Описание проекта

Мобильные приложения — Выделение групп пользователей на основе поведения

Выделите группы пользователей, которые различаются по метрикам:

1. retention rate,
2. время, проведённое в приложении,
3. частота действий,
4. конверсия в целевое действие — просмотр контактов.

- **Проведите исследовательский анализ данных**
- **Сегментируйте пользователей на основе действий**
- **Проверьте статистические гипотезы**
    1. *Некоторые пользователи установили приложение по ссылке из* `yandex`*, другие — из* `google`*. Проверьте гипотезу: две эти группы демонстрируют разную конверсию в просмотры контактов.*
    2. *Сформулируйте собственную гипотезу. Дополните её нулевой и альтернативной гипотезами. Проведите статистический тест.*

Описание данных

Датасет содержит данные о событиях, совершенных в мобильном приложении "Ненужные вещи". В нем пользователи продают свои ненужные вещи, размещая их на доске объявлений.

В датасете содержатся данные пользователей, впервые совершивших действия в приложении после 7 октября 2019 года.

Датасет **mobile_dataset.csv** содержит колонки:

- `event.time` — время совершения
- `event.name` — название события
- `user.id` — идентификатор пользователя

Датасет **mobile_sources.csv** содержит колонки:

- `userId` — идентификатор пользователя
- `source` — источник, с которого пользователь установил приложение

Расшифровки событий:

- `advert_open` — открытие карточки объявления
- `photos_show` — просмотр фотографий в объявлении
- `tips_show` — пользователь увидел рекомендованные объявления
- `tips_click` — пользователь кликнул по рекомендованному объявлению
- `contacts_show` и `show_contacts` — пользователь нажал на кнопку "посмотреть номер телефона" на карточке объявления
- `contacts_call` — пользователь позвонил по номеру телефона на карточке объявления
- `map` — пользователь открыл карту размещенных объявлений
- `search_1` — `search_7` — разные события, связанные с поиском по сайту
- `favorites_add` — добавление объявления в избранное

In [3]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats as st
import math as mth
import numpy as np
import plotly.express as px
from plotly import graph_objects as go
from datetime import datetime
from datetime import date
from datetime import timedelta

from statsmodels.stats.proportion import proportions_ztest
```

In [4]:
```
mobile_sources = pd.read_csv('https://code.s3.yandex.net/datasets/mobile_soures.csv')
mobile_dataset = pd.read_csv('https://code.s3.yandex.net/datasets/mobile_dataset.csv'
```

Предобработка

In [5]:
```
# функция для краткого ознакомления с датасетом
def research(df):
    print(df.info(), '\n')
    print('Дуликаты:', df.duplicated().sum())
    print('\n Пропуски:\n', df.isna().sum())
```

In [6]:
```
research(mobile_sources)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4293 entries, 0 to 4292
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   userId  4293 non-null   object
 1   source  4293 non-null   object
dtypes: object(2)
memory usage: 67.2+ KB
None

Дуликаты: 0

 Пропуски:
 userId    0
source    0
dtype: int64
```

In [7]:
```
research(mobile_dataset)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 74197 entries, 0 to 74196
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   event.time  74197 non-null  object
 1   event.name  74197 non-null  object
 2   user.id     74197 non-null  object
dtypes: object(3)
memory usage: 1.7+ MB
None

Дуликаты: 0

 Пропуски:
 event.time    0
event.name    0
user.id       0
dtype: int64
```

Очевидных дубликатов нет, пропусков нет.

Для начала проверим кол-во уникальных пользователей в датасетах.

Поменяем названия колонок на более подходящие.

Для колонки `event.time` поменяем тип данных на datetime.

Проверим данные на неявные дубликаты

In [8]:
```
# Проверим на кол-во уникальных пользователей оба датафрейма
mobile_dataset['user.id'].nunique() == mobile_sources['userId'].nunique()
```

Out[8]:
```
True
```

In [9]:
```
# Проверим не повторяются ли айдишники пользователей
mobile_sources['userId'].duplicated().sum()
```

Out[9]:
```
0
```

In [10]:
```
# Заменим названия колонок
mobile_dataset.columns = [col.lower().replace('.','_') for col in mobile_dataset.colu
mobile_sources.columns = ['user_id', 'source']
```

In [11]:
```
# Приведем данные к нужному типу
mobile_dataset['event_time'] = pd.to_datetime(mobile_dataset['event_time'])
```

In [12]:
```
# посмотрим на список всех возможных действий и изучим на предмет неявных дубликатов
mobile_dataset['event_name'].unique()
```

Out[12]:
```
array(['advert_open', 'tips_show', 'map', 'contacts_show', 'search_4',
       'search_5', 'tips_click', 'photos_show', 'search_1', 'search_2',
       'search_3', 'favorites_add', 'contacts_call', 'search_6',
       'search_7', 'show_contacts'], dtype=object)
```

Предлагаю сразу объединить `show_contacts` и `contacts_show` в одно, тк это одинаковые действия. Также предлагаю объединить все действия связанные с поисков в одно

In [13]:
```
# список всех действий с поиском
search = ['search_4', 'search_2', 'search_5', 'search_6', 'search_1', 'search_3', 'se
# Функция для замены событий по условию
def correct_event(event):
    if event in search:
        return 'search'
    elif (event == 'show_contacts') | (event == 'contacts_show'):
        return 'contacts_show'
```

```
    else:
        return event
```

In [14]:
```
mobile_dataset['event_name'] = mobile_dataset['event_name'].apply(correct_event)
```

In [15]:
```
# Соберем общий датафрейм для удобства
df = mobile_sources.merge(mobile_dataset, on='user_id', how='inner')
df.head()
```

Out[15]:

	user_id	source	event_time	event_name
0	020292ab-89bc-4156-9acf-68bc2783f894	other	2019-10-07 00:00:00.431357	advert_open
1	020292ab-89bc-4156-9acf-68bc2783f894	other	2019-10-07 00:00:01.236320	tips_show
2	020292ab-89bc-4156-9acf-68bc2783f894	other	2019-10-07 00:00:07.039334	tips_show
3	020292ab-89bc-4156-9acf-68bc2783f894	other	2019-10-07 00:01:27.770232	advert_open
4	020292ab-89bc-4156-9acf-68bc2783f894	other	2019-10-07 00:01:34.804591	tips_show

In [16]:
```
# добавим колонку с датой без времени
df['event_date'] = df['event_time'].dt.date
df['event_date'] = pd.to_datetime(df['event_date'])
df.head()
```

Out[16]:

	user_id	source	event_time	event_name	event_date
0	020292ab-89bc-4156-9acf-68bc2783f894	other	2019-10-07 00:00:00.431357	advert_open	2019-10-07
1	020292ab-89bc-4156-9acf-68bc2783f894	other	2019-10-07 00:00:01.236320	tips_show	2019-10-07
2	020292ab-89bc-4156-9acf-68bc2783f894	other	2019-10-07 00:00:07.039334	tips_show	2019-10-07
3	020292ab-89bc-4156-9acf-68bc2783f894	other	2019-10-07 00:01:27.770232	advert_open	2019-10-07
4	020292ab-89bc-4156-9acf-68bc2783f894	other	2019-10-07 00:01:34.804591	tips_show	2019-10-07

In [17]:
```
# Проверим общий датафрейм
research(df)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 74197 entries, 0 to 74196
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   user_id     74197 non-null  object
 1   source      74197 non-null  object
 2   event_time  74197 non-null  datetime64[ns]
 3   event_name  74197 non-null  object
 4   event_date  74197 non-null  datetime64[ns]
dtypes: datetime64[ns](2), object(3)
memory usage: 3.4+ MB
None

Дуликаты: 0

 Пропуски:
 user_id       0
source        0
event_time    0
event_name    0
event_date    0
dtype: int64
```

Дублей нет, пропусков нет, типы данных - ОК

Добавлю в датафрейм некоторую информацию информацию:

- Время суток (утро, днеь, вечер, ночь)
- Совершил ли пользователь целовое действие. (Просмотр контактов)

Это понадобится для исследовательского анализа данных.

In [18]:
```python
# Функция возвращает время суток
def get_time_of_day(datetime):
    if datetime.hour >= 6 and datetime.hour <= 11:
        return 'morning'
    elif datetime.hour >= 12 and datetime.hour <= 17:
        return 'afternoon'
    elif datetime.hour >= 18 and datetime.hour <= 24:
        return 'evening'
    else:
        return 'night'
```

In [19]:
```python
df['time_of_day'] = df['event_time'].apply(get_time_of_day)
df.head()
```

Out[19]:

	user_id	source	event_time	event_name	event_date	time_of_day
0	020292ab-89bc-4156-9acf-68bc2783f894	other	2019-10-07 00:00:00.431357	advert_open	2019-10-07	night
1	020292ab-89bc-4156-9acf-68bc2783f894	other	2019-10-07 00:00:01.236320	tips_show	2019-10-07	night
2	020292ab-89bc-4156-9acf-68bc2783f894	other	2019-10-07 00:00:07.039334	tips_show	2019-10-07	night
3	020292ab-89bc-4156-9acf-68bc2783f894	other	2019-10-07 00:01:27.770232	advert_open	2019-10-07	night
4	020292ab-89bc-4156-9acf-68bc2783f894	other	2019-10-07 00:01:34.804591	tips_show	2019-10-07	night

In [20]:
```python
# Проставим каждому пользователю True или False в зависимости от выполнения целевого
df['contact_show'] = df['user_id'].isin(df[df['event_name'] == 'contacts_show']['user
```

```
df.head()
```

Out[20]:

	user_id	source	event_time	event_name	event_date	time_of_day	contact_show
0	020292ab-89bc-4156-9acf-68bc2783f894	other	2019-10-07 00:00:00.431357	advert_open	2019-10-07	night	False
1	020292ab-89bc-4156-9acf-68bc2783f894	other	2019-10-07 00:00:01.236320	tips_show	2019-10-07	night	False
2	020292ab-89bc-4156-9acf-68bc2783f894	other	2019-10-07 00:00:07.039334	tips_show	2019-10-07	night	False
3	020292ab-89bc-4156-9acf-68bc2783f894	other	2019-10-07 00:01:27.770232	advert_open	2019-10-07	night	False
4	020292ab-89bc-4156-9acf-68bc2783f894	other	2019-10-07 00:01:34.804591	tips_show	2019-10-07	night	False

In [21]:
```
# посмотрим данными за какой период мы обладаем
display(df['event_time'].min())
df['event_time'].max()
```

```
Timestamp('2019-10-07 00:00:00.431357')
```

Out[21]:
```
Timestamp('2019-11-03 23:58:12.532487')
```

Мы познакомились с данными и сделали предобработку.

- В нашем распоряжении 2 датасета с сессиями и источниками, из которых пришли пользователи
- Всего 4293 уникальных пользователя
- Данные предоставлены за период с 07 октября по 3 ноября включительно
- Для удобсвта собрали один общий датафрейм, в которым добавили некоторые данные

Исследовательский анализ данных

Проведем EDA для анализа данных и выявления критерия деления пользователей на группы. Исследуем следующие показатели:

- Кол-во действий по дням и времени суток
- Кол-во пользователей по дням и времени суток
- Кол-во действий на каждого пользователя
- Кол-во пользователей из каждого источника
- Кол-во активных дней для каждого пользователя
- Продолжительность сессий для каждого пользователя

После проведения исследовательского анализа данных, сделаем первые выводы и примем решения о критерии для сегментации пользователей.

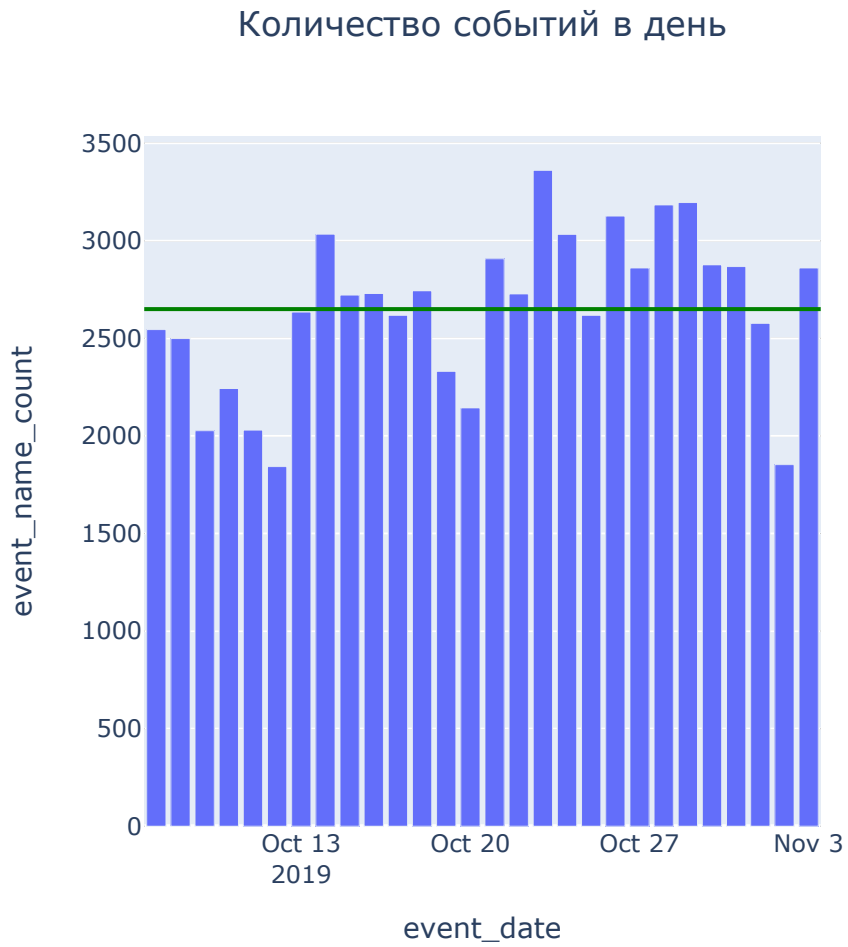
Количество действий в день

In [22]:
```
# сводник для первых графиков. Сохранять в отдельное переменной пока не вижу смысла
df.groupby('event_date', as_index=False).agg({'event_name': 'count', 'user_id': 'nuni
```

Out[22]:

	event_date	event_name	user_id
0	2019-10-07	2545	204
1	2019-10-08	2499	204
2	2019-10-09	2027	227
3	2019-10-10	2243	233
4	2019-10-11	2030	206

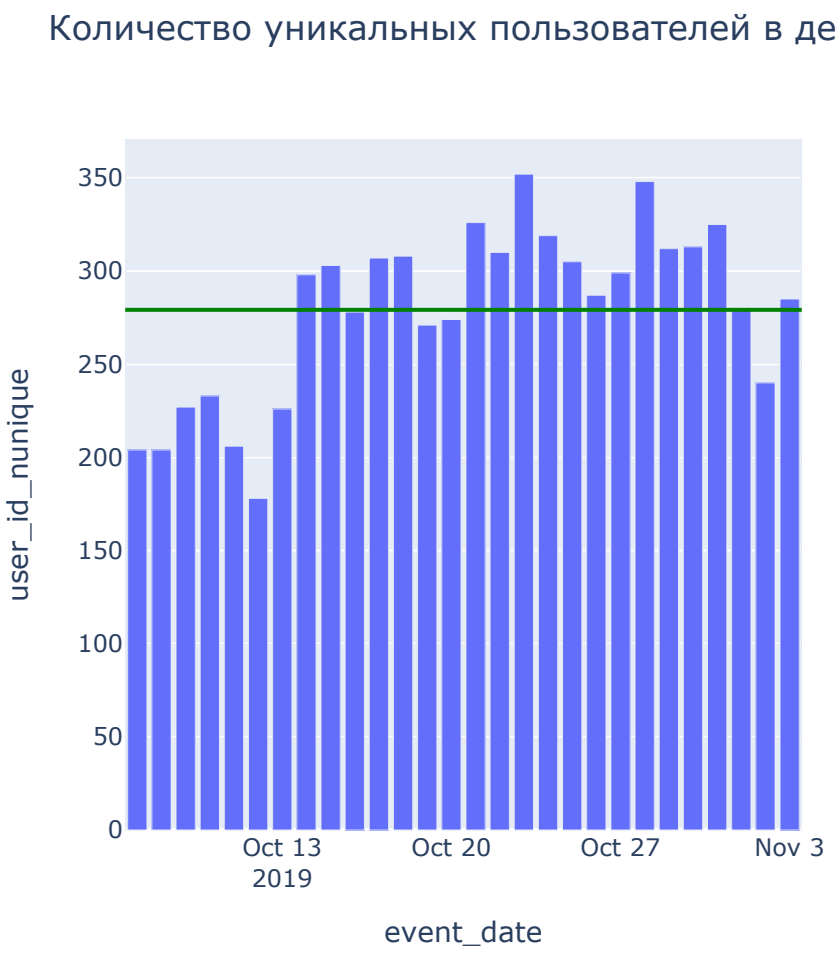
In [23]:
```
# Построим график по кол-ву действий в день
fig_01 = px.bar(df.groupby('event_date', as_index=False)
                .agg({'event_name': 'count', 'user_id': 'nunique'}),

                x='event_date', y='event_name',
                title='Количество событий в день')

# Добавим линию среднего значения кол-ва действий в день
fig_01.add_hline(y=df.groupby('event_date', as_index=False)
                 .agg({'event_name': 'count', 'user_id': 'nunique'})['event_name']
                 .mean(), line_color="green"
                 )
# Обновим название для оси У и выравним заголовок по центру
fig_01.update_yaxes(title_text='event_name_count')
fig_01.update_layout(title_x=0.5)

# Вывод графика
fig_01.show()
```



Среднее кол-во действий в день около 2600. Меньше всего действий было 12 октября (1843) и 2 ноября(1853). Больше всего действий было совершено 23 октября (3361)

Количество уникальных пользователей в день

In [24]:
```
fig_02 = px.bar(df.groupby('event_date', as_index=False).agg({'event_name': 'count',
                x='event_date', y='user_id',
                title='Количество уникальных пользователей в день')

# Добавим линию среднего значения кол-ва действий в день
fig_02.add_hline(y=df.groupby('event_date', as_index=False)
                 .agg({'event_name': 'count', 'user_id': 'nunique'})['user_id']
                 .mean(), line_color="green"
                 )

# Обновим название для оси У и выравним заголовок по центру
fig_02.update_yaxes(title_text='user_id_nunique')
fig_02.update_layout(title_x=0.5)

# Вывод графика
fig_02.show()
```



Среднее кол-во уникальных пользователей в день около 280. Наименьшее кол-во уникальныз пользователей было 12 октября - 178, что также совпадает с наименьшим кол-вом действий в день. Однако 2 ноября, в день когда совершлиось второе снизу кол-во действий, уникальных пользователей было 240 пользователей. Максимальное кол-во пользователей было в приложении 23 октября (352), когда и совершилось максимальное кол-во действий в день.

Количество действий и пользователей по времени суток

Предлагаю взглянуть на кол-во действий и кол-во уникальных пользоваталей еще и в разрезе времени суток

In [25]:
```
# сводник по кол-ву действий, уникальных пользователей и среднего
# кол-ва действий на пользователя в зависимости от дня недели
(
    df.groupby('time_of_day', as_index=False)
    .agg({'event_name': 'count', 'user_id': 'nunique'})
    # доля от всех действий
    .assign(event_share=lambda x: x['event_name'] / x['event_name'].sum())
    # доля от всех пользователей
    .assign(user_share=lambda x: x['user_id'] / x['user_id'].sum())
    #кол-во действий на пользователя в среднем
    .assign(ratio=lambda x: x['event_name'] / x['user_id'])
)
```

Out[25]:

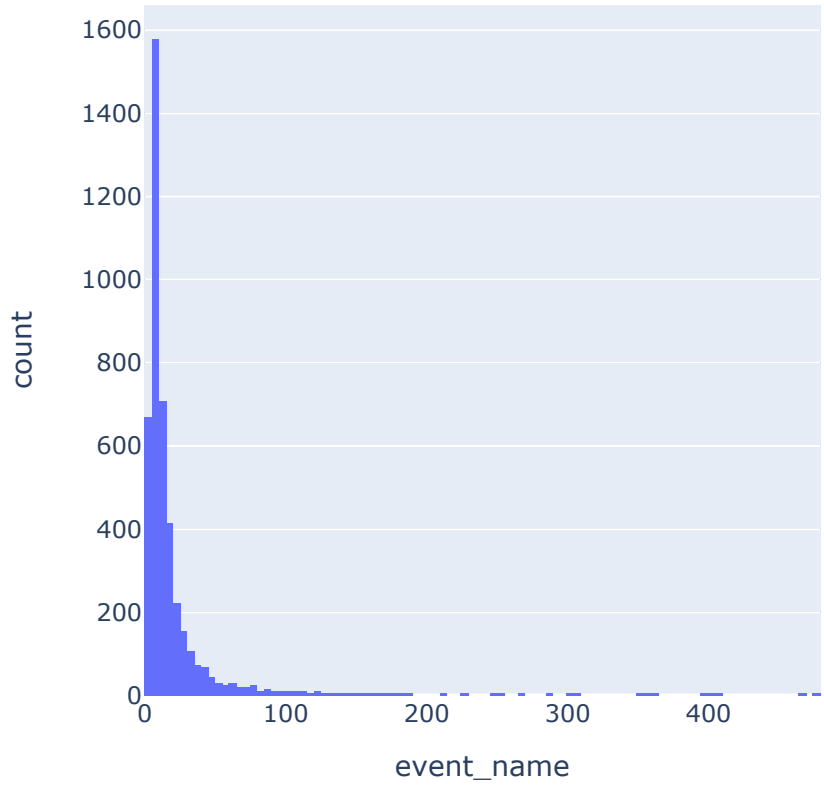
	time_of_day	event_name	user_id	event_share	user_share	ratio
0	afternoon	30454	2406	0.410448	0.383426	12.657523
1	evening	25976	2063	0.350095	0.328765	12.591372
2	morning	13976	1418	0.188363	0.225976	9.856135
3	night	3791	388	0.051094	0.061833	9.770619

Результаты достаточно ожидаемые:

- Днем и вечером пользователей и действий больше всего. Далее следует утро, а вот ночью и пользователей и действий меньше всего
- Среднее кол-во действий на пользователя днем и вечером примерно около 12, а вот ночью и утром около 10

Количество действий на каждого пользователя

In [26]:
```
# построим гистограмму по кол-во действий на пользователя
fig_03 = px.histogram(df.groupby('user_id', as_index=False)['event_name']
                      .count(), x="event_name",
                      title='Количество действий на каждого пользователя'
                     )
fig_03.update_layout(title_x=0.5)

fig_03.show()
```



In [27]:
```
df.groupby('user_id', as_index=False)['event_name'].count()['event_name'].describe()
```

Out[27]:
```
count    4293.000000
mean       17.283252
std        29.130677
min         1.000000
25%         5.000000
50%         9.000000
75%        17.000000
max       478.000000
Name: event_name, dtype: float64
```

Глядя на график, можно явно заметить выбросы в данных. Посмотрим на 90, 95й и 99й процентили

In [28]:
```
np.percentile(df.groupby('user_id', as_index=False)['event_name']
                        .count()['event_name'], [90, 95, 99])
```

Out[28]:
```
array([ 36.,  59., 132.])
```

99% пользователйй совершает 132 или меньше действий.
С одной стороны, руки сразу чешутся удалить эти данные и списать на выбросы. Но предлагаю сначала посмотреть какие именно действия совершают эти пользователи

In [29]:
```
df[(df['user_id'].isin(
    df.groupby('user_id', as_index=False)['event_name']
    .count().query('event_name > 132')['user_id'].unique()))]['event_name'].value_cou
```

Out[29]:
```
tips_show        5849
advert_open      1162
contacts_show     873
map               625
photos_show       542
search            299
favorites_add     195
tips_click         94
Name: event_name, dtype: int64
```

In [30]:
```
# Разбивка по всем действиям
df['event_name'].value_counts()
```

Out[30]:
```
tips_show        40055
photos_show      10012
search            6784
advert_open       6164
contacts_show     4529
map               3881
favorites_add     1417
tips_click         814
contacts_call      541
Name: event_name, dtype: int64
```

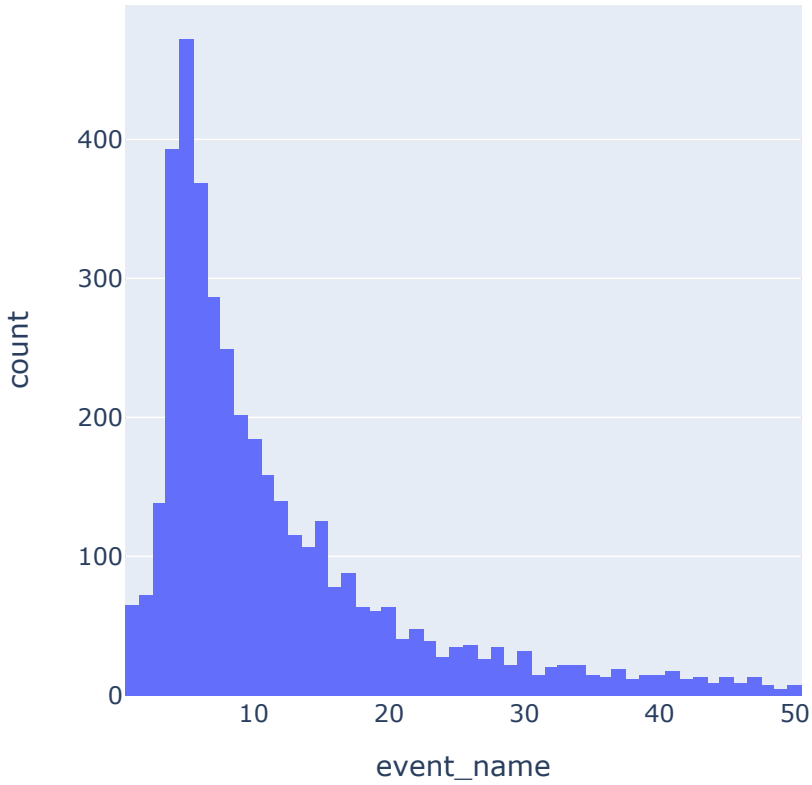
С одной стороны мы имеем 1% пользователей, которые совершают очень много действий. С другой стороны, данные пользователи совершают примерно 20(!)% целевых действий от общего количества. Не будем удалять данных пользователей

In [31]:
```
len(df.groupby('user_id', as_index=False)['event_name']
                      .count().query('event_name <= 50'))
```

Out[31]:
```
4034
```

Большинство пользователей совершает 0-50 действий. Таких пользователей 4034 из 4251
Скорректирую и выведу на графике только таких пользователей

In [32]:
```
# построим гистограмму по кол-во действий на пользователя с учетом фильтра
fig_04 = px.histogram(df.groupby('user_id', as_index=False)['event_name']
                      .count().query('event_name <= 50'), x="event_name",
                        title='Количество действий на каждого пользователя'
                       )
fig_04.update_layout(title_x=0.5)

fig_04.show()
```



Большинство пользователей совершает по 5 действий (472 пользователя). Также у нас есть 65 пользователей, которые совершают лишь 1 действие. Пользователей, которые совершают лишь 2 действия 72.
Посмотрим на пользователей, которые совершают одно действие и на то, какое именно действие они совершают

In [33]:
```
(
    df[df['user_id'].isin
       (df.groupby('user_id', as_index=False)['event_name'].count()
             .query('event_name == 1')['user_id'].unique())]
    ['event_name'].value_counts()
)
```

Out[33]:
```
tips_show        30
map              16
search            8
contacts_show     6
photos_show       4
advert_open       1
Name: event_name, dtype: int64
```

Большинство этих пользоватлей лишь просматривают объявление. Некоторые смотрят карту с объявлениями или пользуются поиском.
6 человек смотрят контакты, что является нашим целевым действием. Могу предположить, что данные пользователи, например, открыли определенное объявление до того момента, за который мы располагаем данными. А через какое-то время вернулись в приложение, которое работало в фоне, и решили открыть контакт

Количество пользователей по каждому источнику

Посмотрим на количество пользователей, которые пришли из разных источников

In [34]:
```
(
    df.groupby('source', as_index=False)['user_id'].nunique()
    .assign(source_shape=lambda x: x['user_id'] / x['user_id'].sum())
)
```

Out[34]:

	source	user_id	source_shape
0	google	1129	0.262986
1	other	1230	0.286513
2	yandex	1934	0.450501

Большинство пользователей переходит в приложение из Яндекса (1934 пользователя (45%)). Пользователей из Гугла и "других" источников примерно одинаково

Количество активных дней на пользователя

In [35]:
```
(
    df.groupby('user_id', as_index=False)
    ['event_date'].nunique().sort_values('event_date', ascending=False)
)
```

Out[35]:

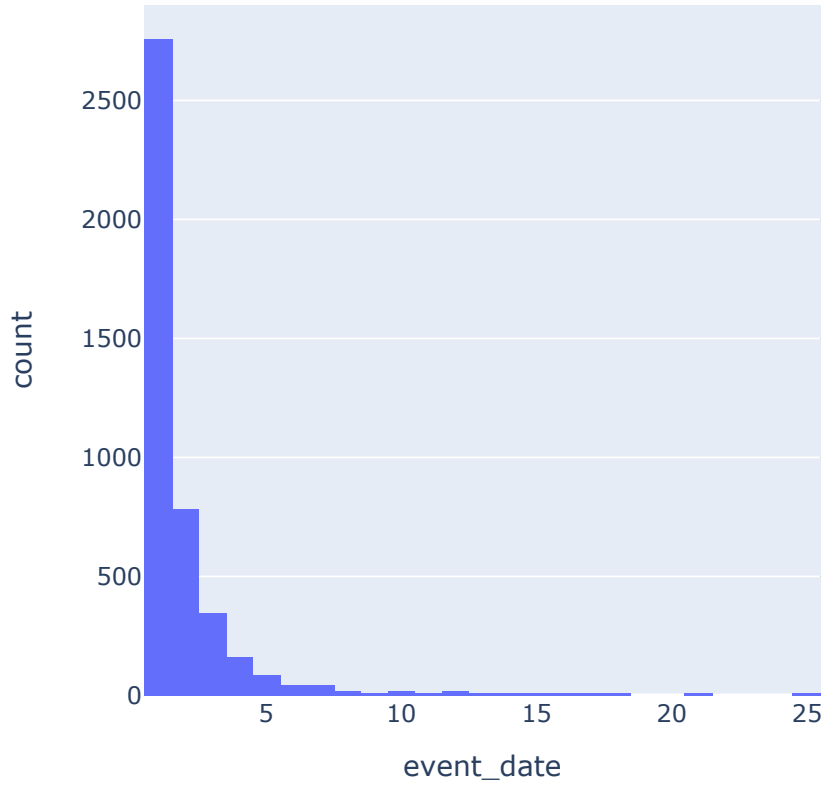
	user_id	event_date
615	21230dd9-2f7f-4b77-a436-43d4d10388e0	25
2103	7e797355-265c-4997-ba47-2258e06d3c66	21
235	0d9e5bb7-0ad6-4b62-a118-b1e4f5b31dfa	21
401	1580911b-65db-4f1a-be7e-1ca39becac30	18
1668	6383ff6a-04b8-4562-a98f-bb4f760d3c39	18
...	...	...
2368	8bf8f713-99bb-408e-ac73-fdc1b2357e30	1
2369	8c227867-4ed5-457d-a7d2-bb0e5b7374d4	1
2370	8c24ccb9-5f01-49c8-b8f3-da3319e3dc59	1
1003	3a9e494e-2cfa-4746-bb7d-659b177c986f	1
2146	80a9887a-d45f-44a2-9473-1446f68b9c16	1

4293 rows × 2 columns

Визуализируем данные для наглядности

In [36]:
```
# построим гистограмму по кол-во действий на пользователя с учетом фильтра
fig_05 = px.histogram(df.groupby('user_id', as_index=False)
                      ['event_date'].nunique()
                      .sort_values('event_date', ascending=False), x="event_date",
                         title='Количество активных дней для каждого пользователя'
                     )
fig_05.update_layout(title_x=0.5)
fig_05.show()
```

Количество активных дней для каждого пользователя



In [37]:
```
df.groupby('user_id', as_index=False)['event_date'].nunique().describe()
```

Out[37]:

	event_date
count	4293.000000
mean	1.820871
std	1.762537
min	1.000000
25%	1.000000
50%	1.000000
75%	2.000000
max	25.000000

Здесь также можно обратить внимание на явные выбросы. Посмотрим на них повнимательнее

In [38]:
```
np.percentile(df.groupby('user_id', as_index=False)
    ['event_date'].nunique()['event_date'],
                [90, 95, 99])
```

Out[38]:
```
array([ 3.,  5., 10.])
```

Очень хочется удалить данные, в которых пользователи имеют больше 10 активынх дней. Но не будем забывать про целевое действие. Посмотрим на действия, которые совершают данные пользователи.

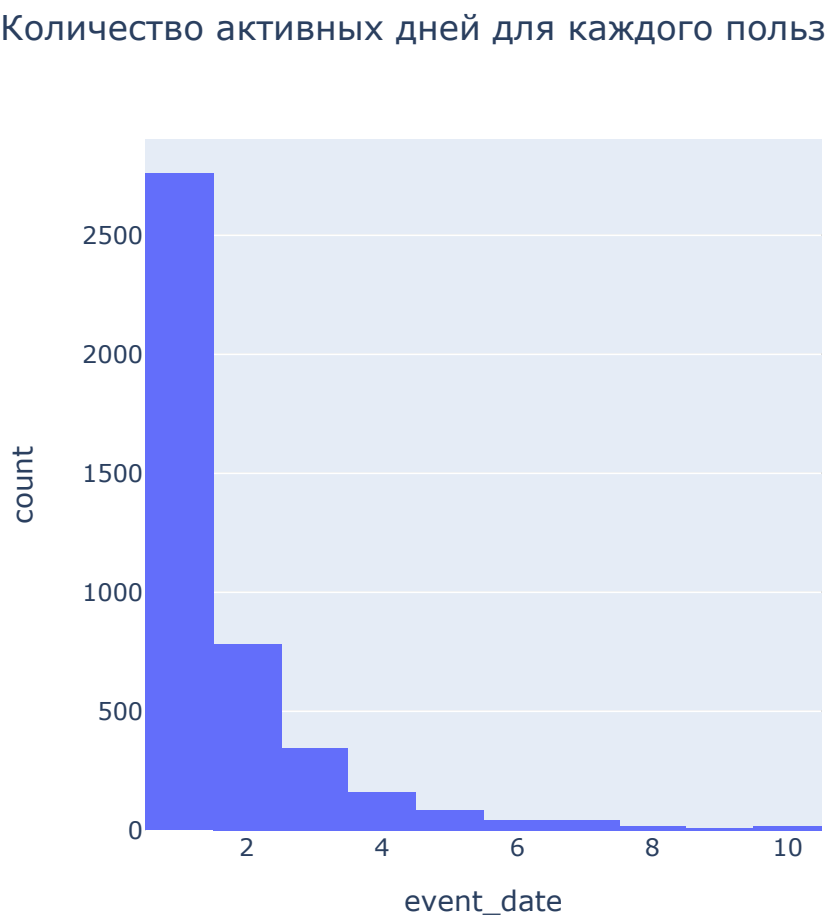
In [39]:
```
df[(df['user_id'].isin(
```

```
    df.groupby('user_id', as_index=False)['event_date']
    .nunique().query('event_date > 10')['user_id'].unique())][['event_name'].value_co
```

Out[39]:
```
tips_show        3563
photos_show       847
contacts_show     828
map               405
search            397
advert_open       366
favorites_add      67
tips_click         60
contacts_call      16
Name: event_name, dtype: int64
```

Здесь так же 1% пользователей совершает 20% целевых действий. Пока не будем удалять данных пользователей

In [40]:
```
# построим гистограмму по кол-во действий на пользователя с учетом фильтра
fig_06 = px.histogram(df.groupby('user_id', as_index=False)
                      ['event_date'].nunique().query('event_date <= 10'), x="event_da
                      title='Количество активных дней для каждого пользователя'
                     )
fig_06.update_layout(title_x=0.5)

fig_06.show()
```



Подавляющее большинство пользователей заходит в приложение в 1 день. 2756 пользователей из 4293. Также достаточно весомая часть пользователей заходит в приложение 2-3 дня (786 и 346) пользователя соответсвенно

Сессии пользователей

Рассчитаем номер сессии для каждого пользователя. Будем считать, что сессии разные, если между ними прошло более 20 минут.
Время 20 минут выбрано неспроста. Считаю, что паузу в 20 минут между сессиями можно объяснить, например, звонков продавцу или поиском сторонней информации по объявлению.

In [41]:
```
# Отсортируем данные
df = df.sort_values(['user_id', 'event_time'])

# Разница между
diff_timestamp = df.groupby('user_id')['event_time'].diff()

# Зададим правило для новой сессии (более 20 минут)
new_session = (diff_timestamp.isnull()) | (diff_timestamp > timedelta(minutes=20))

# Создадим session_id
df['session_id'] = df.loc[new_session, ['user_id', 'event_time']] \
    .groupby('user_id').rank(method='first').astype(int)

# Заменим неверные значения
df['session_id'] = df['session_id'].fillna(method='ffill').astype(int)
```

In [42]:
```
df.head()
```

Out[42]:

	user_id	source	event_time	event_name	event_date	time_of_day	contact_show	se
2171	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other	2019-10-07 13:39:45.989359	tips_show	2019-10-07	afternoon	False	
2172	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other	2019-10-07 13:40:31.052909	tips_show	2019-10-07	afternoon	False	
2173	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other	2019-10-07 13:41:05.722489	tips_show	2019-10-07	afternoon	False	
2174	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other	2019-10-07 13:43:20.735461	tips_show	2019-10-07	afternoon	False	
2175	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other	2019-10-07 13:45:30.917502	tips_show	2019-10-07	afternoon	False	

Теперь посчитаем кол-во сессий для кажого пользователя

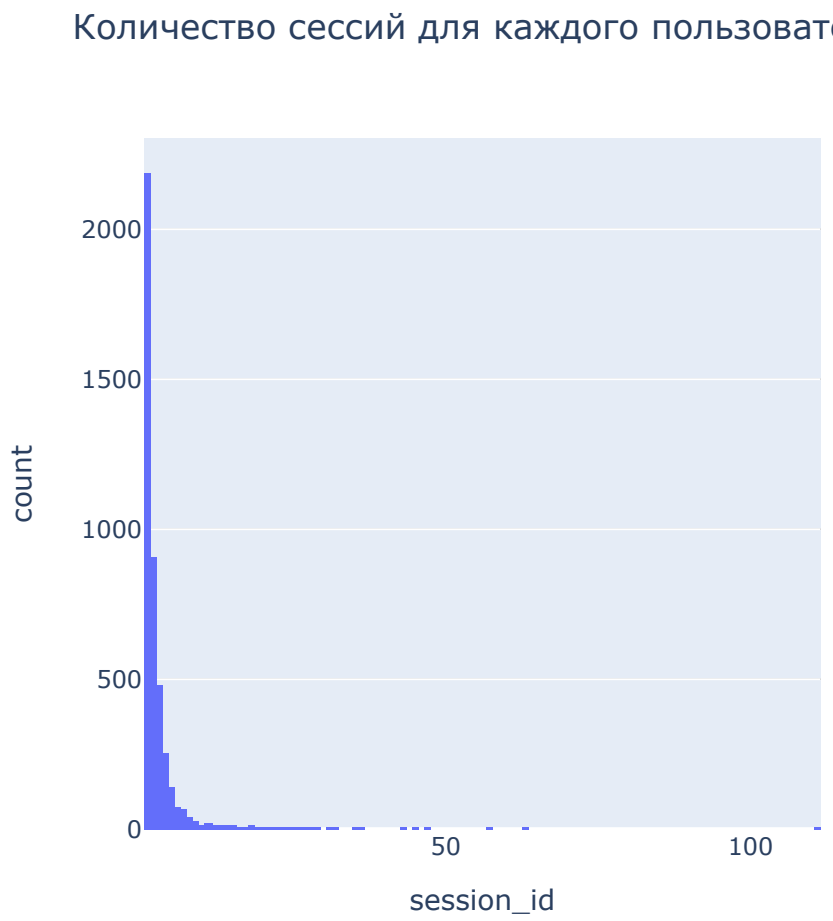
In [43]:
```
(
    df.groupby('user_id', as_index=False)['session_id'].nunique()
)
```

Out[43]:

	user_id	session_id
0	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	4
1	00157779-810c-4498-9e05-a1e9e3cedf93	8
2	00463033-5717-4bf1-91b4-09183923b9df	1
3	004690c3-5a84-4bb7-a8af-e0c8f8fca64e	8
4	00551e79-152e-4441-9cf7-565d7eb04090	3
...	...	...
4288	ffab8d8a-30bb-424a-a3ab-0b63ebbf7b07	2
4289	ffc01466-fdb1-4460-ae94-e800f52eb136	1
4290	ffcf50d9-293c-4254-8243-4890b030b238	1
4291	ffe68f10-e48e-470e-be9b-eeb93128ff1a	3
4292	fffb9e79-b927-4dbb-9b48-7fd09b23a62b	32

4293 rows × 2 columns

In [44]:
```
# Визуализируем
fig_07 = px.histogram(df.groupby('user_id', as_index=False)
                      ['session_id'].nunique(), x="session_id",
                     title='Количество сессий для каждого пользователя'
                     )
fig_07.update_layout(title_x=0.5)

fig_07.show()
```



In [45]:
```
df.groupby('user_id', as_index=False)['session_id'].nunique().describe()
```

Out[45]:

	session_id
count	4293.000000
mean	2.556487
std	3.885942
min	1.000000
25%	1.000000
50%	1.000000
75%	3.000000
max	111.000000

Есть пользователи, у которых более 100 сессий. Посмотрим на процентили.

In [46]:
```
np.percentile(df.groupby('user_id', as_index=False)['session_id'].nunique()['session_
```

Out[46]:
```
array([ 5.,  7., 18.])
```

У 99% пользователей 18 или меньше сессий. Посмотрим на 1% пользователей, у которых кол-во сессий больше 18, и на то, какие действия они совершают

In [47]:
```
df[(df['user_id'].isin(
    df.groupby('user_id', as_index=False)['session_id']
    .nunique().query('session_id > 18')['user_id'].unique()))]['event_name'].value_co
```

Out[47]:
```
tips_show        3405
photos_show       963
contacts_show     773
search            564
advert_open       478
map               395
favorites_add      68
tips_click         59
contacts_call      18
Name: event_name, dtype: int64
```

Данный 1% пользователей совершает 773 из 4529 просмотров контактов. Не буду удалять эти данные.

Теперь рассмотрим еще один очень важный показатель - продолжительность сессий для каждого пользователя.

In [48]:
```
time_by_session = (
    df.groupby(['user_id', 'session_id'], as_index=False)['event_time']
    .agg(['first', 'last', 'count'])
    .reset_index()
    .rename(columns={'first': 'session_start', 'last': 'session_end' ,'count': 'event
    .assign(session_duration=lambda x: ((x['session_end'] - x['session_start']).dt.se
)
time_by_session
```

Out[48]:

	user_id	session_id	session_start	session_end	event_per_session	session_durati
0	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	1	2019-10-07 13:39:45.989359	2019-10-07 13:49:41.716617	9	9.9166
1	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	2	2019-10-09 18:33:55.577963	2019-10-09 18:42:22.963948	4	8.4500
2	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	3	2019-10-21 19:52:30.778932	2019-10-21 20:07:30.051028	14	14.9833
3	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	4	2019-10-22 11:18:14.635436	2019-10-22 11:30:52.807203	8	12.6333
4	00157779-810c-4498-9e05-a1e9e3cedf93	1	2019-10-19 21:34:33.849769	2019-10-19 21:59:54.637098	9	25.3333
...	...	...	...	...	...	
10970	fffb9e79-b927-4dbb-9b48-7fd09b23a62b	28	2019-11-02 01:16:48.947231	2019-11-02 01:16:48.947231	1	0.0000
10971	fffb9e79-b927-4dbb-9b48-7fd09b23a62b	29	2019-11-02 18:01:27.094834	2019-11-02 18:17:41.386651	2	16.2333
10972	fffb9e79-b927-4dbb-9b48-7fd09b23a62b	30	2019-11-02 19:25:53.794029	2019-11-02 19:30:50.471310	4	4.9333
10973	fffb9e79-b927-4dbb-9b48-7fd09b23a62b	31	2019-11-03 14:32:55.956301	2019-11-03 14:48:44.263356	15	15.8000
10974	fffb9e79-b927-4dbb-9b48-7fd09b23a62b	32	2019-11-03 15:36:01.007440	2019-11-03 16:08:25.388712	14	32.4000

10975 rows × 6 columns

In [49]:

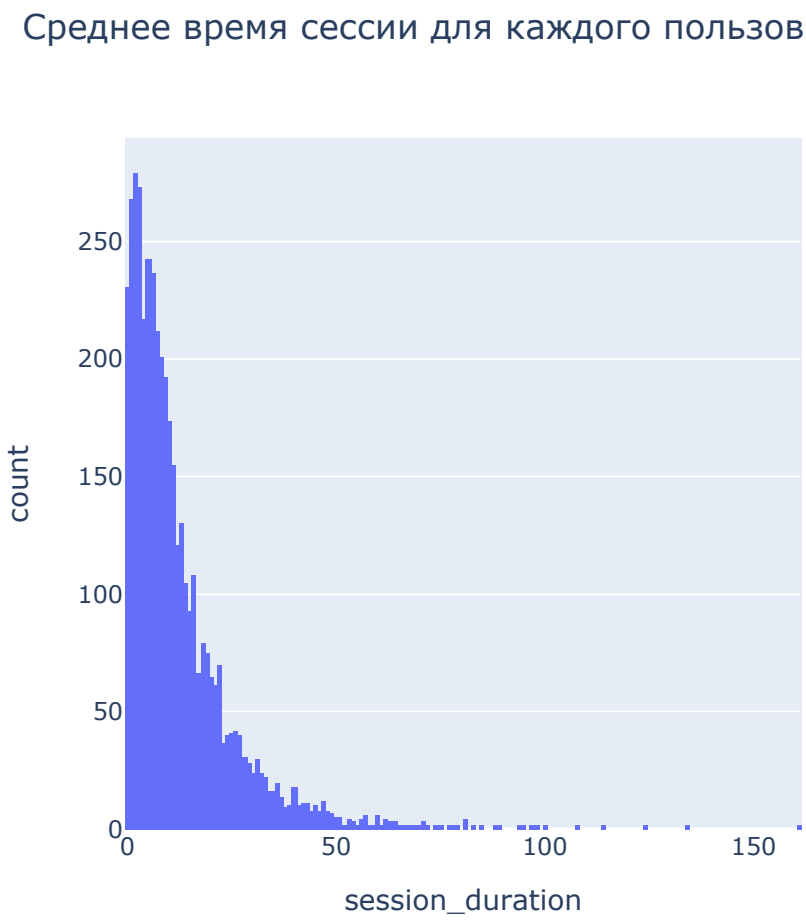
```
# Посчитаем среднее время сессии для каждого пользователя
time_by_session.groupby('user_id', as_index=False)['session_duration'].mean()
```

Out[49]:

	user_id	session_duration
0	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	11.495833
1	00157779-810c-4498-9e05-a1e9e3cedf93	18.241667
2	00463033-5717-4bf1-91b4-09183923b9df	24.700000
3	004690c3-5a84-4bb7-a8af-e0c8f8fca64e	7.883333
4	00551e79-152e-4441-9cf7-565d7eb04090	3.105556
...	...	...
4288	ffab8d8a-30bb-424a-a3ab-0b63ebbf7b07	24.708333
4289	ffc01466-fdb1-4460-ae94-e800f52eb136	0.866667
4290	ffcf50d9-293c-4254-8243-4890b030b238	1.333333
4291	ffe68f10-e48e-470e-be9b-eeb93128ff1a	12.950000
4292	fffb9e79-b927-4dbb-9b48-7fd09b23a62b	15.505729

4293 rows × 2 columns

In [50]:
```
# посмотрим на распределение
fig_08 = px.histogram(time_by_session.groupby('user_id', as_index=False)
                      ['session_duration'].mean(), x="session_duration",
                     title='Среднее время сессии для каждого пользователя'
                     )
fig_08.update_layout(title_x=0.5)

fig_08.show()
```



In [51]:
```
time_by_session.groupby('user_id', as_index=False)['session_duration'].mean().describ
```

Out[51]:

	session_duration
count	4293.000000
mean	12.318067
std	13.264624
min	0.000000
25%	3.605556
50%	8.362500
75%	16.250000
max	160.966667

Больше всего пользователей со средним временем сессии 1,5 - 2,5 минуты. Также можно заметить, что большинство пользователей имеют среднее время сессии до 20 минут.

Подведем итоги исследовательского анализа данных

- Среднее кол-во действий в день около 2600. Меньше всего действий было 12 октября (1843)Б а больше всего 23 октября (3361)
- В среднем в день приложением пользуется около 280 уникальных пользователей. Меньше всего пользователей на сайте было 12 октября (178), а больше всего 23 октября(352)
- Возможно, 12 октября приложение работало не стабильно
- Больше всего действий соврешается днем и вечером. Ночью действия почти не совершаются
- В среднем каждый пользователь совершил 17 действий. Медиана на отметке 9. Один из пользователей совершил аж 478 действий
- Из Яндекса приходит больше всего пользователей (45%). Из остальных источников приходит примерно одинкаовое кол-во пользователей
- Среднее кол-во активных дней для всех пользователей - 2. Максимальное кол-во активных дней на пользователя - 25
- Среднее кол-во сессий на каждого пользователя - 2,5. Средняя продолжительность сессии - 12 (медиана 8)

Сегментация пользователей и расчет метрик

Цель данной задачи состоит в сегментации пользователей на основе их поведения для дальнейшнего влияния на пользователя с целью удержания и повышения конверсии.
Я предлагаю разделить пользователей на две группы по средней длительности сессии. Считаю, что данное разделение может быть полезно для бизнеса, тк это позволит изучить поведение данных пользователей и даст возможность изучить точки воздействия на такие группы с целью повышения прибыли. Например, в ходе анализа мы будем рассчитывать удержание и конверсию пользователей по группам. Для начала можно предположить, что пользователи с короткими сессиями чаще заходят и возвращаются в приложение, а пользователи с длинными сессиями чаще совершают целевое действие, тк дольше ищут подходящее объявление

In [52]:
```
# Посмотрим на среднее время сессий для пользователя
avg_session_time =(
    time_by_session.groupby('user_id', as_index=False)['session_duration'].mean()
)
avg_session_time
```

Out[52]:

	user_id	session_duration
0	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	11.495833
1	00157779-810c-4498-9e05-a1e9e3cedf93	18.241667
2	00463033-5717-4bf1-91b4-09183923b9df	24.700000
3	004690c3-5a84-4bb7-a8af-e0c8f8fca64e	7.883333
4	00551e79-152e-4441-9cf7-565d7eb04090	3.105556
...	...	...
4288	ffab8d8a-30bb-424a-a3ab-0b63ebbf7b07	24.708333
4289	ffc01466-fdb1-4460-ae94-e800f52eb136	0.866667
4290	ffcf50d9-293c-4254-8243-4890b030b238	1.333333
4291	ffe68f10-e48e-470e-be9b-eeb93128ff1a	12.950000
4292	fffb9e79-b927-4dbb-9b48-7fd09b23a62b	15.505729

4293 rows × 2 columns

Разделим пользователей на 2 ровные группы по среднему времени сессии. Для этого применим функцию `qcut` которая делит данные на ровные части в зависисмости от выбранного параметра. В данном случае мы будем делить данные пополам (то есть по медиане)

Комментарии от тимлида ✔ :
Здорово, что знаешь и применяешь подобный инструмент

In [53]:
```
pd.qcut(avg_session_time['session_duration'],q=2)
```

Out[53]:
```
0       (8.362, 160.967]
1       (8.362, 160.967]
2       (8.362, 160.967]
3        (-0.001, 8.362]
4        (-0.001, 8.362]
              ...
4288    (8.362, 160.967]
4289     (-0.001, 8.362]
4290     (-0.001, 8.362]
4291    (8.362, 160.967]
4292    (8.362, 160.967]
Name: session_duration, Length: 4293, dtype: category
Categories (2, interval[float64, right]): [(-0.001, 8.362] < (8.362, 160.967]]
```

Итого у нас получится 2 группы:

- **Группа A (время сессии от 0 до 8.362)**
- **Группа B (время сессии от 8.362)**

In [54]:
```
avg_session_time['group'] = pd.qcut(avg_session_time['session_duration'],
                                    q=[0, .5, 1],
                                    labels=['A', 'B'])
avg_session_time.head()
```

Out[54]:

	user_id	session_duration	group
0	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	11.495833	B
1	00157779-810c-4498-9e05-a1e9e3cedf93	18.241667	B
2	00463033-5717-4bf1-91b4-09183923b9df	24.700000	B
3	004690c3-5a84-4bb7-a8af-e0c8f8fca64e	7.883333	A
4	00551e79-152e-4441-9cf7-565d7eb04090	3.105556	A

Присоединим информацию о группе в общий датафрейм

In [55]:
```
df = df.merge(avg_session_time[['user_id', 'group']], on='user_id', how='inner')
```

In [56]:
```
display(df.head())
df.groupby('group', as_index=False)['user_id'].nunique()
```

	user_id	source	event_time	event_name	event_date	time_of_day	contact_show	sessi
0	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other	2019-10-07 13:39:45.989359	tips_show	2019-10-07	afternoon	False	
1	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other	2019-10-07 13:40:31.052909	tips_show	2019-10-07	afternoon	False	
2	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other	2019-10-07 13:41:05.722489	tips_show	2019-10-07	afternoon	False	
3	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other	2019-10-07 13:43:20.735461	tips_show	2019-10-07	afternoon	False	
4	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other	2019-10-07 13:45:30.917502	tips_show	2019-10-07	afternoon	False	

Out[56]:

	group	user_id
0	A	2147
1	B	2146

В итоге получили 2 группы. В группе A - 2147 пользователей, в группе B - 2146.
Теперь расчитаем метрики для этих двух групп.

Расчет метрик

Время в приложении и частота действий

Напомню, что нам необходимо сравнить две группы пользователей по следуюшим метрикам:

1. retention rate,
2. время, проведённое в приложении,
3. частота действий,
4. конверсия в целевое действие — просмотр контактов.

Начну с расчета времени проведенного в приложении и частоты действий. За частоту действий примем среднее кол-во действий за сессию для каждой группы.

In [57]:
```
# добавим информацию о группе пользователей в датафрейм с временем сессий
time_by_session = time_by_session.merge(avg_session_time[['user_id', 'group']], on='u
```

In [58]:
```
(
    time_by_session.groupby('group', as_index=False)
    .agg({'session_id': 'count', 'event_per_session': 'sum','session_duration': 'sum'
    .rename(columns={'session_id': 'session_count','event_per_session': 'events_count
    .assign(session_duration_hours=lambda x: round(x['session_duration']/60,2))
    .assign(avg_event_per_session=lambda x: round(x['events_count']/x['session_count'
    .assign(avg_session_duration=lambda x: round(x['session_duration']/x['session_cou
)
```

Out[58]:

	group	session_count	events_count	session_duration	session_duration_hours	avg_event_per_sessi
0	A	5913	22161	23357.983333	389.30	3.
1	B	5062	52036	95224.966667	1587.08	10.

Хоть в группе A среднее время почти в 6 раз меньше, кол-во сессий по группам различается не так сильно, хоть и достаточно. В группе A 5913 сессий, а в группе B 5062. Кол-во событий в группе B более чем в 2 раза больше, чем в группе A. Среднее время сессии в группе A почти 4 минуты, а в группе B почти 19 минут.

1. **Время, проведенное в приложении:** A - 389 часов, B - 1587 часов
2. **Частота действий:** A - 3,75 действий за сессию, B - 10,28 действий за сессию

Retention rate и конверсия

Для рассчета RR сначала подготовим данные. Создадим датафрейм с информацией по сессиям. В целом, большинство информации уже есть в `time_by_session`. Добавим туда информацию о первом визите для пользователя и лайфтайм

In [59]:
```
# создадим датафрейм копией
sessions = time_by_session.copy()

# добавим информацию о первом визите
first_visit = df.groupby('user_id').agg({'event_time':'first'}).reset_index()
first_visit.rename(columns={'event_time':'first_visit'}, inplace=True)

# добавим информацию о первом визите
sessions = sessions.merge(first_visit, on='user_id', how='left')
sessions['dt'] = sessions['first_visit'].dt.date

# посчитаем лайфтайм
sessions['lifetime'] = (sessions['session_start'] - sessions['first_visit']).dt.days


sessions.head()
```

Out[59]:

	user_id	session_id	session_start	session_end	event_per_session	session_duration	g
0	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	1	2019-10-07 13:39:45.989359	2019-10-07 13:49:41.716617	9	9.916667	
1	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	2	2019-10-09 18:33:55.577963	2019-10-09 18:42:22.963948	4	8.450000	
2	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	3	2019-10-21 19:52:30.778932	2019-10-21 20:07:30.051028	14	14.983333	
3	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	4	2019-10-22 11:18:14.635436	2019-10-22 11:30:52.807203	8	12.633333	
4	00157779-810c-4498-9e05-a1e9e3cedf93	1	2019-10-19 21:34:33.849769	2019-10-19 21:59:54.637098	9	25.333333	

Также соберем профили пользователей. Для начала соберем пользователей, которые совершили целевое действие

In [60]:
```
# Датафрейм пользователей, которые совершили целевое действие — просмотр контаков
users_contact_show = df[df['event_name'] == 'contacts_show']
```

In [61]:
```
def get_profiles(sessions):

    # сортируем сессии по ID пользователя и дате посещения
    # группируем по ID и находим первые значения session_start и channel
    # столбец с временем первого посещения назовём first_ts
    # от англ. first timestamp — первая временная отметка
    profiles = (
        df.sort_values(by=['user_id', 'event_time'])
        .groupby('user_id')
        .agg({'event_time': 'first', 'source': 'first'})
        .rename(columns={'event_time': 'first_ts'})
        .reset_index()  # возвращаем user_id из индекса
    )

    # определяем дату первого посещения
    profiles['dt'] = profiles['first_ts'].dt.date



    # определим смотрел ли пользователь контакты
    profiles['contact_show'] = profiles['user_id'].isin(users_contact_show['user_id'])

    # добавим информацию о группе
    profiles = profiles.merge(df[['user_id', 'group']], on='user_id', how='inner')
    profiles = profiles.drop_duplicates()

    return profiles
```

In [62]:
```
profiles = get_profiles(df)
profiles
```

Out[62]:

	user_id	first_ts	source	dt	contact_show	group
0	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	2019-10-07 13:39:45.989359	other	2019-10-07	False	B
35	00157779-810c-4498-9e05-a1e9e3cedf93	2019-10-19 21:34:33.849769	yandex	2019-10-19	True	B
106	00463033-5717-4bf1-91b4-09183923b9df	2019-11-01 13:54:35.385028	yandex	2019-11-01	False	B
116	004690c3-5a84-4bb7-a8af-e0c8f8fca64e	2019-10-18 22:14:05.555052	google	2019-10-18	False	A
148	00551e79-152e-4441-9cf7-565d7eb04090	2019-10-25 16:44:41.263364	yandex	2019-10-25	True	A
...	...	...	...	...	...	...
73855	ffab8d8a-30bb-424a-a3ab-0b63ebbf7b07	2019-10-13 16:11:27.414960	yandex	2019-10-13	False	B
73872	ffc01466-fdb1-4460-ae94-e800f52eb136	2019-10-07 20:32:49.997044	yandex	2019-10-07	True	A
73879	ffcf50d9-293c-4254-8243-4890b030b238	2019-10-23 11:51:35.199237	google	2019-10-23	False	A
73881	ffe68f10-e48e-470e-be9b-eeb93128ff1a	2019-10-21 16:39:33.867145	yandex	2019-10-21	True	B
73894	fffb9e79-b927-4dbb-9b48-7fd09b23a62b	2019-10-12 00:57:21.241896	google	2019-10-12	True	B

4293 rows × 6 columns

In [63]:

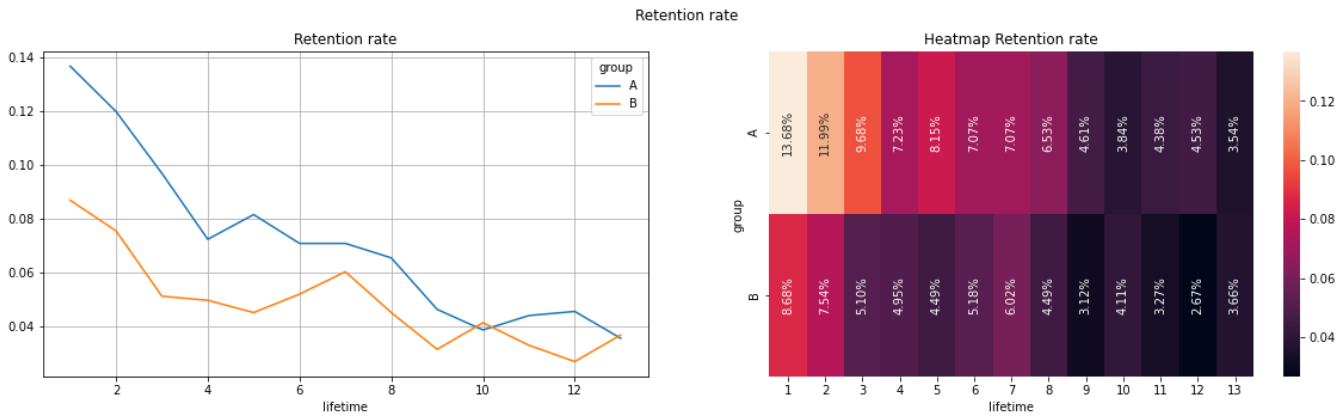
```
def get_retention(
    profiles,
    sessions,
    observation_date,
    horizon_days,
    dimensions=[],
    ignore_horizon=False,
):


    # исключаем пользователей, не «доживших» до горизонта анализа
    last_suitable_acquisition_date = observation_date
    if not ignore_horizon:
        last_suitable_acquisition_date = observation_date - timedelta(
            days=horizon_days - 1
        )
    result_raw = profiles.query('dt <= @last_suitable_acquisition_date')

    # собираем «сырые» данные для расчёта удержания
    result_raw = result_raw.merge(
        sessions[['user_id', 'session_start']], on='user_id', how='left'
    )
    result_raw['lifetime'] = (
        result_raw['session_start'] - result_raw['first_ts']
    ).dt.days

    # функция для группировки таблицы по желаемым признакам
    def group_by_dimensions(df, dims, horizon_days):
        result = df.pivot_table(
            index=dims, columns='lifetime', values='user_id', aggfunc='nunique'
        )
        cohort_sizes = (
            df.groupby(dims)
            .agg({'user_id': 'nunique'})
```

```
            .rename(columns={'user_id': 'cohort_size'})
        )
        result = cohort_sizes.merge(result, on=dims, how='left').fillna(0)
        result = result.div(result['cohort_size'], axis=0)
        result = result[['cohort_size'] + list(range(horizon_days))]
        result['cohort_size'] = cohort_sizes
        return result

    # получаем таблицу удержания
    result_grouped = group_by_dimensions(result_raw, dimensions, horizon_days)

    # получаем таблицу динамики удержания
    result_in_time = group_by_dimensions(
        result_raw, dimensions + ['dt'], horizon_days
    )

    # возвращаем обе таблицы и сырые данные
    return result_raw, result_grouped, result_in_time
```

Момент анализа возьмем 4 ноября, а горизонт определим на 2 недели. Это позволит более детально изучить метрику

In [64]:
```
# Удержание
dimensions = ['group']
retention_raw, retention_grouped, retention_history = get_retention(
    profiles, sessions, datetime(2019, 11, 4).date(), 14, dimensions=dimensions)
```

In [65]:
```
# Построение гистрограмм
fig, axs = plt.subplots(1,2)
fig.suptitle("Retention rate")
fig.set_figheight(5)
fig.set_figwidth(20)
retention_grouped.drop(columns=['cohort_size', 0]).T.plot(ax=axs[0], grid=True)
axs[0].set_title('Retention rate')
axs[0].set_xlabel('lifetime')
sns.heatmap(retention_grouped.drop(columns=['cohort_size', 0]), annot=True,   fmt='.2
           annot_kws={'rotation':"vertical"})
axs[1].set_title('Heatmap Retention rate')
axs[1].set_xlabel('lifetime');
```



По графикам отчетливо видно, что коэф. удержания у группы А выше, чем у группы В. Особенно хорошо это видно в первые дни "жизни". Также хорошо заметно, что удержание резко снижается в первые 4 дня, а далее снижается более плавно

In [66]:
```
# Добавим номер недели в профили пользователей
profiles['week'] = profiles['first_ts'].dt.isocalendar().week
```

In [67]:
```
# Собираем таблицу удержания по неделям
result_raw_week = profiles.merge(
    sessions[['user_id', 'session_start']], on='user_id', how='left'
```

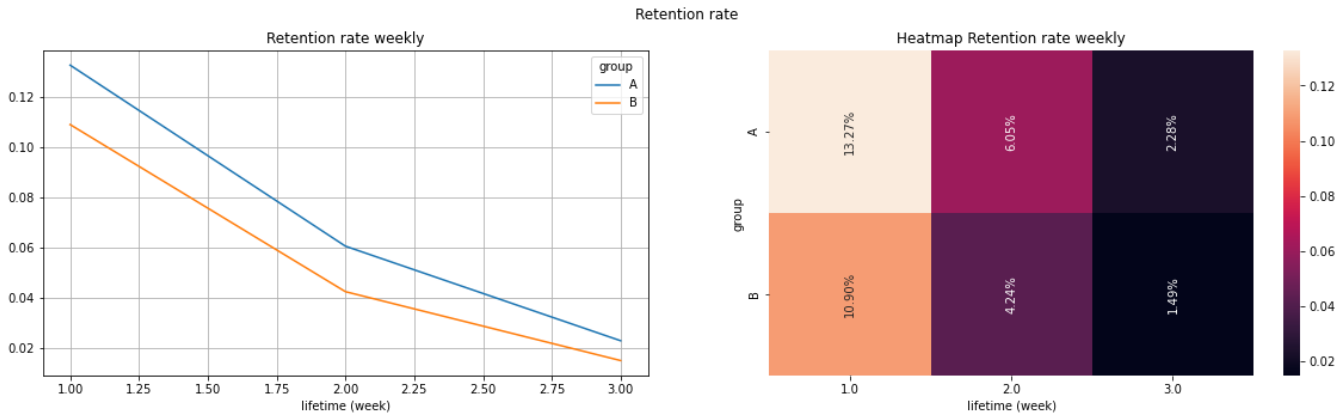
```
)


result_raw_week['lifetime_week'] = np.floor((
    result_raw_week['session_start'] - result_raw_week['first_ts']
).dt.days/7)

# строим таблицу удержания
result_grouped_week = result_raw_week.pivot_table(
    index=['group'], columns='lifetime_week', values='user_id', aggfunc='nunique'
)


# вычисляем размеры когорт
cohort_sizes_week = (
    result_raw_week.groupby(['group'])
    .agg({'user_id': 'nunique'})
    .rename(columns={'user_id': 'cohort_size'})
)


# объединяем размеры когорт и таблицу удержания
result_grouped_week = cohort_sizes_week.merge(
    result_grouped_week, on=['group'], how='left'
).fillna(0)


# делим данные таблицы удержания на размеры когорт
result_grouped_week = result_grouped_week.div(
    result_grouped_week['cohort_size'], axis=0
)

# возвращаем размер когорт
result_grouped_week['cohort_size'] = cohort_sizes_week

result_grouped_week
```

Out[67]:

group	cohort_size	0.0	1.0	2.0	3.0
A	2147	1.0	0.132743	0.060550	0.022823
B	2146	1.0	0.109040	0.042404	0.014911

In [68]:

```
# Построение гистрограмм
fig, axs = plt.subplots(1,2)
fig.suptitle("Retention rate")
fig.set_figheight(5)
fig.set_figwidth(20)
result_grouped_week.drop(columns=['cohort_size', 0]).T.plot(ax=axs[0], grid=True)
axs[0].set_title('Retention rate weekly')
axs[0].set_xlabel('lifetime (week)')
sns.heatmap(result_grouped_week.drop(columns=['cohort_size', 0]), annot=True,   fmt='
            annot_kws={'rotation':"vertical"})
axs[1].set_title('Heatmap Retention rate weekly')
axs[1].set_xlabel('lifetime (week)');
```



По графикам отчетливо видно, что коэф. удержания у группы А выше, чем у группы В. Особенно хорошо это видно в первые дни "жизни". Также хорошо заметно, что удержание резко снижается в первые 4 дня, а далее снижается более плавно

Понедельный коэф. удержания так же выше у группы А. Однако у обоих групп удержание падает почти вдвое во вторую неделю жизни. Большинство пользоателей приложения не заходят повторно после первой недели использования.

Для рассчета конверсии сначала создадим датафрейм, в котором для каждого пользователя укажем время совершения целевого действия

In [69]:
```
show_contact = df[df['event_name'] == 'contacts_show'][['user_id', 'event_time']]
```

In [70]:
```
show_contact
```

Out[70]:

	user_id	event_time
50	00157779-810c-4498-9e05-a1e9e3cedf93	2019-10-20 19:17:18.659799
54	00157779-810c-4498-9e05-a1e9e3cedf93	2019-10-20 19:23:11.839947
56	00157779-810c-4498-9e05-a1e9e3cedf93	2019-10-20 19:30:31.912891
60	00157779-810c-4498-9e05-a1e9e3cedf93	2019-10-20 20:04:53.349091
78	00157779-810c-4498-9e05-a1e9e3cedf93	2019-10-29 21:26:40.258472
...	...	...
74173	fffb9e79-b927-4dbb-9b48-7fd09b23a62b	2019-11-03 14:38:51.134084
74175	fffb9e79-b927-4dbb-9b48-7fd09b23a62b	2019-11-03 14:41:24.780546
74177	fffb9e79-b927-4dbb-9b48-7fd09b23a62b	2019-11-03 14:42:26.444553
74188	fffb9e79-b927-4dbb-9b48-7fd09b23a62b	2019-11-03 15:48:05.420247
74193	fffb9e79-b927-4dbb-9b48-7fd09b23a62b	2019-11-03 15:51:57.899997

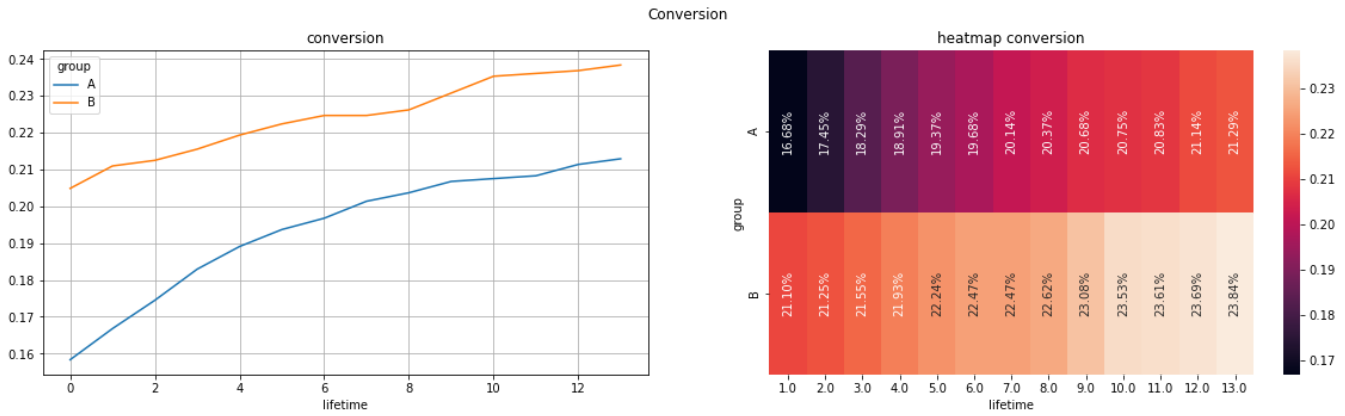
4529 rows × 2 columns

In [71]:
```
# функция для расчёта конверсии

def get_conversion(
    profiles,
    show_contact,
    observation_date,
    horizon_days,
    dimensions=[],
    ignore_horizon=False,
):
```

```
    # исключаем пользователей, не «доживших» до горизонта анализа
    last_suitable_acquisition_date = observation_date
    if not ignore_horizon:
        last_suitable_acquisition_date = observation_date - timedelta(
            days=horizon_days - 1
        )
    result_raw = profiles.query('dt <= @last_suitable_acquisition_date')

    # определяем дату и время первого просмотра контактов для каждого пользователя
    first_show_contact = (
        show_contact.sort_values(by=['user_id', 'event_time'])
        .groupby('user_id')
        .agg({'event_time': 'first'})
        .reset_index()
    )

    # добавляем данные о просмотрах контаков в профили
    result_raw = result_raw.merge(
        first_show_contact[['user_id', 'event_time']], on='user_id', how='left'
    )

    # рассчитываем лайфтайм для каждого просмотра контактов
    result_raw['lifetime'] = (
        result_raw['event_time'] - result_raw['first_ts']
    ).dt.days

    # группируем по cohort, если в dimensions ничего нет
    if len(dimensions) == 0:
        result_raw['cohort'] = 'All users'
        dimensions = dimensions + ['cohort']

    # функция для группировки таблицы по желаемым признакам
    def group_by_dimensions(df, dims, horizon_days):
        result = df.pivot_table(
            index=dims, columns='lifetime', values='user_id', aggfunc='nunique'
        )
        result = result.fillna(0).cumsum(axis = 1)
        cohort_sizes = (
            df.groupby(dims)
            .agg({'user_id': 'nunique'})
            .rename(columns={'user_id': 'cohort_size'})
        )
        result = cohort_sizes.merge(result, on=dims, how='left').fillna(0)
        # делим каждую «ячейку» в строке на размер когорты
        # и получаем conversion rate
        result = result.div(result['cohort_size'], axis=0)
        result = result[['cohort_size'] + list(range(horizon_days))]
        result['cohort_size'] = cohort_sizes
        return result

    # получаем таблицу конверсии
    result_grouped = group_by_dimensions(result_raw, dimensions, horizon_days)

    # для таблицы динамики конверсии убираем 'cohort' из dimensions
    if 'cohort' in dimensions:
        dimensions = []

    # получаем таблицу динамики конверсии
    result_in_time = group_by_dimensions(
        result_raw, dimensions + ['dt'], horizon_days
    )

    # возвращаем обе таблицы и сырые данные
    return result_raw, result_grouped, result_in_time
```

In [72]:
```
# Конверсия
conversion_raw, conversion_grouped, conversion_history = get_conversion(
    profiles, show_contact, datetime(2019, 11, 4).date(), 14, dimensions=dimensions
)
```

In [73]:
```
# Построение гистрограмм
fig, axs = plt.subplots(1,2)
fig.suptitle("Conversion")
fig.set_figheight(5)
fig.set_figwidth(20)
conversion_grouped.drop(columns=['cohort_size']).T.plot(ax=axs[0], grid=True)
axs[0].set_title('conversion')
axs[0].set_xlabel('lifetime')
sns.heatmap(conversion_grouped.drop(columns=['cohort_size', 0]), annot=True,   fmt='.
           annot_kws={'rotation':"vertical"})
axs[1].set_title('heatmap conversion')
axs[1].set_xlabel('lifetime');
```



А вот конверсия у пользователей с бОльшим среднем временем сессии выше и заметно стабильнее по дням жизни. Конверсия для таких пользователей держится в промежутке от 21 до почти 24 процентов. У группы А конверсия в промежутке от 16 до 21 с небольшим процентов

Также можно посчитать конверсию "в лоб". Поделим количество уникальных пользователей, которые совершили целевое действие на общее количество пользователей

In [74]:
```
conv_all = df[df['contact_show'] == True]['user_id'].nunique() / df['user_id'].nuniqu
```

In [75]:
```
# общая конверсия
conv_all = df[df['contact_show'] == True]['user_id'].nunique() / df['user_id'].nuniqu

# конверсия для группа A
conv_a = (
    df[(df['group'] == 'A') & (df['contact_show'] == True)]['user_id'].nunique() /
    df[df['group'] == 'A']['user_id'].nunique())

# конверсия для группы B
conv_b = (
    df[(df['group'] == 'B') & (df['contact_show'] == True)]['user_id'].nunique() /
    df[df['group'] == 'B']['user_id'].nunique())



print(f'Общая конверсия: {conv_all:.2f}')
print(f'Конверсия группы A: {conv_a:.2f}')
print(f'Конверсия группы B: {conv_b:.2f}')
```

```
Общая конверсия: 0.23
Конверсия группы A: 0.22
Конверсия группы B: 0.24
```

По средней конверсии разница уже не столь ощутима. Разница между группами всего 2 процента. Но насколько статистически значима данная разница мы разберемся на следующем этапе

Проверка статистических гипотез

Для начала проверим статистическую гипотезу из задания

Некоторые пользователи установили приложение по ссылке из `yandex`, другие — из `google`. Проверьте гипотезу: две эти группы демонстрируют разную конверсию в просмотры контактов.

- **H0** - Между пользователями, установившими приложение по ссылке из `yandex` `google`, нет сатитистической разницы в конверсии в просмотры в контакты
- **H1** - Статистическая разница есть

Соберем сводник, в котором для каждогшо источника посчитаем кол-во пользователей всего и кол-во пользователей, которые соврешили целевое действие.

In [76]:
```
sources = profiles.groupby('source', as_index=False).agg({'user_id': 'nunique', 'cont
sources
```

Out[76]:

	source	user_id	contact_show
0	google	1129	275
1	other	1230	228
2	yandex	1934	478

Для сравнения двух конверсий из двух выборок я буду испольщовать Z-тест. Кровень значимости установлю 5%

In [77]:
```
alpha = 0.05

stat, pval = proportions_ztest(np.array([sources.loc[0][2],
                                         sources.loc[2][2]]),
                               np.array([sources.loc[0][1],
                                         sources.loc[2][1]]))
print('p-value {0:0.3f}'.format(pval))

if (pval < alpha):
        print("Отвергаем нулевую гипотезу, между выборками есть статистически значимы
else:
        print("Не получилось отвергнуть нулевую гипотезу, статистически значимых разл
```

```
p-value 0.824
Не получилось отвергнуть нулевую гипотезу, статистически значимых различий в выборках нет.
```

С достаточно большой уверенностью можно заявить, что статистически значимых различий в конверсии пользователей из разных источников нет. Можно смело сказать, что с вероятностью в 82% отвергнуть нулевую гипотезу будет ошибкой.

Теперь сформулируем и проверим собственную гипотезу. За основу я возьму группы, на которые мы сегментровали пользователей (не зря же мы это сделали), а сравнивать мы будем также конверсии в целевое действие - просмотр контактов

Между пользователями с разным временем сессий нет статистической разницы в конверсии в просмотры в контакты

- H0 - У пользователей с короткими и длинными сессиями нет значимой разницы в конверсии в целевое действие
- H1 - Статистическая разница есть

In [78]:
```
group_test = profiles.groupby('group', as_index=False).agg({'user_id': 'nunique', 'co
group_test
```

Out[78]:

	group	user_id	contact_show
0	A	2147	466
1	B	2146	515

In [79]:
```
alpha = 0.05

stat, pval = proportions_ztest(np.array([group_test.loc[0][2],
                                         group_test.loc[1][2]]),
                               np.array([group_test.loc[0][1],
                                         group_test.loc[1][1]]))
print('p-value {0:0.3f}'.format(pval))

if (pval < alpha):
        print("Отвергаем нулевую гипотезу, между выборками есть статистически значимы
else:
        print("Не получилось отвергнуть нулевую гипотезу, статистически значимых разл
```

```
p-value 0.074
Не получилось отвергнуть нулевую гипотезу, статистически значимых различий в выборках нет.
```

Здесь мы тоже не можем отвергнуть нулевую гипотезу, но уровень p-значимости значительно ниже, чем в первом тесте - всего около 7,4%

Выводы

В данном проекте мы выполнили достаточно большой объем задач.

- Ознакомились с данными
- Выполнили предобработку
- Проверили исследовательский аналих данных и изучили поведение пользователей
- На основе EDA сегментировали пользователей по среднему времени сессий
- Для сегментов пользователей посчитали основные метрики
- Провели статистические тесты

В конце можно составить обобщенный вывод по проекту В нашем распоряжении были данные о пользовательских сессиях мобильного приложения с частными объявлениями. Всего 4293 уникальных пользователя. В среднем в день приложение пользовались около 280 пользователей, которые совершали в среднем около 2600 различных действий. Большинство действий совершалось днем и вечером. Ночью практически не совершались действия. В среднем каждый пользователь совершил по 17 действий. Большинство пользователей пришло по ссылке из Яндекса. Среднее время сессии 12 минут.
На основе EDA было принято сегментировать пользователей по среднему времени сессии. Можно добавить, что пользователи с короткими сессиями немного лучше удерживаются в приложении. Стоит обратить внимание, на пользователей с большими сессиями и поработать

над увеличение коэф. удержания для них.
Также были проведены два статистических теста, которые показали, что статистически значимых различий между конверсией в просмотры контактов между пользователями из разных источниками и пользователями с разным среднем временем сессии нет.

Рекомендую произвести мероприятия для увеличения коэф. удержания группы с более длинными сессиями. Например, настроить рассылку push-уведомлений для таких клиентов, в которых можно рассказывать о новых объявлениях по последним поискам пользователя. Или напоминание об объявлениях, добавленных в избранное.

Также необходимо поработать над конверсией для пользователей. Как вариант, необходимо проработать систему проверки объявлений на корректность описаний и фотографий. Это повысит просмотр объявлений среди пользователей, и как следствие - конверсию в целевое действие