      6            1                 1
1      7           66                16
2      8         3691               355
3      9        15848               762
4     10        21755              1086
5     11        28624              1134
6     12        39190              1448
7     13        37159              1402
8     14        34588              1242
9     15        42284              1106
10    16        28912               630
11    17        14409               354
12    18         4410               112
13    19         1703                78
14    20          447                14
```

In [139…

```
import matplotlib.pyplot as plt
import seaborn as sns

# График количества заказов по часам
plt.figure(figsize=(14, 6))
sns.lineplot(data=hourly_data, x='hour', y='order_count', marker='o')
plt.title('Количество заказов по часам')
plt.xlabel('Часы')
plt.ylabel('Количество заказов')
plt.grid(True)
plt.show()

# График количества уникальных покупателей по часам
plt.figure(figsize=(14, 6))
sns.lineplot(data=hourly_data, x='hour', y='unique_customers', marker='o', color='green')
plt.title('Количество уникальных покупателей по часам')
plt.xlabel('Часы')
plt.ylabel('Количество уникальных покупателей')
plt.grid(True)
plt.show()
```



Выводы по количеству заказоыв по часам

- Пиковые часы: Максимальное количество заказов с 10:00 до 15:00, с наивысшим пиком в 12:00.

- Минимальные часы: Наименьшая активность клиентов наблюдается ранним утром (6:00 - 7:00) и поздним вечером (18:00 - 20:00).

Выводы по количеству уникальных покупателей по часам

- Пиковые часы: Максимальное количество уникальных покупателей также наблюдается с 10:00 до 15:00, с пиком в 12:00.
- Минимальные часы: Минимальная активность уникальных покупателей утром и вечером схожа с активностью заказов.

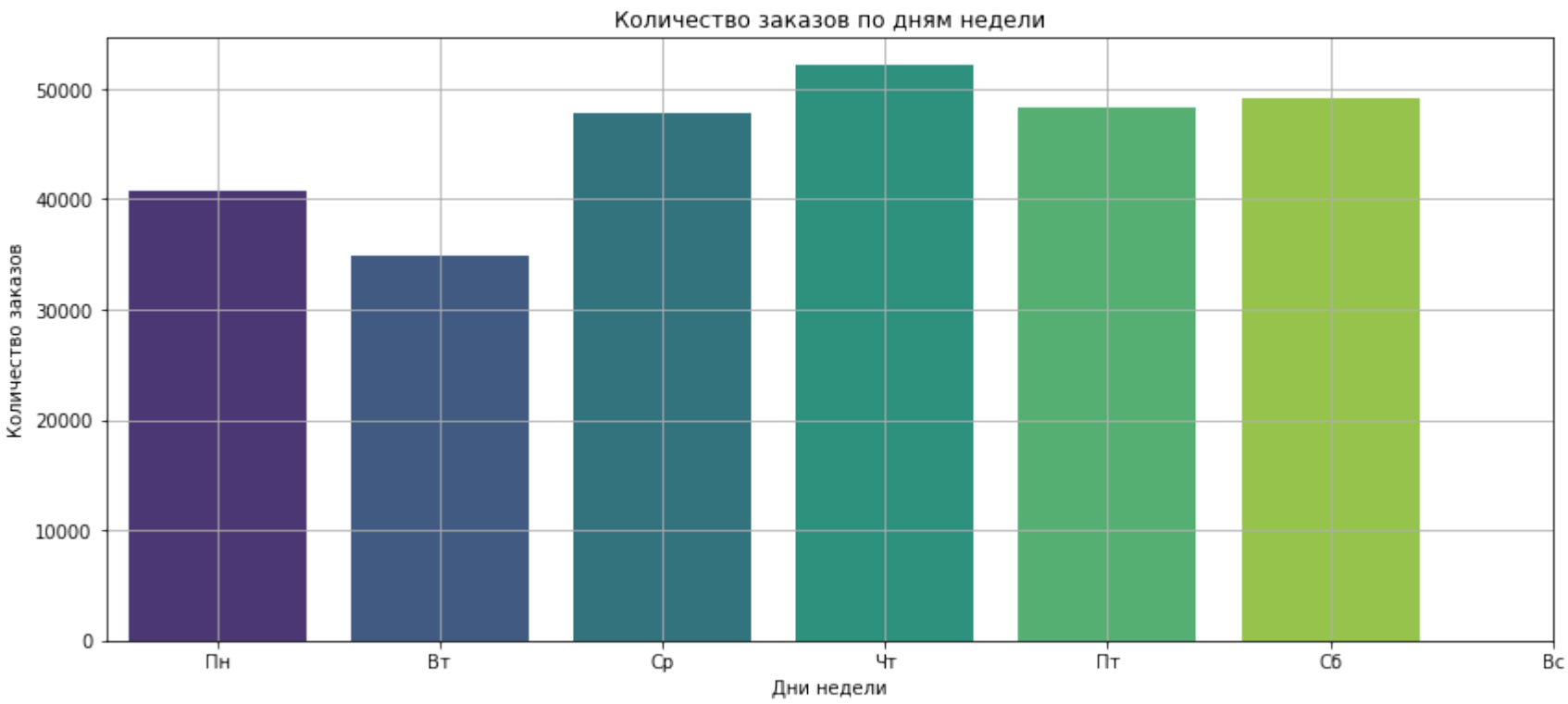
In [140…

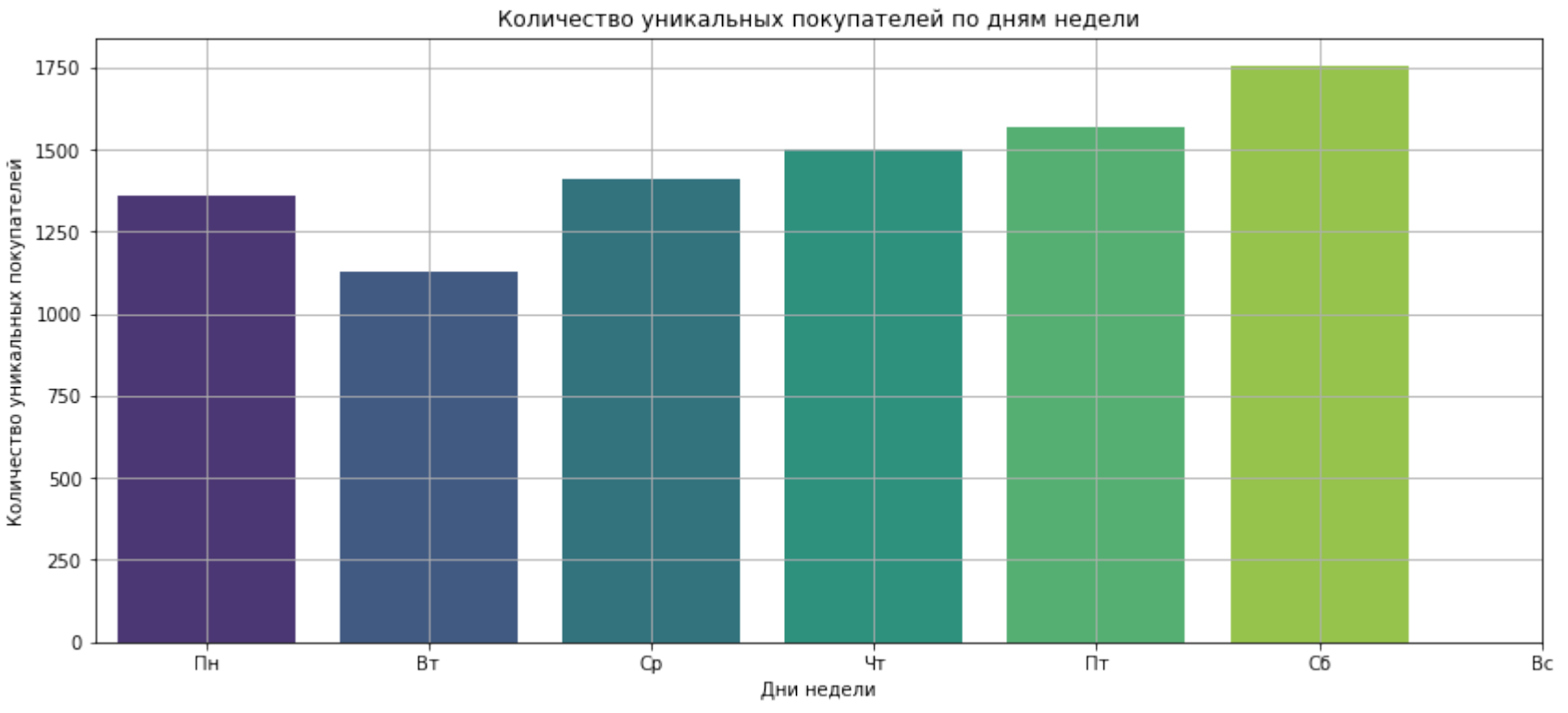
```
# Подсчет количества заказов по дням недели
orders_by_day = analysis_period.groupby('day_of_week').size().reset_index(name='order_count')

# Подсчет количества уникальных покупателей по дням недели
unique_customers_by_day = analysis_period.groupby('day_of_week')['customer_id'].nunique().reset_index(name='unique_customers')

# Объединение данных
daily_data = pd.merge(orders_by_day, unique_customers_by_day, on='day_of_week')

print(daily_data)
```

```
   day_of_week  order_count  unique_customers
0            0        40815              1357
1            2        34948              1130
2            3        47792              1412
3            4        52145              1501
4            5        48301              1570
5            6        49086              1753
```

In [141…

```
# График количества заказов по дням недели
plt.figure(figsize=(14, 6))
sns.barplot(data=daily_data, x='day_of_week', y='order_count', palette='viridis')
plt.title('Количество заказов по дням недели')
plt.xlabel('Дни недели')
plt.ylabel('Количество заказов')
plt.xticks(ticks=[0, 1, 2, 3, 4, 5, 6], labels=['Пн', 'Вт', 'Ср', 'Чт', 'Пт', 'Сб', 'Вс'])
plt.grid(True)
plt.show()

# График количества уникальных покупателей по дням недели
plt.figure(figsize=(14, 6))
sns.barplot(data=daily_data, x='day_of_week', y='unique_customers', palette='viridis')
plt.title('Количество уникальных покупателей по дням недели')
plt.xlabel('Дни недели')
plt.ylabel('Количество уникальных покупателей')
plt.xticks(ticks=[0, 1, 2, 3, 4, 5, 6], labels=['Пн', 'Вт', 'Ср', 'Чт', 'Пт', 'Сб', 'Вс'])
plt.grid(True)
plt.show()
```





Выводы по количеству заказоыв по дням недели

- Самые активные дни: Максимальная активность заказов наблюдается в четверг и субботу, с пиком в субботу
- Самые пассивные дни: Наименьшая активность заказов наблюдается во вторник .

Выводы по количеству уникальных покупателей по дням недели

- Самые активные дни: Максимальная активность уникальных покупателей также приходится на субботу, что согласуется с количеством заказов.
- Самые пассивные дни: Минимальная активность уникальных покупателей также наблюдается во вторник

Таким образом выводы о цикличности

1. Пиковые часы: с 10 до 15 с писком покупателей в 12
2. Пиковые дни - четверг и суббота. Возомжно в эти дни нужны скидки, распроджаы и так далее
3. Провалы по дням и часам- ранне утро и вечер, а по дням это вторник. То есть во вторник можно проводить какие-нибудь технические работы ибо клиентов на минимуме

2. Рассчитайте по месяцам среднюю выручку с клиента в день и количество уникальных покупателей. Сделайте вывод о наличии или отсутствии сезонности, если это возможно.

In [142…
```
# Добавление столбца с датой без времени для группировки по дням
analysis_period['date_only'] = analysis_period['entry_date'].dt.date
analysis_period.head()
```

```
/tmp/ipykernel_31/2238501940.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  analysis_period['date_only'] = analysis_period['entry_date'].dt.date
```

Out[142…

	entry_date	order_id	customer_id	quantity	price	name_clust	entry_id	country_id	entry	total_cost	year	month	hour	day_of_week	da
0	2018-12-01 08:26:00	3031	2150	6	339	740	891	28	белый металлический фонарь	2034	2018	12	8	5	2
1	2018-12-01 08:26:00	3031	2150	8	275	132	1596	28	кремовая вешалка в форме сердечек Купидона	2200	2018	12	8	5	2
2	2018-12-01 08:26:00	3031	2150	6	339	197	166	28	Вязаная грелка с флагом Союза	2034	2018	12	8	5	2
3	2018-12-01 08:26:00	3031	2150	2	765	767	1810	28	набор 7 скворечников для бабушек	1530	2018	12	8	5	2
4	2018-12-01 08:26:00	3031	2150	6	425	383	2585	28	стеклянный матовый держатель в форме звезды	2550	2018	12	8	5	2

In [143…
```
# Добавление столбца с датой без времени для группировки по дням с использованием .loc
analysis_period.loc[:, 'date_only'] = analysis_period['entry_date'].dt.date

# Проверка первых строк данных после преобразования
analysis_period.head()
```

```
/opt/conda/lib/python3.9/site-packages/pandas/core/indexing.py:1676: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_column(ilocs[0], value, pi)
```

Out[143…

	entry_date	order_id	customer_id	quantity	price	name_clust	entry_id	country_id	entry	total_cost	year	month	hour	day_of_week	da
0	2018-12-01 08:26:00	3031	2150	6	339	740	891	28	белый металлический фонарь	2034	2018	12	8	5	2
1	2018-12-01 08:26:00	3031	2150	8	275	132	1596	28	кремовая вешалка в форме сердечек Купидона	2200	2018	12	8	5	2
2	2018-12-01 08:26:00	3031	2150	6	339	197	166	28	Вязаная грелка с флагом Союза	2034	2018	12	8	5	2
3	2018-12-01 08:26:00	3031	2150	2	765	767	1810	28	набор 7 скворечников для бабушек	1530	2018	12	8	5	2
4	2018-12-01 08:26:00	3031	2150	6	425	383	2585	28	стеклянный матовый держатель в форме звезды	2550	2018	12	8	5	2

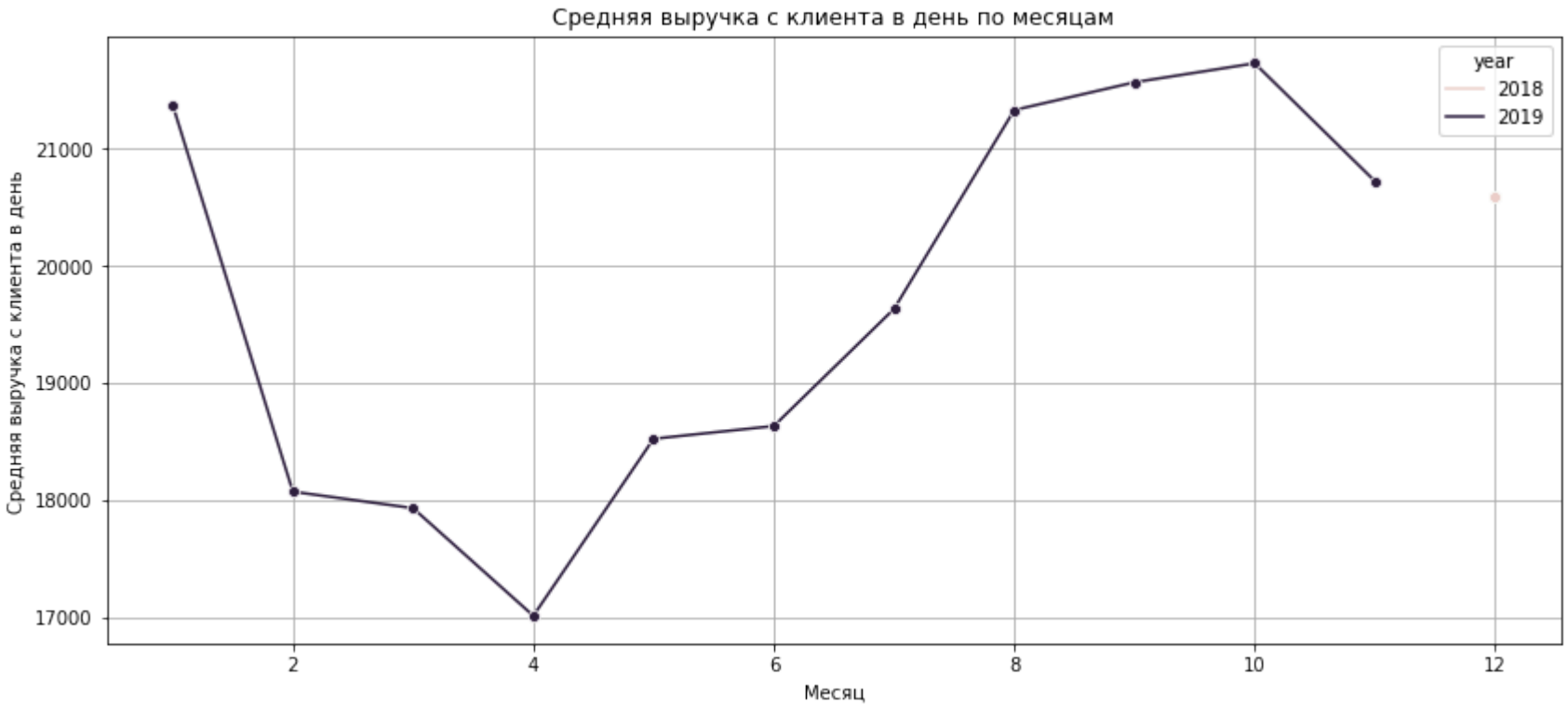
In [144…

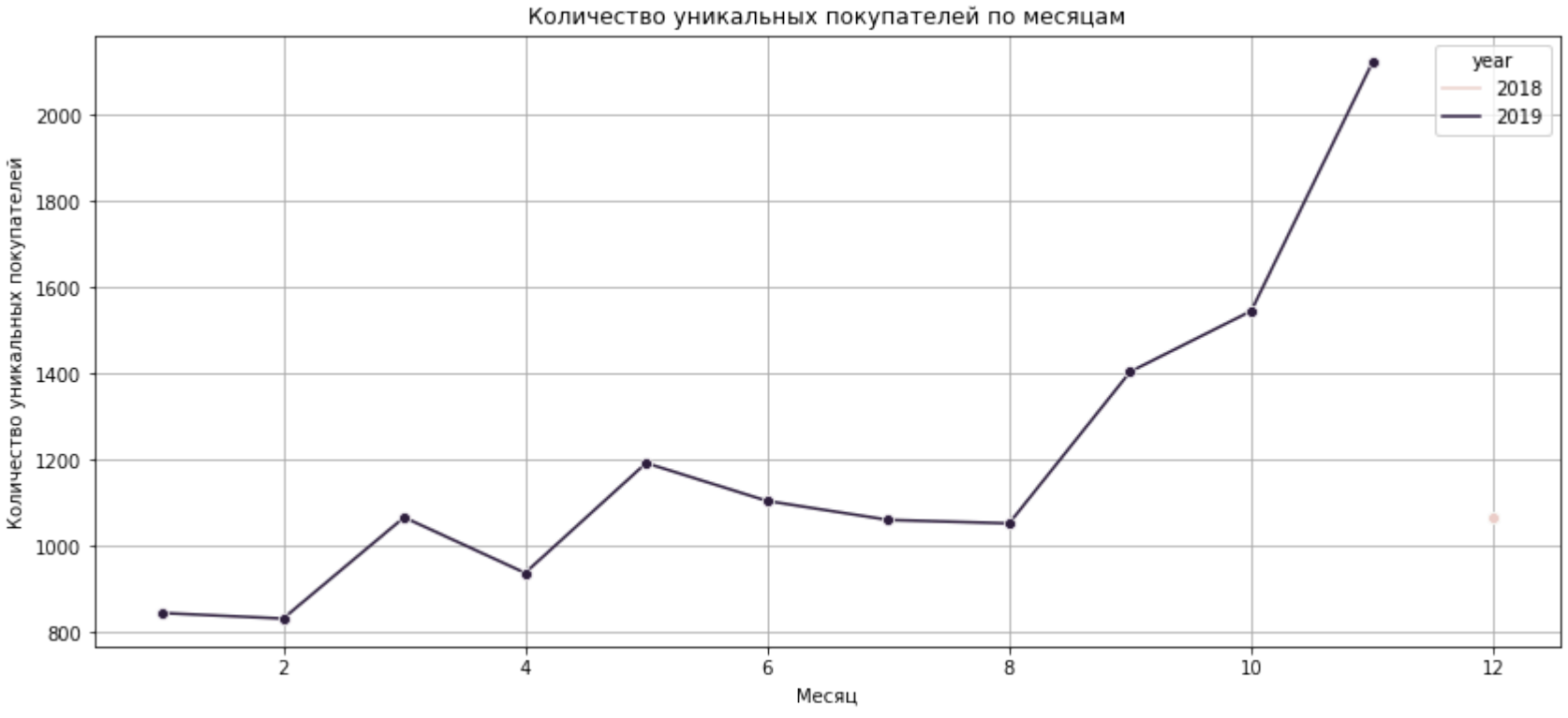
```
# Группировка данных по году, месяцу и дню
daily_revenue = analysis_period.groupby(['year', 'month', 'date_only']).agg({
    'total_cost': 'sum',  # Выручка за день
    'customer_id': 'nunique'  # Количество уникальных покупателей за день
}).reset_index()

# Рассчет средней выручки с клиента в день
daily_revenue['average_revenue_per_customer'] = daily_revenue['total_cost'] / daily_revenue['customer_id']

# Группировка данных по месяцам для рассчета среднего значения
monthly_avg_revenue = daily_revenue.groupby(['year', 'month']).agg({
    'average_revenue_per_customer': 'mean',  # Средняя выручка с клиента в день
    'customer_id': 'sum'  # Количество уникальных покупателей за месяц
}).reset_index()

print(monthly_avg_revenue)
```

```
    year  month  average_revenue_per_customer  customer_id
0   2018     12                  20595.655517         1067
1   2019      1                  21375.224355          844
2   2019      2                  18069.085049          831
3   2019      3                  17928.272286         1066
4   2019      4                  17008.525794          937
5   2019      5                  18521.069739         1192
6   2019      6                  18630.084202         1104
7   2019      7                  19630.948313         1060
8   2019      8                  21330.544616         1052
9   2019      9                  21568.078252         1404
10  2019     10                  21730.328986         1545
11  2019     11                  20723.088935         2121
```

In [145…

```
import matplotlib.pyplot as plt
import seaborn as sns

# График средней выручки с клиента в день по месяцам
plt.figure(figsize=(14, 6))
sns.lineplot(data=monthly_avg_revenue, x='month', y='average_revenue_per_customer', hue='year', marker='o')
plt.title('Средняя выручка с клиента в день по месяцам')
plt.xlabel('Месяц')
plt.ylabel('Средняя выручка с клиента в день')
plt.grid(True)
plt.show()

# График количества уникальных покупателей по месяцам
plt.figure(figsize=(14, 6))
sns.lineplot(data=monthly_avg_revenue, x='month', y='customer_id', hue='year', marker='o', color='green')
plt.title('Количество уникальных покупателей по месяцам')
plt.xlabel('Месяц')
plt.ylabel('Количество уникальных покупателей')
plt.grid(True)
plt.show()
```





Вывод о средней выручки клиента в день по месяцам:

- Высокие значения: Январь, сентябрь, октябрь и ноябрь 2019 года, что может быть связано с праздничными сезонами и подготовкой к ним.
- Низкие значения: с февраля по апрель 2019 года показывают более низкую среднюю выручку с клиента в день, возможно из-за отсутствия крупных праздников в эти месяцы.

Вывод о количестве уникальных покупателей по месяцам:

- явный тренд в 2019 году с пиквм в ноябре: Наблюдаются в январе, сентябре, октябре и ноябре, что может быть связано с праздниками и крупными распродажами.
- Сезонный спад в январе феврале

Вывод о сезонности выручки клиента:

- Сезонные пики: Наблюдаются в январе, сентябре, октябре и ноябре, что может быть связано с праздниками и крупными распродажами.
- Сезонные спады: Замечены с февраля по апрель, когда активность снижается.

Общий вывод На основании анализа графиков, можно заключить, что в данных присутствует сезонность.

- Пики активности и роста средней выручки с клиента совпадают с предпраздничными сезонами и крупными распродажами.
- Спады активности наблюдаются в менее активные месяцы, такие как январь и февраль.

3. Рассчитайте стики-фактор за второй и третий квартал 2019 года.

In [146…
```
# Фильтрация данных для второго квартала 2019 года (апрель-июнь)
Q2_2019 = analysis_period[(analysis_period['year'] == 2019) & (analysis_period['month'].isin([4, 5, 6]))]

# Фильтрация данных для третьего квартала 2019 года (июль- ентябрь)
Q3_2019 = analysis_period[(analysis_period['year'] == 2019) & (analysis_period['month'].isin([7, 8, 9]))]
```

проверим правлиность филтрации по кварталам методом describe

In [147…
```
Q2_2019['month'].describe()
```

Out[147…
```
count    54939.000000
mean         5.068531
std          0.798322
min          4.000000
25%          4.000000
50%          5.000000
75%          6.000000
max          6.000000
Name: month, dtype: float64
```

In [148…
```
Q3_2019['month'].describe()
```

Out[148…
```
count    66221.000000
mean         8.070431
std          0.844238
min          7.000000
25%          7.000000
50%          8.000000
75%          9.000000
max          9.000000
Name: month, dtype: float64
```

In [149…
```
# Количество уникальных покупателей во втором квартале 2019 года
unique_customers_Q2 = Q2_2019['customer_id'].nunique()

# Количество уникальных покупателей в третьем квартале 2019 года
unique_customers_Q3 = Q3_2019['customer_id'].nunique()

print(f'Уникальных покупателей во втором квартале 2019: {unique_customers_Q2}')
print(f'Уникальных покупателей в третьем квартале 2019: {unique_customers_Q3}')
```

```
Уникальных покупателей во втором квартале 2019: 1852
Уникальных покупателей в третьем квартале 2019: 1989
```

In [150…
```
# Список уникальных покупателей в каждом квартале
customers_Q2 = set(Q2_2019['customer_id'].unique())
customers_Q3 = set(Q3_2019['customer_id'].unique())

# Количество возвращающихся покупателей
returning_customers = len(customers_Q2 & customers_Q3)

print(f'Возвращающихся покупателей: {returning_customers}')
```

```
Возвращающихся покупателей: 1096
```

In [151…
```
# Расчет стики-фактора
stickiness_factor = returning_customers / unique_customers_Q3

print(f'Стики-фактор за второй и третий квартал 2019 года: {stickiness_factor:.2f}')
```

```
Стики-фактор за второй и третий квартал 2019 года: 0.55
```

Это свидетельствует о высокой лояльности клиентов и их склонности возвращаться для повторных покупок.

4. Составьте профиль каждого клиента, включите в профиль количество заказов, дату первого и последнего заказа, общую сумму всех заказов, среднюю цену заказа, а также другие показатели по вашему выбору.

- профиль каждого клиента включает следующие показатели:
  - Количество заказов (total_orders)
  - Дата первого заказа (first_order_date)
  - Дата последнего заказа (last_order_date)
  - Общая сумма всех заказов (total_spent)
  - Средняя сумма заказа (average_total_cost)
  - Среднее количество товаров в заказе (average_quantity)
  - Средняя цена заказа (average_order_value)

In [152…
```
# Группировка данных по клиентам
customer_profile = analysis_period.groupby('customer_id').agg({
    'order_id': 'nunique',  # Количество заказов
    'entry_date': ['min', 'max'],  # Дата первого и последнего заказа
    'total_cost': ['sum', 'mean'],  # Общая сумма всех заказов и средняя сумма заказа
    'quantity': 'mean'  # Среднее количество товаров в заказе
}).reset_index()

# Переименуем столбцы для удобства
customer_profile.columns = ['customer_id', 'total_orders', 'first_order_date', 'last_order_date', 'total_spent', 'average_total_cost', 'ave

# Рассчитаем среднюю сумму заказа
customer_profile['average_order_value'] = customer_profile['total_spent'] / customer_profile['total_orders']
customer_profile
```

Out[152...

	customer_id	total_orders	first_order_date	last_order_date	total_spent	average_total_cost	average_quantity	average_order_value
0	-1	1706	2018-12-01 14:32:00	2019-11-30 17:14:00	51626176	687.221969	2.494695	30261.533411
1	1713	2	2019-05-22 10:39:00	2019-10-12 10:23:00	42220	1835.652174	10.608696	21110.000000
2	1717	13	2019-01-06 14:14:00	2019-11-30 12:59:00	131861	252.607280	1.816092	10143.153846
3	1718	1	2019-08-05 13:35:00	2019-08-05 13:35:00	3120	1560.000000	4.000000	3120.000000
4	1719	1	2019-06-12 10:53:00	2019-06-12 10:53:00	4692	938.400000	10.400000	4692.000000
...	...	...	...	...	...	...	...	...
4034	7647	1	2019-05-19 17:47:00	2019-05-19 17:47:00	2930	1465.000000	7.000000	2930.000000
4035	7648	7	2019-02-16 12:33:00	2019-11-03 14:37:00	73399	1668.159091	7.522727	10485.571429
4036	7650	1	2019-02-02 16:01:00	2019-02-02 16:01:00	18640	1694.545455	11.272727	18640.000000
4037	7651	1	2019-11-21 09:51:00	2019-11-21 09:51:00	77977	1813.418605	8.581395	77977.000000
4038	7653	6	2018-12-07 14:57:00	2019-10-31 12:25:00	210066	2188.187500	9.208333	35011.000000

4039 rows × 8 columns

5. Разделите клиентов на возвратных и нет по признаку наличия повторных покупок, для каждой из групп на основе профилей клиентов (когда это возможно) рассчитайте средние показатели и оцените их.

- Возвратные клиенты: Клиенты, совершившие более одного заказа.
- Новые клиенты: Клиенты, совершившие только один заказ.

In [153...
```
# Разделение клиентов на возвратных и новых
returning_customers = customer_profile[customer_profile['total_orders'] > 1]
new_customers = customer_profile[customer_profile['total_orders'] == 1]
```

In [154...
```
returning_customers
```

Out[154...

	customer_id	total_orders	first_order_date	last_order_date	total_spent	average_total_cost	average_quantity	average_order_value
0	-1	1706	2018-12-01 14:32:00	2019-11-30 17:14:00	51626176	687.221969	2.494695	30261.533411
1	1713	2	2019-05-22 10:39:00	2019-10-12 10:23:00	42220	1835.652174	10.608696	21110.000000
2	1717	13	2019-01-06 14:14:00	2019-11-30 12:59:00	131861	252.607280	1.816092	10143.153846
11	1728	5	2019-04-07 09:35:00	2019-10-25 11:52:00	131816	1856.563380	9.042254	26363.200000
12	1730	2	2019-03-18 12:41:00	2019-11-01 13:57:00	24490	2449.000000	9.800000	12245.000000
...	...	...	...	...	...	...	...	...
4026	7638	9	2019-02-17 10:30:00	2019-11-04 09:07:00	257147	1823.737589	8.375887	28571.888889
4028	7640	3	2019-05-23 09:43:00	2019-10-18 15:22:00	138500	1573.863636	8.727273	46166.666667
4031	7644	2	2019-01-18 09:50:00	2019-04-08 12:33:00	34036	2431.142857	10.285714	17018.000000
4035	7648	7	2019-02-16 12:33:00	2019-11-03 14:37:00	73399	1668.159091	7.522727	10485.571429
4038	7653	6	2018-12-07 14:57:00	2019-10-31 12:25:00	210066	2188.187500	9.208333	35011.000000

2503 rows × 8 columns

In [155...
```
returning_customers['total_orders'].describe()
```

Out[155...
```
count    2503.000000
mean        6.024371
std        34.670134
min         2.000000
25%         2.000000
50%         4.000000
75%         6.000000
max      1706.000000
Name: total_orders, dtype: float64
```

In [156...
```
new_customers
```

Out[156…

	customer_id	total_orders	first_order_date	last_order_date	total_spent	average_total_cost	average_quantity	average_order_value
3	1718	1	2019-08-05 13:35:00	2019-08-05 13:35:00	3120	1560.000000	4.000000	3120.0
4	1719	1	2019-06-12 10:53:00	2019-06-12 10:53:00	4692	938.400000	10.400000	4692.0
5	1720	1	2019-03-07 09:52:00	2019-03-07 09:52:00	11300	1883.333333	5.500000	11300.0
6	1722	1	2019-09-27 11:58:00	2019-09-27 11:58:00	12690	2115.000000	6.333333	12690.0
7	1723	1	2019-10-12 15:22:00	2019-10-12 15:22:00	2110	1055.000000	8.000000	2110.0
...	...	...	...	...	...	...	...	...
4032	7645	1	2019-05-09 13:49:00	2019-05-09 13:49:00	16920	2417.142857	6.285714	16920.0
4033	7646	1	2019-04-21 13:11:00	2019-04-21 13:11:00	49750	1604.838710	7.290323	49750.0
4034	7647	1	2019-05-19 17:47:00	2019-05-19 17:47:00	2930	1465.000000	7.000000	2930.0
4036	7650	1	2019-02-02 16:01:00	2019-02-02 16:01:00	18640	1694.545455	11.272727	18640.0
4037	7651	1	2019-11-21 09:51:00	2019-11-21 09:51:00	77977	1813.418605	8.581395	77977.0

1536 rows × 8 columns

In [157…
```
new_customers['total_orders'].describe()
```

Out[157…
```
count    1536.0
mean        1.0
std         0.0
min         1.0
25%         1.0
50%         1.0
75%         1.0
max         1.0
Name: total_orders, dtype: float64
```

In [158…
```
# Средние показатели для возвратных клиентов
returning_customers_avg = returning_customers.mean()

# Средние показатели для новых клиентов
new_customers_avg = new_customers.mean()

print("Средние показатели для возвратных клиентов:")
print(returning_customers_avg)

print("\nСредние показатели для новых клиентов:")
print(new_customers_avg)
```

```
Средние показатели для возвратных клиентов:
customer_id              4690.005593
total_orders                6.024371
total_spent            103204.976029
average_total_cost       1550.863100
average_quantity            7.251183
average_order_value     15419.020862
dtype: float64

Средние показатели для новых клиентов:
customer_id              4681.986328
total_orders                1.000000
total_spent            14482.894531
average_total_cost       1529.690135
average_quantity            7.248415
average_order_value     14482.894531
dtype: float64
/tmp/ipykernel_31/2529238472.py:2: FutureWarning: DataFrame.mean and DataFrame.median with numeric_only=None will include datetime64 and dat
etime64tz columns in a future version.
  returning_customers_avg = returning_customers.mean()
/tmp/ipykernel_31/2529238472.py:5: FutureWarning: DataFrame.mean and DataFrame.median with numeric_only=None will include datetime64 and dat
etime64tz columns in a future version.
  new_customers_avg = new_customers.mean()
```

In [159…
```
# Средние показатели для возвратных клиентов
returning_customers_avg = returning_customers.mean(numeric_only=True)

# Средние показатели для новых клиентов
new_customers_avg = new_customers.mean(numeric_only=True)

print("Средние показатели для возвратных клиентов:")
print(returning_customers_avg)

print("\nСредние показатели для новых клиентов:")
print(new_customers_avg)
```

```
Средние показатели для возвратных клиентов:
customer_id              4690.005593
total_orders                6.024371
total_spent            103204.976029
average_total_cost       1550.863100
average_quantity            7.251183
average_order_value     15419.020862
dtype: float64

Средние показатели для новых клиентов:
customer_id              4681.986328
total_orders                1.000000
total_spent            14482.894531
average_total_cost       1529.690135
average_quantity            7.248415
average_order_value    14482.894531
dtype: float64
```

Выводы по оценке

- Возвратные клиенты: делают больше заказов, тратят больше денег и заказывают больше товаров. Средний чек заказов также высок, что делает их важной целевой аудиторией.
- Новые клиенты: Новые клиенты имеют меньше заказов и общая сумма заказов у них значительно ниже.
- Общее между клиентами - средняя стоимость заказа и среднее количество товаров в заказе сопоставимы
- в дальнейшим необходимо выполнить три следующих шага
    - удержание возвратных клиентов
    - привлечение новых клиентов
    - перевод новвых клиентов в картегорию возвртаных

Выводы по шагу 3

- Цикличность покупательской активности:
    - По часам:
        - Пиковые часы активности: 10:00-15:00, максимум в 12:00
        - Минимальная активность: раннее утро (6:00-7:00) и поздний вечер (18:00-20:00)
    - По дням недели:
        - Самые активные дни: четверг и суббота
    - Наименьшая активность: вторник
- Сезонность:
    - максимум: январь, сентябрь, октябрь и ноябрь 2019 года
    - Спады: февраль-апрель 2019 года
- Высокий стики-фактор

*** к содержанию**

Шаг 4. Провение RFM-сегментацию клиентов

напомним статистику столбцов от исходной таблицы до таблицы для RFM-сегментации

In [160…
```
df.describe()
```

Out[160…

	order_id	customer_id	quantity	price	name_clust	entry_id	country_id
count	353367.000000	353367.000000	353367.000000	3.533670e+05	353367.000000	353367.000000	353367.000000
mean	26663.279831	3479.992538	10.218348	4.016779e+02	468.644602	1517.811349	26.741045
std	13368.784949	2549.306356	147.510432	5.084618e+03	259.160574	833.700318	4.998306
min	3031.000000	-1.000000	-9600.000000	-1.106206e+06	0.000000	0.000000	0.000000
25%	14832.000000	-1.000000	1.000000	1.250000e+02	242.000000	875.000000	28.000000
50%	27316.000000	3630.000000	3.000000	2.080000e+02	448.000000	1558.000000	28.000000
75%	38445.000000	5633.000000	10.000000	4.130000e+02	702.000000	2223.000000	28.000000
max	48253.000000	7653.000000	80995.000000	1.354133e+06	929.000000	2916.000000	29.000000

In [161…
```
comb_data.describe()
```

Out[161…

	order_id	customer_id	quantity	price	name_clust	entry_id	country_id
count	353367.000000	353367.000000	353367.000000	3.533670e+05	353367.000000	353367.000000	353367.000000
mean	26663.279831	3479.992538	10.218348	4.016779e+02	468.644602	1517.811349	26.741045
std	13368.784949	2549.306356	147.510432	5.084618e+03	259.160574	833.700318	4.998306
min	3031.000000	-1.000000	-9600.000000	-1.106206e+06	0.000000	0.000000	0.000000
25%	14832.000000	-1.000000	1.000000	1.250000e+02	242.000000	875.000000	28.000000
50%	27316.000000	3630.000000	3.000000	2.080000e+02	448.000000	1558.000000	28.000000
75%	38445.000000	5633.000000	10.000000	4.130000e+02	702.000000	2223.000000	28.000000
max	48253.000000	7653.000000	80995.000000	1.354133e+06	929.000000	2916.000000	29.000000

In [162…
```
filtered_data.describe()
```

Out[162…

	order_id	customer_id	quantity	price	name_clust	entry_id	country_id	total_cost	year	
count	287044.000000	287044.000000	287044.000000	287044.000000	287044.000000	287044.000000	287044.000000	287044.000000	287044.000000	287044.(
mean	26840.026055	3357.074431	4.938887	256.513005	478.076413	1510.617020	26.966601	1023.904698	2018.924635	7.5
std	13422.453882	2544.366348	4.515071	182.902203	255.922840	826.781953	4.577417	1087.447821	0.263979	3.5
min	3031.000000	-1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	2018.000000	1.(
25%	14876.000000	-1.000000	1.000000	125.000000	267.000000	877.000000	28.000000	295.000000	2019.000000	5.(
50%	27490.000000	3451.000000	3.000000	208.000000	456.000000	1538.000000	28.000000	663.000000	2019.000000	8.(
75%	38769.000000	5502.000000	8.000000	375.000000	709.000000	2194.000000	28.000000	1500.000000	2019.000000	11.(
max	48253.000000	7653.000000	23.000000	833.000000	929.000000	2916.000000	29.000000	19159.000000	2019.000000	12.(

In [163…
```
analysis_period.describe()
```

Out[163…

	order_id	customer_id	quantity	price	name_clust	entry_id	country_id	total_cost	year	
count	273087.000000	273087.000000	273087.000000	273087.000000	273087.000000	273087.000000	273087.000000	273087.000000	273087.000000	273087.(
mean	25785.911006	3370.836184	4.955651	256.725505	478.653371	1511.513455	26.951418	1027.393398	2018.920783	7.3
std	12903.763648	2540.265546	4.519364	182.571692	256.126875	826.583183	4.609106	1085.090312	0.270077	3.4
min	3031.000000	-1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	2018.000000	1.(
25%	14316.000000	-1.000000	1.000000	125.000000	267.000000	877.000000	28.000000	298.000000	2019.000000	5.(
50%	26482.000000	3473.000000	3.000000	208.000000	456.000000	1538.000000	28.000000	672.000000	2019.000000	8.(
75%	37165.000000	5502.000000	8.000000	375.000000	709.000000	2195.000000	28.000000	1500.000000	2019.000000	10.(
max	46551.000000	7653.000000	23.000000	833.000000	929.000000	2916.000000	29.000000	19159.000000	2019.000000	12.(

- стоит обратить внимание на customer_id=-1(возможно некорректная запись) и total_cost=0(клиент не совершил покупку или возврат)
- предлагается убрать эти данные

для rfm исключим customer_id=-1 и total_cost=0

In [164…
```
# Исключение записей с customer_id=-1
analysis_period = analysis_period[analysis_period['customer_id'] != -1]
```

In [165…
```
# Исключение записей с total_cost=0
analysis_period = analysis_period[analysis_period['total_cost'] != 0]
```

In [166…
```
analysis_period.describe()
```

Out[166…

	order_id	customer_id	quantity	price	name_clust	entry_id	country_id	total_cost	year	n
count	197943.000000	197943.000000	197943.000000	197943.000000	197943.00000	197943.000000	197943.000000	197943.000000	197943.000000	197943.0(
mean	26658.843197	4650.300182	5.889807	236.234744	475.55229	1505.526879	26.573908	1156.603694	2018.934360	7.45
std	12639.426879	1717.558157	4.654633	171.647211	255.74546	820.489811	5.328315	1105.479569	0.247653	3.34
min	3031.000000	1713.000000	1.000000	6.000000	0.00000	1.000000	0.000000	8.000000	2018.000000	1.0(
25%	15770.000000	3117.000000	2.000000	125.000000	247.00000	875.000000	28.000000	375.000000	2019.000000	5.0(
50%	27740.000000	4689.000000	4.000000	169.000000	456.00000	1528.000000	28.000000	935.000000	2019.000000	8.0(
75%	37883.000000	6001.000000	10.000000	295.000000	706.00000	2189.000000	28.000000	1650.000000	2019.000000	10.0(
max	46551.000000	7653.000000	23.000000	829.000000	929.00000	2916.000000	29.000000	16830.000000	2019.000000	12.0(

1. Разделите клиентов на группы по методике RFM

In [167…
```
analysis_period.head(3)
```

Out[167…

	entry_date	order_id	customer_id	quantity	price	name_clust	entry_id	country_id	entry	total_cost	year	month	hour	day_of_week	da
0	2018-12-01 08:26:00	3031	2150	6	339	740	891	28	белый металлический фонарь	2034	2018	12	8	5	2
1	2018-12-01 08:26:00	3031	2150	8	275	132	1596	28	кремовая вешалка в форме сердечек Купидона	2200	2018	12	8	5	2
2	2018-12-01 08:26:00	3031	2150	6	339	197	166	28	Вязаная грелка с флагом Союза	2034	2018	12	8	5	2

In [168…
```
import pandas as pd

# Загрузим данные в датафрейм (если нужно)
# analysis_period = pd.read_csv('your_dataset.csv')  # Если данные загружаются из csv файла

# Определение самой новой даты в данных
newest_date = analysis_period['entry_date'].max()

print("Самая новая дата в данных:", newest_date)
```

```
Самая новая дата в данных: 2019-11-30 17:37:00
```

считать RFM будем на момент 2019-12-01

In [169…
```
# Установим дату анализа
analysis_date = pd.to_datetime('2019-12-01')

# Рассчитаем количество дней с момента покупки до даты анализа
analysis_period['recency'] = (analysis_date - analysis_period['entry_date']).dt.days

# Проверим, что новый столбец добавлен в датафрейм
analysis_period.head()
```

Out[169…

	entry_date	order_id	customer_id	quantity	price	name_clust	entry_id	country_id	entry	total_cost	year	month	hour	day_of_week	da
0	2018-12-01 08:26:00	3031	2150	6	339	740	891	28	белый металлический фонарь	2034	2018	12	8	5	2
1	2018-12-01 08:26:00	3031	2150	8	275	132	1596	28	кремовая вешалка в форме сердечек Купидона	2200	2018	12	8	5	2
2	2018-12-01 08:26:00	3031	2150	6	339	197	166	28	Вязаная грелка с флагом Союза	2034	2018	12	8	5	2
3	2018-12-01 08:26:00	3031	2150	2	765	767	1810	28	набор 7 скворечников для бабушек	1530	2018	12	8	5	2
4	2018-12-01 08:26:00	3031	2150	6	425	383	2585	28	стеклянный матовый держатель в форме звезды	2550	2018	12	8	5	2

In [170…
```
# Создание агрегированных данных по клиентам
rfm= analysis_period.groupby('customer_id').agg({
    'recency': 'min',  # Давность (минимальное количество дней с момента последней покупки)
    'order_id': 'nunique',  # Частота (количество заказов)
    'total_cost': 'sum'  # Стоимость (общая сумма заказов)
}).reset_index()

# Переименуем столбцы для удобства
rfm.columns = ['customer_id', 'recency', 'frequency', 'monetary']

print(rfm.head())
```

```
   customer_id  recency  frequency  monetary
0         1713       49          2     42220
1         1717        0         13    131861
2         1718      117          1      3120
3         1719      171          1      4692
4         1720      268          1     11300
```

In [171…
```
rfm.describe()
```

Out[171…

	customer_id	recency	frequency	monetary
count	4037.000000	4037.000000	4037.000000	4.037000e+03
mean	4687.445876	91.977211	3.691355	5.671083e+04
std	1715.525030	99.097991	5.761390	9.760588e+04
min	1713.000000	0.000000	1.000000	1.650000e+02
25%	3209.000000	16.000000	1.000000	1.166100e+04
50%	4686.000000	49.000000	2.000000	2.649500e+04
75%	6173.000000	147.000000	4.000000	6.338600e+04
max	7653.000000	364.000000	173.000000	2.175048e+06

In [172…

```
# Присвоение оценок по Recency, Frequency и Monetary
rfm['r'] = pd.qcut(rfm['recency'], q=[0, .33, .66, 1], labels=[3, 2, 1])
rfm['f'] = pd.cut(rfm['frequency'], bins=[0, 2, 4, rfm['frequency'].max()], labels=[1, 2, 3])
rfm['m'] = pd.qcut(rfm['monetary'], q=[0, .33, .66, 1], labels=[3, 2, 1])
```

In [173…

```
rfm.head(3)
```

Out[173…

	customer_id	recency	frequency	monetary	r	f	m
0	1713	49	2	42220	2	1	2
1	1717	0	13	131861	3	3	1
2	1718	117	1	3120	1	1	3

In [174…

```
rfm.describe()
```

Out[174…

	customer_id	recency	frequency	monetary
count	4037.000000	4037.000000	4037.000000	4.037000e+03
mean	4687.445876	91.977211	3.691355	5.671083e+04
std	1715.525030	99.097991	5.761390	9.760588e+04
min	1713.000000	0.000000	1.000000	1.650000e+02
25%	3209.000000	16.000000	1.000000	1.166100e+04
50%	4686.000000	49.000000	2.000000	2.649500e+04
75%	6173.000000	147.000000	4.000000	6.338600e+04
max	7653.000000	364.000000	173.000000	2.175048e+06

In [175…

```
# Найдем групповой RFM индекс:
rfm[['r','f','m']] = rfm[['r','f','m']].astype('str')
rfm['rfm_group'] = rfm['r'] + rfm['f'] + rfm['m']
# Найдем сумму индексов RFM:
rfm[['r','f','m']] = rfm[['r','f','m']].astype('int')
rfm['rfm_sum'] = rfm[['r','f','m']].sum(axis=1)
# Выведем на экран первые строки таблицы:
print(rfm.head(3))
```

```
   customer_id  recency  frequency  monetary  r  f  m rfm_group  rfm_sum
0         1713       49          2     42220  2  1  2       212        5
1         1717        0         13    131861  3  3  1       331        7
2         1718      117          1      3120  1  1  3       113        5
```

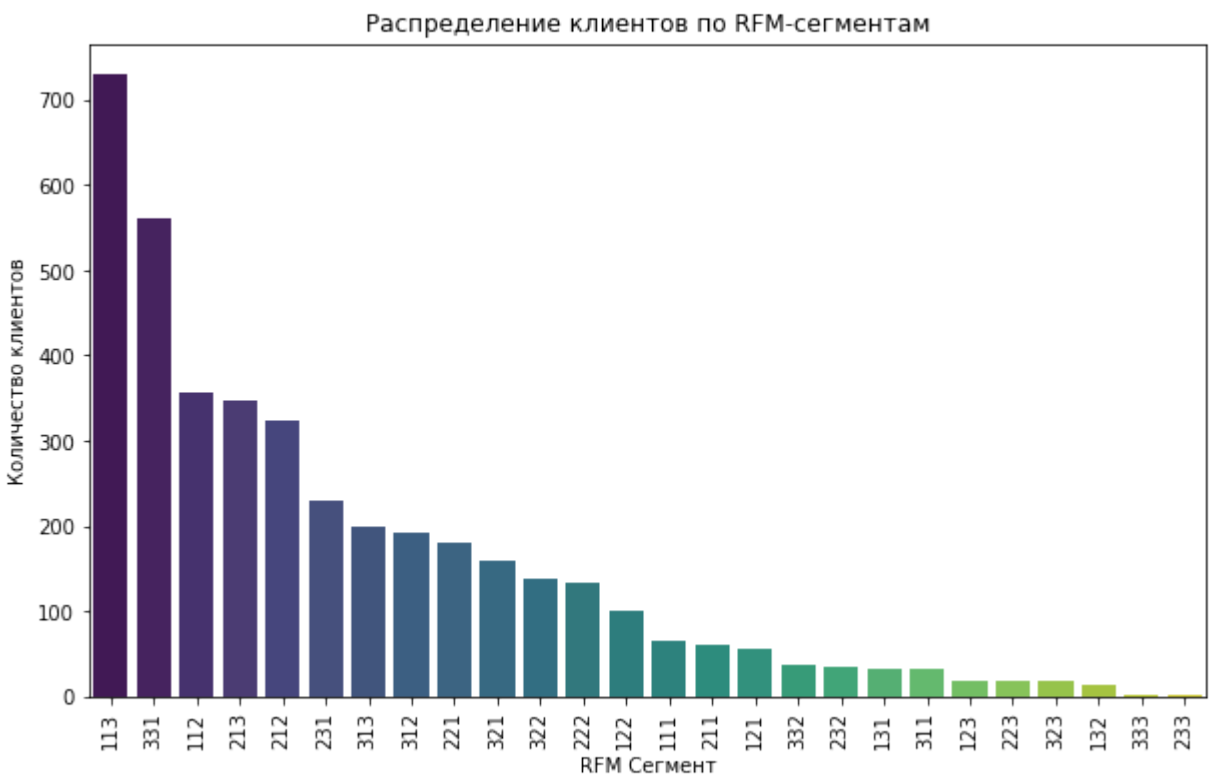
In [176…

```
# Сгруппируем данные по сегментам и подсчитаем их размер с сумму индексов:
rfm_group = rfm.groupby('rfm_group').agg({'customer_id':'nunique',
'rfm_sum':'mean'}).reset_index()
# Выведем на экран результат:
print(rfm_group.sort_values(by='customer_id', ascending=False))
```

	rfm_group	customer_id	rfm_sum
2	113	730	5
23	331	561	7
1	112	357	4
10	213	347	6
9	212	324	5
14	231	229	6
19	313	199	7
18	312	192	6
11	221	180	5
20	321	159	6
21	322	138	7
12	222	134	6
4	122	101	5
0	111	64	3
8	211	61	4
3	121	55	4
24	332	38	8
15	232	35	7
6	131	33	5
17	311	31	5
13	223	18	7
5	123	18	6
22	323	17	8
7	132	13	6
25	333	2	9
16	233	1	8

In [177…

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Распределение клиентов по сегментам
segment_distribution = rfm['rfm_group'].value_counts().reset_index()
segment_distribution.columns = ['RFM_Group', 'Count']

plt.figure(figsize=(10, 6))
sns.barplot(data=segment_distribution, x='RFM_Group', y='Count', palette='viridis')
plt.title('Распределение клиентов по RFM-сегментам')
plt.xlabel('RFM Сегмент')
plt.ylabel('Количество клиентов')
plt.xticks(rotation=90)
plt.show()
```



In [178…

```python
# Импортируем необходимые библиотеки:
import plotly.express as px
# Построим график treemap для визуализации результатов RFM сегментации:
fig = px.treemap(rfm_group,
path=['rfm_group'], # Выбираем RFM-сегменты
values='customer_id', # Устанавливаем размер - количество покупателей
color='rfm_sum', # Цвет сегмента будет определять сумма RFM
color_continuous_scale='Sunset',
title='RFM сегментация пользователей интернет-магазина «Подарочек»')
# Отобразим график:
fig.show()
```

Выводы по RFM

- Recency (давность):клиенты с наиболее недавними покупками получили высокие оценки (3), а с самыми старыми - низкие (1).
- Frequency (частота): клиенты с частыми покупками получили высокие оценки (3), а с редкими - низкие (1).
- Monetary (стоимость): клиенты с наибольшими затратами получили высокие оценки (3), а с наименьшими - низкие (1).

Топ-5 крупнейших сегментов:

1. Сегмент 113
    - Давно не совершали покупки
    - Редкие заказы
    - Высокая сумма покупок
2. Сегмент 331
    - Недавние покупки
    - Частые заказы
    - Низкая сумма покупок
3. Сегмент 112
    - Давно не совершали покупки
    - Редкие заказы
    - Средняя сумма покупок
4. Сегмент 213
    - Средняя давность покупок
    - Редкие заказы
    - Высокая сумма покупок
5. Сегмент 212
    - Средняя давность покупок
    - Редкие заказы
    - Средняя сумма покупок

Так же крайние сегменты

111- хужшие клиеты

- Давно не совершали покупки
- Редкие заказы
- Низкая сумма покупок

333- супер клиенты

- Недавние покупки
- Частые заказы
- Высокая сумма покупок

Также можно попробовать выделить следующие группы

Премиум-клиенты (VIP)

- Сегменты: 333, 332, 323
- Особенности: наиболее ценные клиенты, максимальая прибыль

Активные лояльные клиенты

- Сегменты: 331, 321, 322
- Особенности: стабильные покупатели с потенциалом роста среднего чека

Перспективные клиенты

- Сегменты: 113, 213, 313
- Особенности: готовы тратить значительные суммы, но покупают редко

Стандартные клиенты

- Сегменты: 212, 222, 221
- Особенности: стабильные показатели

Уходящие клиенты

- Сегменты: 111, 121, 131
- Особенности: высокий риск потери клиентов

Рекомендации

- развивать различные програмы лояльности
- увеличить частоту покупок
- работать сос редним чеком
- развитие персональных предложений

*** к содержанию**

Шаг 5. Проверка статистических гипотез

1. Сравните доли возвратных и невозвратных клиентов за второй и третий квартал 2019 года при помощи подходящего статистического теста.

для сравнения долей предлагается использовать Z-тест

шаги

- отфильруем по кварталам
- группируем по клинетаи и оасчёт метрики
- класификация на возвртаных и нет

In [179…

```
import pandas as pd

# Фильтрация
q2_data = analysis_period[(analysis_period['entry_date'] >= '2019-04-01') & (analysis_period['entry_date'] <= '2019-06-30')]
q3_data = analysis_period[(analysis_period['entry_date'] >= '2019-07-01') & (analysis_period['entry_date'] <= '2019-09-30')]
display(q2_data['entry_date'].describe())
display(q3_data['entry_date'].describe())
```

```
/tmp/ipykernel_31/3206537748.py:6: FutureWarning:

Treating datetime data as categorical rather than numeric in `.describe` is deprecated and will be removed in a future version of pandas. Sp
ecify `datetime_is_numeric=True` to silence this warning and adopt the future behavior now.
```

```
count                   40254
unique                   3186
top       2019-05-22 13:01:00
freq                      166
first     2019-04-01 08:22:00
last      2019-06-29 16:44:00
Name: entry_date, dtype: object
```

```
/tmp/ipykernel_31/3206537748.py:7: FutureWarning:

Treating datetime data as categorical rather than numeric in `.describe` is deprecated and will be removed in a future version of pandas. Sp
ecify `datetime_is_numeric=True` to silence this warning and adopt the future behavior now.
```

```
count                   47740
unique                   3404
top       2019-09-21 14:40:00
freq                      268
first     2019-07-01 08:16:00
last      2019-09-29 18:53:00
Name: entry_date, dtype: object
```

In [180…]
```
import pandas as pd

# Фильтрация
q2_data = analysis_period[(analysis_period['entry_date'] >= '2019-04-01') & (analysis_period['entry_date'] <= '2019-06-30')]
q3_data = analysis_period[(analysis_period['entry_date'] >= '2019-07-01') & (analysis_period['entry_date'] <= '2019-09-30')]
```

In [181…]
```
# Группировка данных по клиентам
q2_customer_orders = q2_data.groupby('customer_id')['order_id'].nunique().reset_index()
q3_customer_orders = q3_data.groupby('customer_id')['order_id'].nunique().reset_index()
display(q2_customer_orders)
display(q3_customer_orders)
```

	customer_id	order_id
0	1713	1
1	1717	4
2	1719	1
3	1728	2
4	1737	1
...	...	...
1832	7644	1
1833	7645	1
1834	7646	1
1835	7647	1
1836	7653	2

1837 rows × 2 columns

	customer_id	order_id
0	1717	2
1	1718	1
2	1722	1
3	1728	2
4	1735	2
...	...	...
1958	7638	3
1959	7640	1
1960	7642	1
1961	7648	2
1962	7653	1

1963 rows × 2 columns

In [182…]
```
import pandas as pd

# Классификация клиентов на возвратных и невозвратных
q2_customer_orders['type'] = q2_customer_orders['order_id'].apply(lambda x: 'возвратный' if x > 1 else 'невозвратный')
q3_customer_orders['type'] = q3_customer_orders['order_id'].apply(lambda x: 'возвратный' if x > 1 else 'невозвратный')
display(q2_customer_orders)
display(q3_customer_orders)
```

	customer_id	order_id	type
0	1713	1	невозвратный
1	1717	4	возвратный
2	1719	1	невозвратный
3	1728	2	возвратный
4	1737	1	невозвратный
...	...	...	...
1832	7644	1	невозвратный
1833	7645	1	невозвратный
1834	7646	1	невозвратный
1835	7647	1	невозвратный
1836	7653	2	возвратный

1837 rows × 3 columns

	customer_id	order_id	type
0	1717	2	возвратный
1	1718	1	невозвратный
2	1722	1	невозвратный
3	1728	2	возвратный
4	1735	2	возвратный
...	...	...	...
1958	7638	3	возвратный
1959	7640	1	невозвратный
1960	7642	1	невозвратный
1961	7648	2	возвратный
1962	7653	1	невозвратный

1963 rows × 3 columns

In [183…
```python
# Подсчет количества возвратных и невозвратных клиентов
q2_counts = q2_customer_orders['type'].value_counts()
q3_counts = q3_customer_orders['type'].value_counts()

# Создание таблицы для теста
contingency_table = pd.DataFrame({'Q2': q2_counts, 'Q3': q3_counts}).fillna(0)

print(contingency_table)
```

```
                Q2    Q3
невозвратный  1127  1202
возвратный     710   761
```

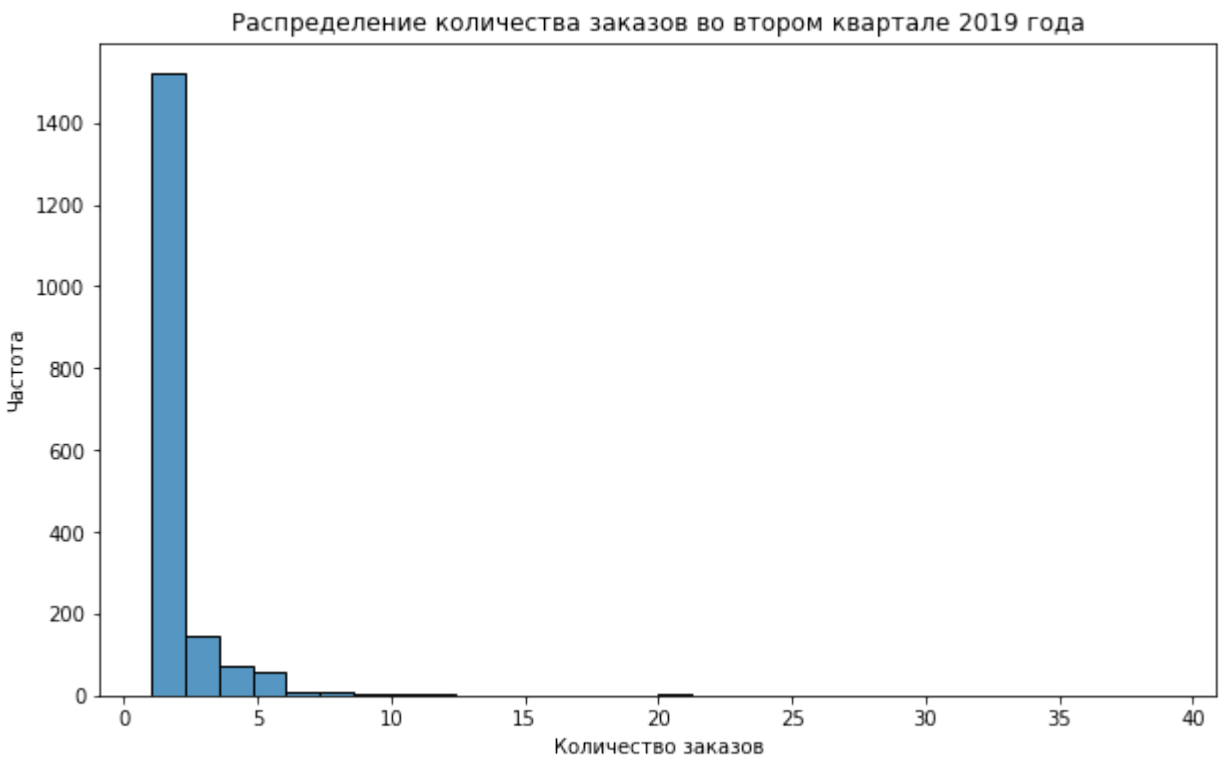
In [184…
```python
q2_customer_orders.describe()
```

Out[184…

	customer_id	order_id
count	1837.000000	1837.000000
mean	4694.522591	1.838868
std	1714.938093	1.903486
min	1713.000000	1.000000
25%	3209.000000	1.000000
50%	4749.000000	1.000000
75%	6169.000000	2.000000
max	7653.000000	39.000000

In [185…
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Данные для второго квартала
data_q2 = q2_customer_orders['order_id']

# Построение гистограммы
plt.figure(figsize=(10, 6))
sns.histplot(data_q2, bins=30)
plt.title('Распределение количества заказов во втором квартале 2019 года')
plt.xlabel('Количество заказов')
```

```
plt.ylabel('Частота')
plt.show()
```

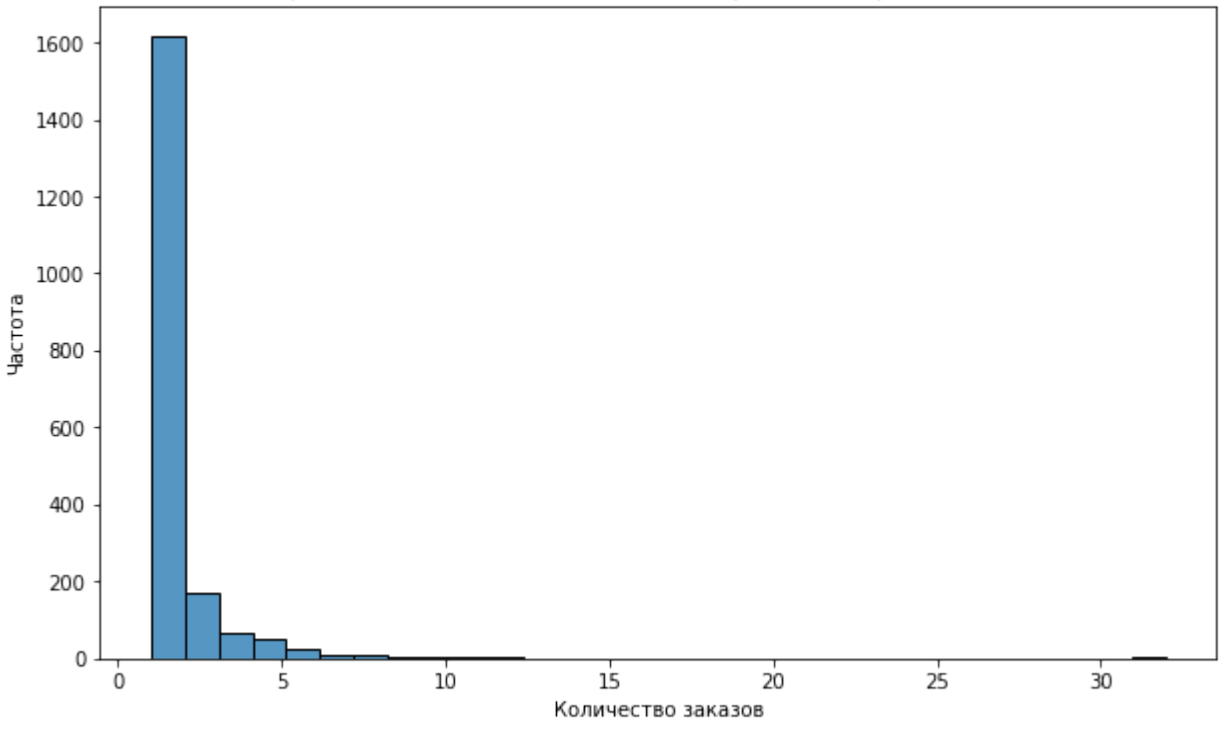


In [186…
```
q3_customer_orders.describe()
```

Out[186…

	customer_id	order_id
count	1963.000000	1963.000000
mean	4760.292919	1.830362
std	1719.118046	1.815174
min	1717.000000	1.000000
25%	3289.500000	1.000000
50%	4757.000000	1.000000
75%	6256.500000	2.000000
max	7653.000000	32.000000

In [187…
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Данные для третьего квартала
data_q3 = q3_customer_orders['order_id']

# Построение гистограммы
plt.figure(figsize=(10, 6))
sns.histplot(data_q3, bins=30)
plt.title('Распределение количества заказов в третьем квартале 2019 года')
plt.xlabel('Количество заказов')
plt.ylabel('Частота')
plt.show()
```



данные распределены схожим образом в обоих кварталах.

In [188…
```
from statsmodels.stats.proportion import proportions_ztest

# Количество возвратных и невозвратных клиентов за каждый квартал
q2_returning = q2_counts['возвратный']
q2_non_returning = q2_counts['невозвратный']
q3_returning = q3_counts['возвратный']
```

```
q3_non_returning = q3_counts['невозвратный']

# Общее количество клиентов за каждый квартал
n_q2 = q2_returning + q2_non_returning
n_q3 = q3_returning + q3_non_returning

# Количество возвратных клиентов за каждый квартал
counts = [q2_returning, q3_returning]

# Общее количество клиентов за каждый квартал
nobs = [n_q2, n_q3]

# Выполнение Z-теста для пропорций
stat, pval = proportions_ztest(counts, nobs)

print(f"Z-статистика: {stat}")
print(f"p-значение: {pval}")
```

```
Z-статистика: -0.07413413721799754
p-значение: 0.9409036524133968
```

- выводы
  - Z-статистика: -0.0741 указывает на незначительное различие между долями возвратных клиентов во втором и третьем кварталах 2019 года.
  - p-значение (p-value): 0.9409 значительно превышает уровень значимости 0.05, нет статист значимых различий,то есть не отклоняем нулевую гипотезу

2. Сравните средние чеки в странах с country_id, равному 3, 6 и 24. На основе статистических тестов сделайте вывод о том, отличаются ли средние чеки в этих странах или нет.

In [189…

```
import pandas as pd

# Фильтрация данных для стран с country_id 3, 6 и 24
country_ids = [3, 6, 24]
filtered_data_3_6_24 = analysis_period[analysis_period['country_id'].isin(country_ids)]
display(filtered_data_3_6_24['country_id'].unique())

# Рассчет среднего чека для каждой страны
country_means = filtered_data_3_6_24.groupby('country_id')['total_cost'].mean().reset_index()
print(country_means)
```

```
array([ 6,  3, 24])
   country_id   total_cost
0           3  1667.637654
1           6  1769.217194
2          24  2087.190840
```

In [190…

```
from scipy.stats import ttest_ind

# Данные для стран с country_id 3, 6 и 24
data_3 = filtered_data_3_6_24[filtered_data_3_6_24['country_id'] == 3]['total_cost']
data_6 = filtered_data_3_6_24[filtered_data_3_6_24['country_id'] == 6]['total_cost']
data_24 = filtered_data_3_6_24[filtered_data_3_6_24['country_id'] == 24]['total_cost']

# Выполнение t-тестов для сравнения средних двух независимых групп

# Сравнение стран 3 и 6
t_stat_3_6, p_value_3_6 = ttest_ind(data_3, data_6, equal_var=False)
print(f"t-тест (страна 3 и страна 6): t-статистика: {t_stat_3_6}, p-значение: {p_value_3_6}")

# Сравнение стран 3 и 24
t_stat_3_24, p_value_3_24 = ttest_ind(data_3, data_24, equal_var=False)
print(f"t-тест (страна 3 и страна 24): t-статистика: {t_stat_3_24}, p-значение: {p_value_3_24}")

# Сравнение стран 6 и 24
t_stat_6_24, p_value_6_24 = ttest_ind(data_6, data_24, equal_var=False)
print(f"t-тест (страна 6 и страна 24): t-статистика: {t_stat_6_24}, p-значение: {p_value_6_24}")
```

```
t-тест (страна 3 и страна 6): t-статистика: -3.858332084231635, p-значение: 0.0001171670936360917
t-тест (страна 3 и страна 24): t-статистика: -3.1028048714018777, p-значение: 0.0023302456942105552
t-тест (страна 6 и страна 24): t-статистика: -2.3641537837125117, p-значение: 0.019510759864204995
```

- во всех тестах p=значение меньше 0.05
- значит отклоняем нулнвую гипотеуз

На основании t-тестов можно сделать вывод, что средние чеки в странах с country_id 3, 6 и 24 статистически значимо различаются.

3. Сформулируйте собственную гипотезу и проверьте её.

Средние чеки клиентов по покупкам в выходные, выше, чем средние чеки клиентов в будни

In [195…

```
import pandas as pd

# Добавим столбец с названием дня недели
analysis_period['day_name'] = analysis_period['entry_date'].dt.day_name()
display(analysis_period['day_name'].unique())
```

```
array(['Saturday', 'Sunday', 'Monday', 'Wednesday', 'Thursday', 'Friday'],
      dtype=object)
```

In [196…
```
# Разделим данные на две группы: покупки в выходные и в будние дни
sat_sun_data = analysis_period[analysis_period['day_name'].isin(['Saturday', 'Sunday'])]
mn_fr_data = analysis_period[analysis_period['day_name'].isin(['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday'])]
display(sat_sun_data['day_name'].unique())
display(mn_fr_data['day_name'].unique())
```

```
array(['Saturday', 'Sunday'], dtype=object)
array(['Monday', 'Wednesday', 'Thursday', 'Friday'], dtype=object)
```

In [197…
```
# Рассчитаем средний чек для каждой группы
sat_sun_mean = sat_sun_data['total_cost'].mean()
mn_fr_mean = mn_fr_data['total_cost'].mean()

print(f"Средний чек в выходные дни: {sat_sun_mean}")
print(f"Средний чек в будние дни: {mn_fr_mean}")
```

```
Средний чек в выходные дни: 1238.05083522948
Средний чек в будние дни: 1109.9641129128365
```

In [198…
```
from scipy.stats import ttest_ind

# Данные для выходных и будних дней
sat_sun_total_cost = sat_sun_data['total_cost']
mn_fr_total_cost = mn_fr_data['total_cost']

# Выполнение t-теста для сравнения средних двух независимых групп
t_stat, p_value = ttest_ind(sat_sun_total_cost, mn_fr_total_cost, equal_var=False)

print(f"t-статистика: {t_stat}")
print(f"p-значение: {p_value}")
```

```
t-статистика: 24.551589490390768
p-значение: 7.792396831790474e-133
```

- отклоняем нулевую гипотезу
- средние чеки в выходные дни статистически значимо отличаются от средних чеков в будние дни

*** к содержанию**

Шаг 6. Выводы по проекту

Выводы

Шаг 1

- Были успешно подготовлены данные для дальнейшего анализа:
  - качественная предобработка, устранены дубликаты и пропуски, трансформированы типы данных

Шаг 2

- Проведенная предобработка позволила очистить данные от аномалий и выбросов, структурировать информацию по годам и месяцам
- Выявлены закономерности продаж, определены периоды максимальной и минимальной активности

Шаг 3

- магазин имеет устойчивую клиентскую базу с выраженной сезонностью продаж
- наиболее активными периодами являются предпраздничные месяцы, особенно январь, сентябрь, октябрь и ноябрь
- выскоий стики фактор показатель успешности в удержании клиентов

Покупательская активность:

- Пиковые часы продаж: 10:00-15:00, максимум в 12:00
- Самые активные дни недели: четверг и суббота
- Сезонные пики продаж: январь, сентябрь, октябрь, ноябрь

Шаг 4

- RFM-анализ показал значительную дифференциацию клиентской базы интернет-магазина "Подарочек"
- Наиболее перспективными являются VIP-клиенты и активные лояльные покупатели.

необходимы персональные предложения, увеличение частоты покупок и среднего чека, а также на возврате клиентов, которые давно не совершали покупки.

RFM-сегментация:

- Выделены 5 основных сегментов клиентов:
  - Премиум-клиенты (VIP)
  - Активные лояльные клиенты

    - Перспективные клиенты
    - Стандартные клиенты
    - Уходящие клиенты

Шаг 5

- Проведенный статистический анализ данных интернет-магазина «Подарочек» выявил значимые различия в покупательском поведении:
    - Стабильность возвратности клиентов между кварталами
    - Существенные различия средних чеков в разных странах
    - Повышенная активность покупок в выходные дни с более высокими тратами

плюсы данного бизнеса

- устойчивая клинтбаза
- сами клиенты достатчно лояльны
- есть выраженные сезонности продаж
- стабильный хороший средний чек

Необходимо в дальнейшим

- нужна программа лояльности для перевода новых клиентов в возвратных, ибо возвртаные клиенты золотая ждиа для любого бизнеса
- привлекать клиентов в периоды спада за счет спеицальных программ
- обратить внимание на вявленную сезонность, нужны скидки в это время

*** к содержанию**
