Аналитика в Яндекс.Афише

Описание проекта

Вас пригласили на стажировку в отдел аналитики Яндекс.Афиши. Первое задание: помочь маркетологам оптимизировать маркетинговые затраты. У вас в распоряжении есть данные от Яндекс.Афиши с июня 2017 по конец мая 2018 года: лог сервера с данными о посещениях сайта Яндекс.Афиши,

- выгрузка всех заказов за этот период,
- статистика рекламных расходов.

Инструкция по выполнению проекта:

Шаг 1. Загрузите данные и подготовьте их к анализу

Загрузите данные о визитах, заказах и расходах в переменные. Оптимизируйте данные для анализа. Убедитесь, что тип данных в каждой колонке — правильный. Путь к файлам:

- /datasets/visits_log.csv.
- /datasets/orders_log.csv.
- /datasets/costs.csv.

Шаг 2. Постройте отчёты и посчитайте метрики

- Продукт
  - Сколько людей пользуются в день, неделю, месяц?
  - Сколько сессий в день?
  - Сколько длится одна сессия?
  - Как часто люди возвращаются?
- Продажи
  - Когда люди начинают покупать?
  - Сколько раз покупают за период?
  - Какой средний чек?
  - Сколько денег приносят? (LTV)
- Маркетинг
  - Сколько денег потратили? Всего / на каждый источник / по времени
  - Сколько стоило привлечение одного покупателя из каждого источника?
  - На сколько окупились расходы? (ROI)

Отобразите на графиках, как эти метрики отличаются по устройствам и по рекламным источникам? Как они меняются во времени?

Шаг 3. Напишите вывод: порекомендуйте маркетологам, куда и сколько им стоит вкладывать денег?

Какие источники/платформы вы бы порекомендовали? Объясните свой выбор: на какие метрики вы ориентируетесь? Почему? Какие выводы вы сделали, узнав значение метрик?

Оформление: Задание выполните в Jupyter Notebook. Программный код заполните в ячейках типа code, текстовые пояснения — в ячейках типа markdown. Примените форматирование и заголовки.

Описание данных

- Таблица visits (лог сервера с информацией о посещениях сайта):
- Uid — уникальный идентификатор пользователя
- Device — категория устройства пользователя
- Start Ts — дата и время начала сессии
- End Ts — дата и время окончания сессии
- Source Id — идентификатор рекламного источника, из которого пришел пользователь
- Таблица orders (информация о заказах):
- Uid — уникальный id пользователя, который сделал заказ
- Buy Ts — дата и время заказа
- Revenue — выручка Яндекс.Афиши с этого заказа
- Таблица costs (информация о затратах на маркетинг):
- source_id — идентификатор рекламного источника

- source_id — идентификатор рекламного источника
- dt — дата
- costs — затраты на этот рекламный источник в этот день

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt
import plotly.graph_objects as go
import plotly.express as px
from termcolor import colored
from IPython.display import Image
from IPython.core.display import HTML
from pathlib import Path
import matplotlib.dates as mdates
```

Шаг 1. Загрузите данные и подготовьте их к анализу

visits

In [2]:

```
visits = pd.read_csv('/datasets/visits_log.csv')
orders = pd.read_csv('/datasets/orders_log.csv')
costs = pd.read_csv('/datasets/costs.csv')
```

смотрим на данные и при необходимости будем менять типы и названия столбцов

In [3]:

```
visits.head()
```

Out[3]:

	Device	End Ts	Source Id	Start Ts	Uid
0	touch	2017-12-20 17:38:00	4	2017-12-20 17:20:00	16879256277535980062
1	desktop	2018-02-19 17:21:00	2	2018-02-19 16:53:00	104060357244891740
2	touch	2017-07-01 01:54:00	5	2017-07-01 01:54:00	7459035603376831527
3	desktop	2018-05-20 11:23:00	9	2018-05-20 10:59:00	16174680259334210214
4	desktop	2017-12-27 14:06:00	3	2017-12-27 14:06:00	9969694820036681168

In [4]:

```
visits.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 359400 entries, 0 to 359399
Data columns (total 5 columns):
Device       359400 non-null object
End Ts       359400 non-null object
Source Id    359400 non-null int64
Start Ts     359400 non-null object
Uid          359400 non-null uint64
dtypes: int64(1), object(3), uint64(1)
memory usage: 13.7+ MB
```

In [5]:

```
visits.columns = ['device', 'end_ts','source_id','start_ts','uid']
visits['end_ts'] = visits['end_ts'].astype('datetime64')
visits['start_ts'] = visits['start_ts'].astype('datetime64')
visits.head()
```

Out[5]:

	device	end_ts	source_id	start_ts	uid

	device	end_ts	source_id	start_ts	uid
0	touch	2017-12-20 17:38:00	4	2017-12-20 17:20:00	16879256277535980062
1	desktop	2018-02-19 17:21:00	2	2018-02-19 16:53:00	10406035724489174O
2	touch	2017-07-01 01:54:00	5	2017-07-01 01:54:00	7459035603376831527
3	desktop	2018-05-20 11:23:00	9	2018-05-20 10:59:00	16174680259334210214
4	desktop	2017-12-27 14:06:00	3	2017-12-27 14:06:00	9969694820036681168

In [6]:

```
visits.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 359400 entries, 0 to 359399
Data columns (total 5 columns):
device       359400 non-null object
end_ts       359400 non-null datetime64[ns]
source_id    359400 non-null int64
start_ts     359400 non-null datetime64[ns]
uid          359400 non-null uint64
dtypes: datetime64[ns](2), int64(1), object(1), uint64(1)
memory usage: 13.7+ MB
```

orders

In [7]:

```
orders.sample(5)
```

Out[7]:

	Buy Ts	Revenue	Uid
30364	2018-01-21 18:58:00	2.14	7397557001469671030
35104	2018-02-15 21:29:00	4.43	7781780505070829705
20873	2017-11-29 10:48:00	6.11	9762375740284072194
47809	2018-05-18 10:02:00	0.61	17462164678248837909
9947	2017-10-01 13:04:00	1.47	4470413981559720404

In [8]:

```
orders.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50415 entries, 0 to 50414
Data columns (total 3 columns):
Buy Ts     50415 non-null object
Revenue    50415 non-null float64
Uid        50415 non-null uint64
dtypes: float64(1), object(1), uint64(1)
memory usage: 1.2+ MB
```

In [9]:

```
orders.columns = ['buy_ts', 'revenue','uid']
orders['buy_ts'] = orders['buy_ts'].astype('datetime64')
```

In [10]:

```
orders.head()
```

Out[10]:

	buy_ts	revenue	uid
0	2017-06-01 00:10:00	17.00	10329302124590727494
1	2017-06-01 00:25:00	0.55	11627257723692907447
2	2017-06-01 00:27:00	0.37	17903680561304213844
3	2017-06-01 00:29:00	0.55	16109239769442553005

	buy_ts	revenue	uid
4	2017-06-01 07:58:00	0.37	14200605875248379450

In [11]:

```
orders.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50415 entries, 0 to 50414
Data columns (total 3 columns):
buy_ts     50415 non-null datetime64[ns]
revenue    50415 non-null float64
uid        50415 non-null uint64
dtypes: datetime64[ns](1), float64(1), uint64(1)
memory usage: 1.2 MB
```

costs

In [12]:

```
costs.head()
```

Out[12]:

	source_id	dt	costs
0	1	2017-06-01	75.20
1	1	2017-06-02	62.25
2	1	2017-06-03	36.53
3	1	2017-06-04	55.00
4	1	2017-06-05	57.08

In [13]:

```
costs.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2542 entries, 0 to 2541
Data columns (total 3 columns):
source_id    2542 non-null int64
dt           2542 non-null object
costs        2542 non-null float64
dtypes: float64(1), int64(1), object(1)
memory usage: 59.7+ KB
```

In [14]:

```
costs['dt'] = costs['dt'].astype('datetime64')
```

In [15]:

```
costs.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2542 entries, 0 to 2541
Data columns (total 3 columns):
source_id    2542 non-null int64
dt           2542 non-null datetime64[ns]
costs        2542 non-null float64
dtypes: datetime64[ns](1), float64(1), int64(1)
memory usage: 59.7 KB
```

Посмотрим более детально

In [16]:

```
visits.describe()
```

Out[16]:

	source_id	uid
count	359400.000000	3.594000e+05
mean	3.750515	9.202557e+18
std	1.917116	5.298433e+18
min	1.000000	1.186350e+13
25%	3.000000	4.613407e+18
50%	4.000000	9.227413e+18
75%	5.000000	1.372824e+19
max	10.000000	1.844668e+19

In [17]:

```
orders.describe()
```

Out[17]:

	revenue	uid
count	50415.000000	5.041500e+04
mean	4.999647	9.098161e+18
std	21.818359	5.285742e+18
min	0.000000	3.135781e+14
25%	1.220000	4.533567e+18
50%	2.500000	9.102274e+18
75%	4.890000	1.368290e+19
max	2633.280000	1.844617e+19

In [18]:

```
costs.describe()
```

Out[18]:

	source_id	costs
count	2542.000000	2542.000000
mean	4.857199	129.477427
std	3.181581	156.296628
min	1.000000	0.540000
25%	2.000000	21.945000
50%	4.000000	77.295000
75%	9.000000	170.065000
max	10.000000	1788.280000

На первый взгляд все хорошо

In [19]:

```
visits.duplicated().value_counts()
```

Out[19]:

```
False    359400
dtype: int64
```

In [20]:

```
orders.duplicated().value_counts()
```

Out[20]:

```
False    50415
dtype: int64
```

In [21]:

```
costs.duplicated().value_counts()
```

Out[21]:

```
False    2542
dtype: int64
```

In [22]:

```
v = visits.copy()
o = orders.copy()
```

Все норм

> **Комментарии от ревьюера:** Здесь всё хорошо. Ты корректно выгрузил данные и посмотрел их содержимое. Провел первичную проверку на дубликаты и пропуски. Теперь можно приступать к анализу

Шаг 2. Постройте отчёты и посчитайте метрики

- Продукт
  - Сколько людей пользуются в день, неделю, месяц?
  - Сколько сессий в день?
  - Сколько длится одна сессия?
  - Как часто люди возвращаются?

Сколько людей пользуются в день, неделю, месяц?

In [23]:

```
visits['session_year'] = visits['start_ts'].dt.year
visits['session_month'] = visits['start_ts'].dt.month
visits['session_week'] = visits['start_ts'].dt.week
visits['session_date'] = visits['start_ts'].dt.date
```

In [24]:

```
dau_total = visits.groupby('session_date').agg({'uid': 'nunique'}).mean()
wau_total = visits.groupby(['session_year', 'session_week']).agg({'uid': 'nunique'}).mean()
mau_total = visits.groupby(['session_year', 'session_month']).agg({'uid': 'nunique'}).mean()
```

In [25]:

```
print('в день пользуются {} людей '.format(int(dau_total)))
print('в неделю пользуются {} людей '.format(int(wau_total)))
print('в месяц пользуются {} людей '.format(int(mau_total)))
```

```
в день пользуются 907 людей
в неделю пользуются 5716 людей
в месяц пользуются 23228 людей
```

In [26]:

```
dau_total_gr = visits.groupby('session_date').agg({'uid': 'nunique'})
wau_total_gr = visits.groupby(['session_year', 'session_week']).agg({'uid': 'nunique'})
mau_total_gr = visits.groupby(['session_year', 'session_month']).agg({'uid': 'nunique'})
```

In [27]:

```
ax_dau = dau_total_gr.plot()
ax_dau.set_title('Зависимость посещения по дням')
ax_dau.set_xlabel('Дата')
ax_dau.set_ylabel('Посещения')
```

Out[27]:

Text(0, 0.5, 'Посещения')

In [28]:

```
ax_wau = wau_total_gr.plot()
ax_wau.set_title('Зависимость посещения по неделям')
ax_wau.set_xlabel('Дата')
ax_wau.set_ylabel('Посещения')
```

Out[28]:

Text(0, 0.5, 'Посещения')

In [29]:

```
ax_mau = mau_total_gr.plot()
ax_mau.set_title('Зависимость посещения по месяцам')
ax_mau.set_xlabel('Дата')
ax_mau.set_ylabel('Посещения')
```

Out[29]:

Text(0, 0.5, 'Посещения')

Комментарии от ревьюера v3: Отлично. Так гораздо лучше

Сколько сессий в день?

In [30]:

```
day_visits = visits.groupby(['start_ts']).agg({'uid':'count'})
print('В день в среднем проходит {} сессий'.format(int(day_visits.mean()[0])))
```

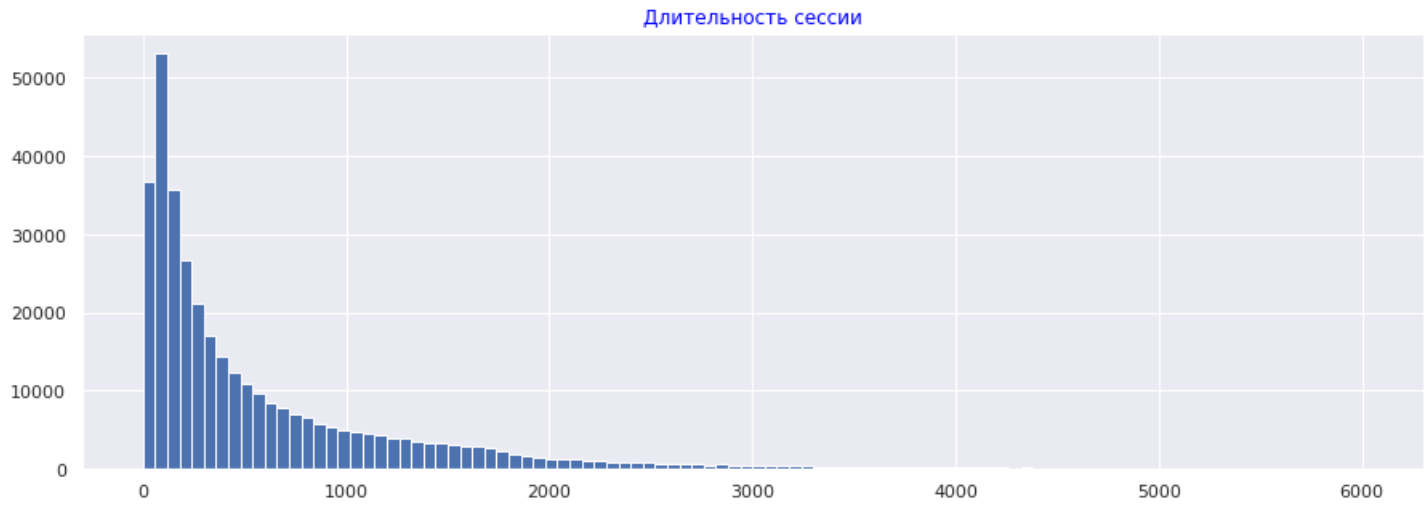
В день в среднем проходит 1 сессий

Комментарии от ревьюера: Да, примерно где то так. То что ты рассчитал количество сессий на уникального пользователя - верное решение

Сколько длится одна сессия?

In [31]:

```
visits['duration'] = (visits['end_ts'] - visits['start_ts']).dt.seconds
sns.set()
visits['duration'].hist(bins= 100, range = [0,6000],figsize = (15,5)).set_title('Длительность сессии', color = 'blue');
```



In [32]:

```
visits['duration'].mode()
```

Out[32]:

```
0    60
dtype: int64
```

In [33]:

```
visits['duration'].describe()
```

Out[33]:

```
count    359400.000000
mean        643.506489
std        1016.334786
min           0.000000
25%         120.000000
50%         300.000000
75%         840.000000
max       84480.000000
Name: duration, dtype: float64
```

Использовали моду и выявили что длительность сессии равна 60 секундам

Комментарии от ревьюера: Всё верно. Мода в данном случае наиболее корреткная метрика среднего

Как часто люди возвращаются?

смотрим на активность пользователей

сгруппируем данные пользователей по первой сессии и добавим в таблицу

In [34]:

```
first_activity = visits.groupby('uid').agg({'start_ts':'min'})
first_activity.columns = ['first_activity']
visits = visits.join(first_activity, on='uid')
```

введем дополнительные столбцы для расчета Retention Rate и посмотрим на "время жизни" когорты

In [35]:

```
visits['first_month'] = visits['first_activity'].astype('datetime64[M]')
visits['session_month'] = visits['session_date'].astype('datetime64[M]')
```

In [36]:

```
visits['cohort_lifetime'] = ((visits['session_month'] - visits['first_month']) / np.timedelta64(1,'M')).round().astype('int')
visits['first_month'] = visits['first_month'].dt.strftime('%Y-%m')
```

In [37]:

```
cohorts = visits.groupby(['first_month','cohort_lifetime'])['uid'].nunique().reset_index()
```

добавим количество пользователей в когортах

In [38]:

```
cohorts_users_count = cohorts[cohorts['cohort_lifetime'] == 0][['first_month','uid']]
cohorts_users_count = cohorts_users_count.rename(columns={'uid':'cohort_users'})
cohorts_users_count.head()
```

Out[38]:

	first_month	cohort_users
0	2017-06	13259
12	2017-07	13140
23	2017-08	10181
33	2017-09	16704
42	2017-10	25977

присоединим к датафрейму

In [39]:

```
cohorts = cohorts.merge(cohorts_users_count,on='first_month')
```

User Retention

In [40]:

```
cohorts['retention'] = cohorts['uid'] / cohorts['cohort_users']
```

In [41]:

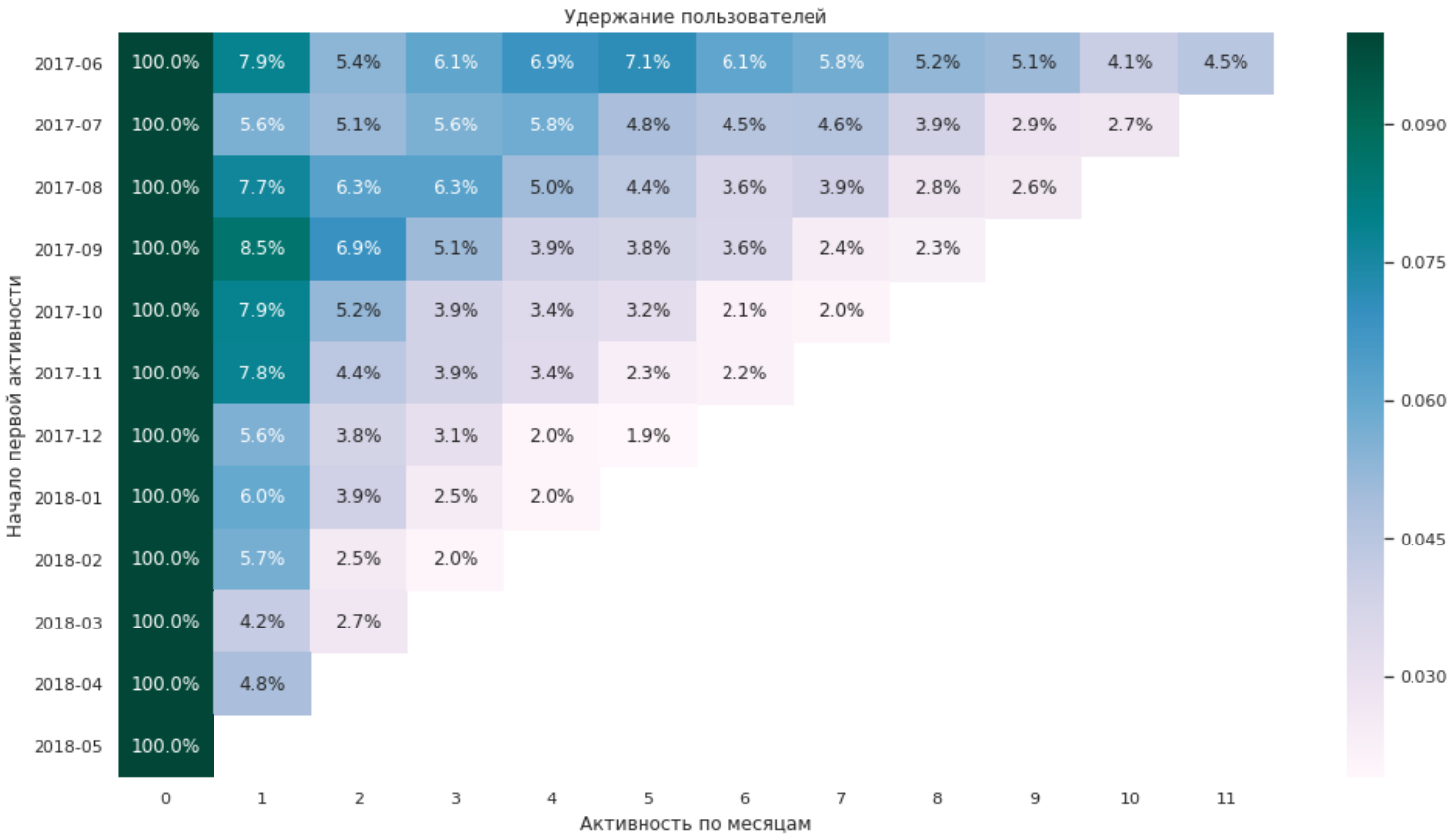
```
retention_pivot = cohorts.pivot_table(
    index='first_month',
```

```
    columns='cohort_lifetime',
    values='retention',
    aggfunc='sum')


sns.set(style = 'white')
plt.figure(figsize=(17, 9))
plt.title('Удержание пользователей')
sns.heatmap(retention_pivot, annot=True, fmt='.1%',vmax=0.10,cmap="PuBuGn");
plt.ylabel('Начало первой активности')
plt.xlabel('Активность по месяцам')
```

Out[41]:

Text(0.5, 57.5, 'Активность по месяцам')



Кофициент удержание значительно мал, пользователи возвращаются редко

Самая высокая активность в сентебре

> **Комментарии от ревьюера v2:** Отлично. Совсем другое дело

- Продажи
  - Когда люди начинают покупать?
  - Сколько раз покупают за период?
  - Какой средний чек?
  - Сколько денег приносят? (LTV)

Когда люди начинают покупать?

In [42]:

```
first_activity = visits[['uid','first_activity']]
buy = orders.merge(first_activity, on='uid')
buy.head()
```

Out[42]:

	buy_ts	revenue	uid	first_activity
0	2017-06-01 00:10:00	17.00	10329302124590727494	2017-06-01 00:09:00

	buy_ts	revenue	uid	first_activity
1	2017-06-01 00:25:00	0.55	11627257723692907447	2017-06-01 00:14:00
2	2017-06-01 00:27:00	0.37	17903680561304213844	2017-06-01 00:25:00
3	2017-06-01 00:29:00	0.55	16109239769442553005	2017-06-01 00:14:00
4	2017-06-01 07:58:00	0.37	14200605875248379450	2017-06-01 07:31:00

проверим дубли

In [43]:

```
buy.duplicated().value_counts()
```

Out[43]:

```
True     711392
False     50415
dtype: int64
```

In [44]:

```
buy = buy.drop_duplicates().reset_index(drop=True)
```

In [45]:

```
buy.duplicated().value_counts()
```

Out[45]:

```
False    50415
dtype: int64
```

Произведем вывод первых покупок

In [46]:

```
first_buy = buy.groupby('uid')['buy_ts'].min()
first_buy.name = 'first_buy'
buy = pd.merge(buy,first_buy,on='uid')
buy.head()
```

Out[46]:

	buy_ts	revenue	uid	first_activity	first_buy
0	2017-06-01 00:10:00	17.00	10329302124590727494	2017-06-01 00:09:00	2017-06-01 00:10:00
1	2017-06-01 00:25:00	0.55	11627257723692907447	2017-06-01 00:14:00	2017-06-01 00:25:00
2	2017-06-01 00:27:00	0.37	17903680561304213844	2017-06-01 00:25:00	2017-06-01 00:27:00
3	2017-06-01 00:29:00	0.55	16109239769442553005	2017-06-01 00:14:00	2017-06-01 00:29:00
4	2017-06-01 07:58:00	0.37	14200605875248379450	2017-06-01 07:31:00	2017-06-01 07:58:00

Вычислим время от первой сессии, до первой покупки

In [47]:

```
buy['seconds_to_buy'] = (buy['first_buy'] - buy['first_activity']).dt.seconds
buy.head()
```

Out[47]:

	buy_ts	revenue	uid	first_activity	first_buy	seconds_to_buy
0	2017-06-01 00:10:00	17.00	10329302124590727494	2017-06-01 00:09:00	2017-06-01 00:10:00	60
1	2017-06-01 00:25:00	0.55	11627257723692907447	2017-06-01 00:14:00	2017-06-01 00:25:00	660
2	2017-06-01 00:27:00	0.37	17903680561304213844	2017-06-01 00:25:00	2017-06-01 00:27:00	120
3	2017-06-01 00:29:00	0.55	16109239769442553005	2017-06-01 00:14:00	2017-06-01 00:29:00	900
4	2017-06-01 07:58:00	0.37	14200605875248379450	2017-06-01 07:31:00	2017-06-01 07:58:00	1620

проведем категоризацию

In [48]:

```
def category_time(row):
    seconds = row['seconds_to_buy']
    if seconds < 600:
        return '0 - 10 минут'
    if seconds <= 1200:
        return '10 - 30 минут'
    if seconds <= 12600:
        return '30 минут - 4 часа'
    else:
        return 'больше 4 часов'

buy['category'] = buy.apply(category_time, axis = 1)
```

In [49]:

```
pie = buy.groupby('category')['uid'].count().reset_index()

labels = pie.category
values = pie.uid

fig = go.Figure(data=[go.Pie(labels=labels, values=values,hole=0.3)])
fig.update_traces(hoverinfo='label+percent', textinfo='label+value+percent'
                  )
fig.update_layout(
    title_text="Время покупки")
fig.show()
```

1.Большая конверсия наблюдается в промежутке от 0 до 10 минут

2.30% заказов в промежутке от 10 минут до 4х часов

3. И 30% с течением большего времени

В принципе все логично - клиент делает покупку сразу, это скорее прописная истина

Комментарии от ревьюера: Хороший подход. Ты создал категории и верно рассчитал показатели

Сколько раз покупают за период?

In [50]:

```
buy['first_buy_month'] = buy['first_buy'].astype('datetime64[M]')
buy['buy_month'] = buy['buy_ts'].astype('datetime64[M]')
```

In [51]:

```
fig = px.line(buy.groupby('buy_month')['revenue'].count().reset_index(), x='buy_month', y='revenue')
fig.update_layout(
    title_text="Заказы в месяц")
fig.show()
```

In [52]:

```
buy.groupby('buy_month')['revenue'].count().mean()
```

Out[52]:

3878.076923076923

In [53]:

```
orders['buy_date'] = orders['buy_ts'].dt.date
orders['buy_week'] = orders['buy_ts'].dt.week
orders['buy_month'] = orders['buy_ts'].dt.month
day_buy = orders.groupby(['buy_date', 'uid']).agg({'buy_ts':'count'}).mean()[0]
week_buy = orders.groupby(['buy_week', 'uid']).agg({'buy_ts':'count'}).mean()[0]
month_buy = orders.groupby(['buy_month', 'uid']).agg({'buy_ts':'count'}).mean()[0]
```

In [54]:

```
print('''В среднем на одного пользователя покупок:
В день - {:.2f},
В неделю - {:.2f},
В месяц - {:.2f},'''.format(day_buy, week_buy, month_buy))
```

В среднем на одного пользователя покупок:
В день - 1.08,
В неделю - 1.16,
В месяц - 1.23,

1. Наблюдаем рост с августа по декабрь, максимальное количество заказов = 6218

2. С декабря по май идет спад, спад по 4346

3. Видно что летом ситуация стабильно низкая

Комментарии от ревьюера v3: Отлично. Это именно то, что нужно

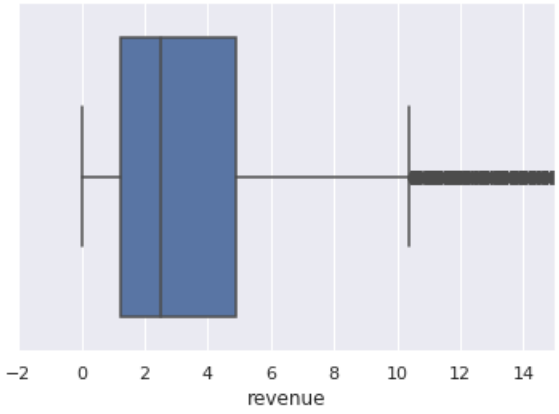
Какой средний чек?

построим диаграмму размаха

In [55]:

```
sns.set()
a= sns.boxplot(x=orders.revenue)
a.set_xlim([-2, 15]);
print('Средний чек = {}'.format(orders.revenue.median()))
```

Средний чек = 2.5



Комментарии от ревьюера: Здесь всё верно. Ты взял медиану, но можно было бы вывести describe с другими показателями, хотя boxplot тоже довольно информативный инструмент. Молодец

Сколько денег приносят? (LTV)

создадим таблицы с группировкой по месяцу покупки и когортам и количеством покупателей соответственно и соединим их

In [56]:

```
cohort_clients = buy.groupby('first_buy_month').agg({'uid': 'nunique'}).reset_index()
cohort_clients.columns = ['first_buy_month', 'n_clients']
cohorts1 = buy.groupby(['first_buy_month','buy_month']).agg({'revenue': 'sum'}).reset_index()
result = pd.merge(cohort_clients, cohorts1, on='first_buy_month')
result.head()
```

Out[56]:

	first_buy_month	n_clients	buy_month	revenue
0	2017-06-01	2023	2017-06-01	9557.49
1	2017-06-01	2023	2017-07-01	981.82
2	2017-06-01	2023	2017-08-01	885.34
3	2017-06-01	2023	2017-09-01	1931.30
4	2017-06-01	2023	2017-10-01	2068.58

добавим возраст когорты

In [57]:

```
result['age_cohorts'] = (result['buy_month'] - result['first_buy_month']) / np.timedelta64(1, 'M')
result['age_cohorts'] = result['age_cohorts'].round().astype('int')
```

Формула маржинальности: M = (B – 3)/B x 100%, где. M – маржа (в процентах); B – валовая выручка (предприятия или отдельно взятого подразделения); 3 – затраты (себестоимость продукции, аренда, зарплаты, налоги).

нет данных по маржинальности, исходя из данных проекта думаю что LTV предлагают посчитать по другой формуле (представлена ниже), но думаю лучше представить маржинальность как 1 и сделать сноску в выводах на этот показатель

Комментарии от ревьюера: Всё правильно. В данном случае маржинальность стоит принять за 1

— AOV (Average Order Value) — средний чек;

— RPR (Repeat Purchase Rates) — частота повторных покупок;

— Lifetime — длительность взаимодействия с клиентом;

LTV = Lifetime x AOV x RPR

In [58]:

```
marginality = 1
```

In [59]:

```
result['gp'] = result['revenue'] * marginality
result['ltv'] = result['gp'] / result['n_clients']
```

In [60]:

```
marginality = 1
```

In [61]:

```
output = result.pivot_table(
        index='first_buy_month',
        columns='age_cohorts',
        values='ltv',
        aggfunc='mean').round(3)
```

In [62]:

```
output.head()
```

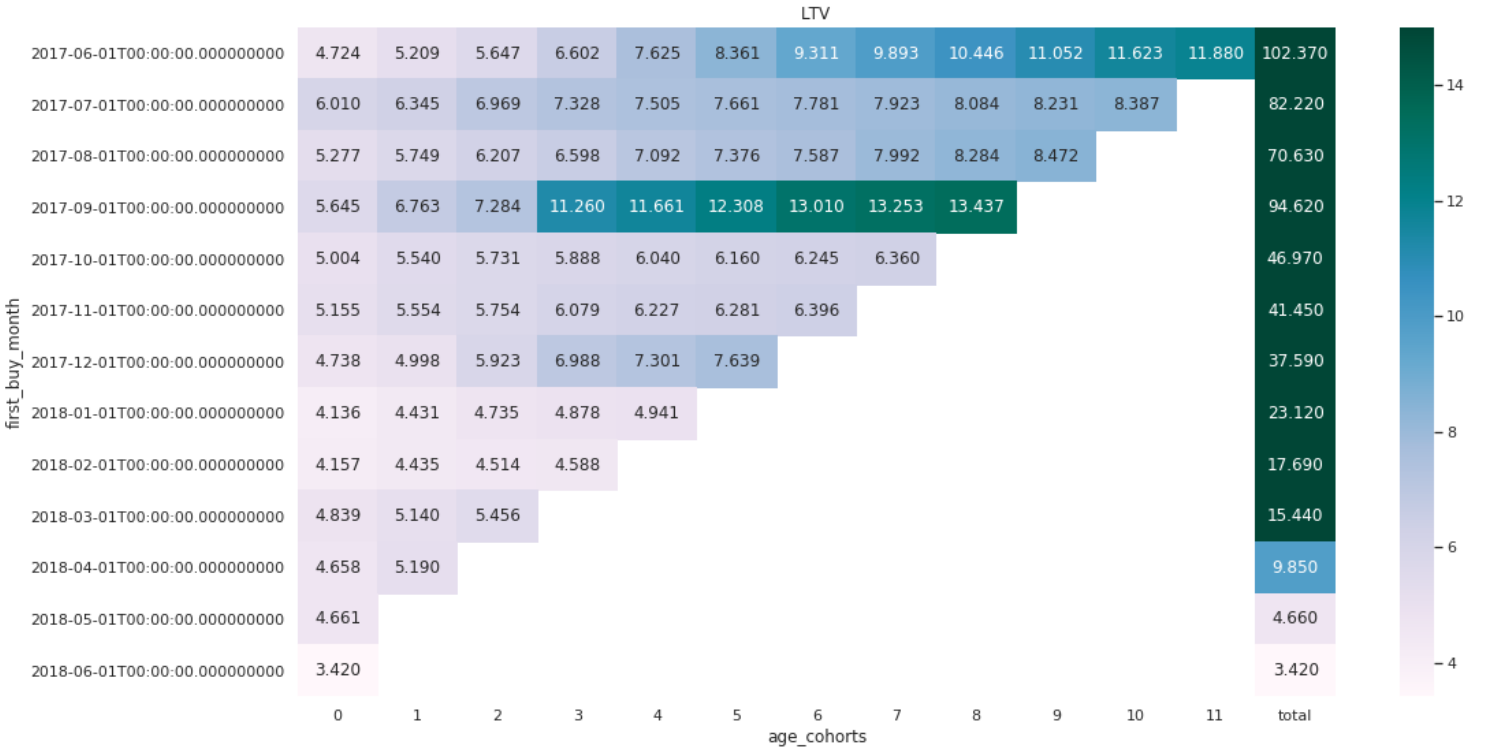
Out[62]:

age_cohorts	0	1	2	3	4	5	6	7	8	9	10	11
first_buy_month												

age_cohorts	0	1	2	3	4	5	6	7	8	9	10	11
first_buy_month												
2017-06-01	4.724	0.485	0.438	0.955	1.023	0.736	0.950	0.582	0.553	0.606	0.571	0.257
2017-07-01	6.010	0.335	0.624	0.359	0.177	0.156	0.120	0.142	0.161	0.147	0.156	NaN
2017-08-01	5.277	0.472	0.458	0.391	0.494	0.284	0.211	0.405	0.292	0.188	NaN	NaN
2017-09-01	5.645	1.118	0.521	3.976	0.401	0.647	0.702	0.243	0.184	NaN	NaN	NaN
2017-10-01	5.004	0.536	0.191	0.157	0.152	0.120	0.085	0.115	NaN	NaN	NaN	NaN

In [63]:

```
sns.set(style='white')
plt.figure(figsize=(17, 9))
plt.title('LTV')
output = output.cumsum(axis=1)
output['total'] = output.sum(axis=1).round(2)
sns.heatmap(output, annot=True, fmt='.3f',vmax=15,cmap="PuBuGn");
```



Извиняюсь, случайно пролистал это замечание(добавил cumsum

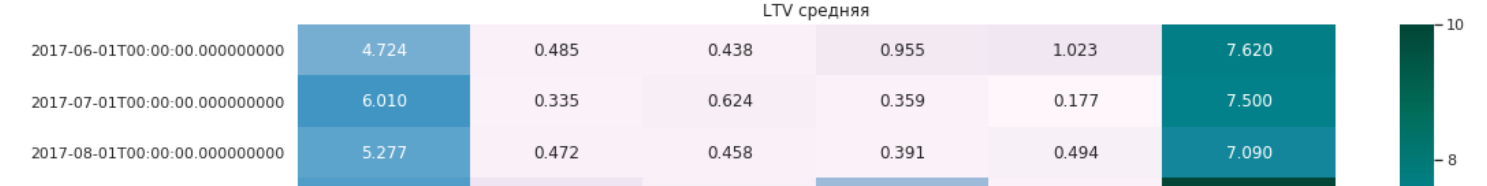
Комментарии от ревьюера v3: Здорово. Теперь тепловая карта корректна!

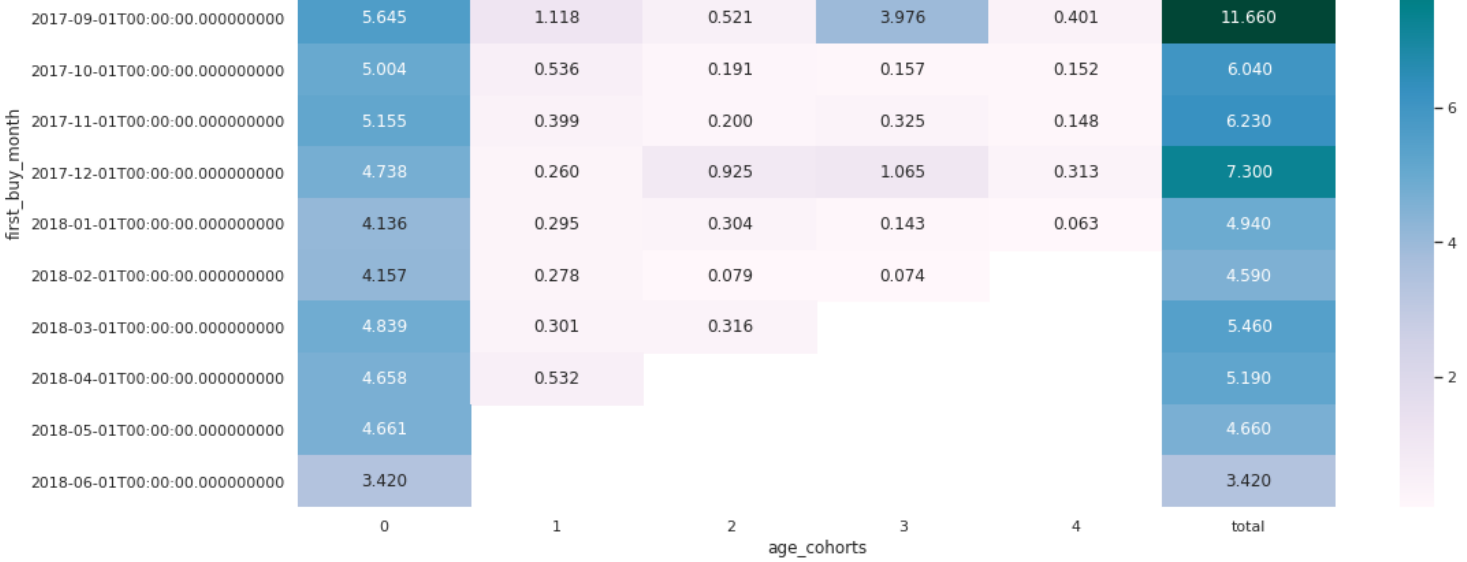
In [64]:

```
output = result.pivot_table(
    index='first_buy_month',
    columns='age_cohorts',
    values='ltv',
    aggfunc='mean').round(3)
```

In [65]:

```
output = output[[0,1,2,3,4]]
sns.set(style='white')
plt.figure(figsize=(17, 9))
plt.title('LTV средняя')
output['total'] = output.sum(axis=1).round(2)
sns.heatmap(output, annot=True, fmt='.3f',vmax=10,cmap="PuBuGn");
```





In [66]:

```
print('LTV за 5 месяцев = {}'.format(output['total'].mean()))
```

LTV за 5 месяцев = 6.284615384615384

1.Клиент тратит деньги в первый месяц и возвращается редко (подвердилось малое удержание пользователей)

2. Сентябрь выделяется с LTV = 3.976

3. Больше всего денег принесла сентябрьская когорта, вторая в июне

4. Продажи начали падать с января по май и колебляться примерно на одном уровне

5. Минимальная когорта в январе - видимо связанная с новогодними праздниками

Маркетинг

- Сколько денег потратили? Всего / на каждый источник / по времени
- Сколько стоило привлечение одного покупателя из каждого источника?
- На сколько окупились расходы? (ROI)

Сколько денег потратили? Всего / на каждый источник / по времени

для начала найдем затраты на привлечения нового клиента (CAC)

смотрим на затраты

In [67]:

```
costs.head()
```

Out[67]:

	source_id	dt	costs
0	1	2017-06-01	75.20
1	1	2017-06-02	62.25
2	1	2017-06-03	36.53
3	1	2017-06-04	55.00
4	1	2017-06-05	57.08

In [68]:

```
costs['month'] = costs['dt'].astype('datetime64[M]')
```

Общая картина есть, группируем по месяцам

In [69]:

```
month_costs = costs.groupby('month')['costs'].sum().reset_index()
month_costs.head()
```

Out[69]:

	month	costs
0	2017-06-01	18015.00
1	2017-07-01	18240.59
2	2017-08-01	14790.54
3	2017-09-01	24368.91
4	2017-10-01	36322.88

добавим данные в к конечной таблице

In [70]:

```
result.head()
```

Out[70]:

	first_buy_month	n_clients	buy_month	revenue	age_cohorts	gp	ltv
0	2017-06-01	2023	2017-06-01	9557.49	0	9557.49	4.724414
1	2017-06-01	2023	2017-07-01	981.82	1	981.82	0.485329
2	2017-06-01	2023	2017-08-01	885.34	2	885.34	0.437637
3	2017-06-01	2023	2017-09-01	1931.30	3	1931.30	0.954671
4	2017-06-01	2023	2017-10-01	2068.58	4	2068.58	1.022531

In [71]:

```
month_costs.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12 entries, 0 to 11
Data columns (total 2 columns):
month    12 non-null datetime64[ns]
costs    12 non-null float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 320.0 bytes
```

In [72]:

```
result1 = pd.merge(result, month_costs, left_on='first_buy_month', right_on='month')
result1['cac'] = result1['costs'] / result1['n_clients']
```

```
result1.sample(5)
```

Out[72]:

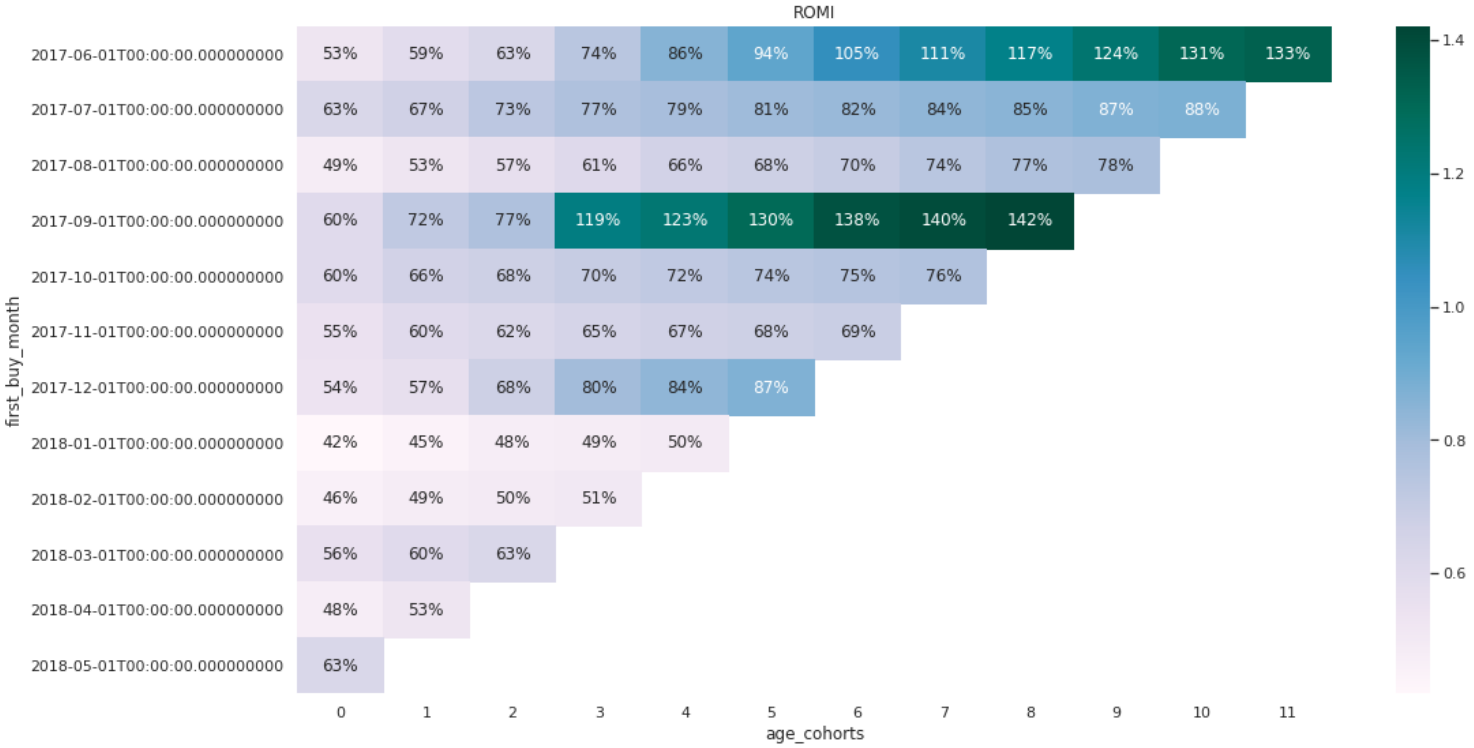
	first_buy_month	n_clients	buy_month	revenue	age_cohorts	gp	ltv	month	costs	cac
38	2017-09-01	2581	2018-02-01	1670.08	5	1670.08	0.647067	2017-09-01	24368.91	9.441654
71	2018-02-01	3651	2018-05-01	270.70	3	270.70	0.074144	2018-02-01	32723.03	8.962758
53	2017-11-01	4081	2018-02-01	1326.13	3	1326.13	0.324952	2017-11-01	37907.88	9.288870
24	2017-08-01	1370	2017-09-01	646.63	1	646.63	0.471993	2017-08-01	14790.54	10.796015
28	2017-08-01	1370	2018-01-01	388.45	5	388.45	0.283540	2017-08-01	14790.54	10.796015

Вычислим ROMI

In [73]:

```
result1['romi'] = result1['ltv'] / result1['cac']
romi = result1.pivot_table(
        index='first_buy_month',
        columns='age_cohorts',
        values='romi',
        aggfunc='mean').cumsum(axis=1).round(2)

sns.set(style='white')
plt.figure(figsize=(17, 9))
plt.title('ROMI')
sns.heatmap(romi, annot=True, fmt='.0%',cmap="PuBuGn");
```



Сколько денег потратили?

Расчитаем сколько всего денег потратили, для этого сгруппируем данные по uid и revenue

In [74]:

```
orders.head()
orders1 = orders.groupby('uid')['revenue'].sum().reset_index()
orders1['revenue'].sum()
```

Out[74]:

252057.2

In [75]:

```
orders1.duplicated()
```

Out[75]:

```
0        False
1        False
2        False
3        False
4        False
         ...
36518    False
36519    False
36520    False
36521    False
36522    False
Length: 36523, dtype: bool
```

на каждый источник/по времени

объединим visits и orders по часу заказа для определения заказа в источнике, в котором была данная сессия

In [76]:

```
v['start_ts'] = v['start_ts'].dt.strftime('%Y-%m-%d %H')
o['buy_ts'] = o['buy_ts'].dt.strftime('%Y-%m-%d %H')
v.drop(['end_ts','device'],axis=1,inplace=True)
v.rename(columns={'start_ts': 'buy_ts'}, inplace=True)
orders_visists = o.merge(v, on = ['buy_ts','uid'], how = 'inner')
orders_visists['buy_month'] = orders_visists['buy_ts'].astype('datetime64[M]')
```

посторим график для наглядности

In [77]:

```
fig = px.bar(
    orders_visists.groupby(['source_id','buy_month'])['revenue'].sum().reset_index(),
    x='buy_month', y='revenue')
fig.show()
```

In [78]:

```
fig = px.line(buy.groupby('buy_month')['revenue'].sum().reset_index(), x='buy_month', y='revenue')
```

```
fig = px.line(buy.groupby('buy_month')['revenue'].sum().reset_index(), x='buy_month', y='revenue')
fig.update_layout(
    title_text="Выручка по месяцам")
fig.show()
```

*Схожий график с данными по заказам в месяц

1. Максимальный доход в декабре, минимальный в августе

2. Большие источники: В декабре 2ой источник, в октябре - 5ый

3. Минимальный источник - 2ой канал в августе

Комментарии от ревьюера: Ок. С продажами ты справился успешно. Идем дальше

Сколько стоило привлечение одного покупателя из каждого источника?

Создадим таблицу где будут:

* затраты
* количество покупателей
* стоимость привлечения покупателя
* выручка

In [79]:

```
o1 = orders.groupby('uid')['revenue'].sum().reset_index()
o2 = visits.copy()
# Добавим общую сумму их покупок
o2 = o2.merge(o1, on ='uid')
# Сгруппируем o2 по uid и выведем первую сессию
o3 = o2.groupby('uid')['start_ts'].min().reset_index()
# Соеденим o2 и o3 по uid и мин.сессии методом inner
o4 = o2.merge(o3, on = ['uid','start_ts'], how = 'inner')
```

In [80]:

```
revenue_group = o4.groupby('source_id')['revenue'].sum().reset_index()
```

```
costs_group = costs.groupby('source_id')['costs'].sum().reset_index()
source_group = o4.groupby('source_id')['uid'].nunique().reset_index()
costs_group = costs_group.merge(source_group,on = 'source_id')
costs_group['cost_per_user'] = costs_group['costs'] / costs_group['uid']
costs_group = costs_group.merge(revenue_group,on = 'source_id')
costs_group
```

Out[80]:

	source_id	costs	uid	cost_per_user	revenue
0	1	20833.27	2899	7.186364	31090.55
1	2	42806.04	3506	12.209367	46923.61
2	3	141321.63	10473	13.493901	54511.24
3	4	61073.60	10296	5.931779	56696.83
4	5	51757.10	6931	7.467479	52624.02
5	9	5517.49	1088	5.071222	5759.40
6	10	5822.49	1329	4.381106	4450.33

Комментарии от ревьюера: Отлично. CAC рассчитан верно

добавим ltv и roi в таблицу

In [81]:

```
costs_group['gp'] = costs_group['revenue'] * marginality
costs_group['ltv'] = costs_group['gp'] / costs_group['uid']
costs_group['roi'] = costs_group['ltv'] / costs_group['cost_per_user'] *100
costs_group
```

Out[81]:

	source_id	costs	uid	cost_per_user	revenue	gp	ltv	roi
0	1	20833.27	2899	7.186364	31090.55	31090.55	10.724577	149.235094
1	2	42806.04	3506	12.209367	46923.61	46923.61	13.383802	109.619133
2	3	141321.63	10473	13.493901	54511.24	54511.24	5.204931	38.572468
3	4	61073.60	10296	5.931779	56696.83	56696.83	5.506685	92.833614
4	5	51757.10	6931	7.467479	52624.02	52624.02	7.592558	101.674978
5	9	5517.49	1088	5.071222	5759.40	5759.40	5.293566	104.384421
6	10	5822.49	1329	4.381106	4450.33	4450.33	3.348631	76.433450

На сколько окупились расходы? (ROI)

In [82]:

```
costs_group.style.bar(subset=['roi', 'uid'], color='lightblue')
```

Out[82]:

	source_id	costs	uid	cost_per_user	revenue	gp	ltv	roi
0	1	20833.3	2899	7.18636	31090.5	31090.5	10.7246	149.235
1	2	42806	3506	12.2094	46923.6	46923.6	13.3838	109.619
2	3	141322	10473	13.4939	54511.2	54511.2	5.20493	38.5725
3	4	61073.6	10296	5.93178	56696.8	56696.8	5.50669	92.8336
4	5	51757.1	6931	7.46748	52624	52624	7.59256	101.675
5	9	5517.49	1088	5.07122	5759.4	5759.4	5.29357	104.384
6	10	5822.49	1329	4.38111	4450.33	4450.33	3.34863	76.4335

Выводы по таблице

1. Прибыльный источник, ROI 149%, исходя из того что привлечено достаточно малое количество клиентов (2899) - рекомендуется вкладывать больше денег в маркетинг для того чтобы клиенты

2. ROI положительный, не так много клиентов - рекомендовано вкладывать деньги на привлечение

3. Не рекомендованный источник, ROI маленький, клиентов при этом много

4. ROI небольшой, но имеет самую большую доходность

5. ROI положительный, имеет доходность на 2ом месте, привлекательный источник

6. ROI положительный, имеет самое низкое количество клиентов

7. Не слишком привлекательный источник

> **Комментарии от ревьюера:** И на этом пункте замечаний нет. Ты хорошо интерпретируешь полученные данные - это хорошее качество дла аналитика. Развивай его и дальше

ОБЩИЙ ВЫВОД

- расчеты производились при маржинальности - 100%

Показатели

DAU, MAU, WAU

- DAU = 907, WAU = 5716, MAU = 23228

Количество сессий в день

- В среднем пользователи совершают 1 сессию в день

Retention Rate

- Кофициент удержание значительно мал, пользователи возвращаются редко
- Самая высокая активность в сентебре

*Необходимо проанализировать сентябрьскую когорту и выяснить с чем связан высокий коэфициент удержания в этом месяце и распространить ту же метрику на другие месяца

*Необходимо увеличить коэфициент удержания

Продажи

Когда люди начинают покупать?

1. Большая конверсия наблюдается в промежутке от 0 до 10 минут
2. 30% заказов в промежутке от 10 минут до 4х часов
3. И 30% с течением времени больше 4х часов

Сколько раз покупают за период?

1. Наблюдаем рост с августа по декабрь, максимальное количество заказов = 6218
2. С декабря по май идет спад, спад по 4346
3. Видно что летом ситуация стабильно низкая

Какой средний чек?

Средний чек = 2.5

Сколько денег приносят? (LTV)

1. Клиент тратит деньги в первый месяц и возвращается редко (подвердилось малое удержание пользователей)
2. Сентябрь выделяется с LTV = 3.976
3. Больше всего денег принесла сентябрьская когорта, вторая в июне
4. Продажи начали падать с января по май и колебляться примерно на одном уровне
5. Минимальная когорта в январе - видимо связанная с новогодними праздниками

Маркетинг

Сколько всего потратили денег?

Всего потрачено 252057.2

ROMI

1. Максимальный доход в декабре, минимальный в августе
2. Большие источники: В декабре 2ой источник, в октябре - 5ый
3. Минимальный источник - 2ой канал в августе

Рекомендации

1. Прибыльный источник, ROI 149%, исходя из того что привлечено достаточно малое количество клиентов (2899) - рекомендуется вкладывать больше денег в маркетинг для того чтобы клиенты
2. ROI положительный, не так много клиентов - рекомендовано вкладывать деньги на привлечение
3. Не рекомендованный источник, ROI маленький, клиентов при этом много
4. ROI небольшой, но имеет самую большую доходность
5. ROI положительный, имеет доходность на 2ом месте, привлекательный источник
6. ROI положительный, имеет самое низкое количество клиентов
7. Не слишком привлекательный источник

Комментарии от ревьюера: Итоговый вывод завершает твое исследование. Ты вывел основные результаты и самое главное - дал рекомендации бизнесу. Это важный пункт и ты его не забыл. Так держать!