План:

МАТЕРИАЛЫ: Ссылка на презентацию:
https://disk.yandex.ru/i/Hl9aBI4oOvpbxg

АНАЛИЗ ПОВЕДЕНИЯ ПОЛЬЗОВАТЕЛЕЙ в мобильном приложении "НЕНУЖНЫЕ ВЕЩИ"

ЦЕЛЬ: выявить закономерности в поведении пользователей и факторы, влияющие на это поведение, для дальнейшей популяризации приложения.

Описание данных

mobile_sources.csv:

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

mobile_dataset.csv:

- `event.time` — время совершения,
- `user.id` — идентификатор пользователя,
- `event.name` — действие пользователя.

Виды действий:

- `advert_open` — открыл карточки объявления,
- `photos_show` — просмотрел фотографий в объявлении,
- `tips_show` — увидел рекомендованные объявления,
- `tips_click` — кликнул по рекомендованному объявлению,
- `contacts_show` и `show_contacts` — посмотрел номер телефона,
- `contacts_call` — позвонил по номеру из объявления,
- `map` — открыл карту объявлений,
- `search_1` — `search_7` — разные действия, связанные с поиском по сайту,
- `favorites_add` — добавил объявление в избранное.

Предобработка данных

Импорт библиотек и загрузка данных:

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats as st
import math as mth
import plotly.express as px
from plotly import graph_objects as go
from tqdm.notebook import tqdm
from plotly.subplots import make_subplots
```


In [2]:
```
mobile_sources = pd.read_csv('https://code.s3.yandex.net/datasets/mobile_sources.csv')
```

In [3]:
```
mobile_dataset = pd.read_csv('https://code.s3.yandex.net/datasets/mobile_dataset.csv')
```

Выведем случайные строки датасета для ознакомления.

In [4]:
```
mobile_sources.sample(5)
```

Out[4]:

	userId	source
2601	56d3680f-2617-4a28-ae4c-6efab4e49cb8	yandex
3469	6315635a-5694-476c-8014-8749514d69e3	yandex
1509	60f24061-aaf4-44cf-8e21-8598dff75757	other
729	dd281955-685d-40ff-9f09-8d159b01818f	yandex
4193	df338ce6-c764-4eb8-b477-ec0f4020861e	yandex

In [5]:
```
mobile_dataset.sample(5)
```

Out[5]:

	event.time	event.name	user.id
73461	2019-11-03 20:22:30.053162	tips_show	e92da447-3a90-4091-847e-5c533a1184f0
48032	2019-10-25 19:14:54.254229	tips_show	6a14b2fe-5aa5-496f-a5d0-71d7870ff91b
7092	2019-10-10 00:23:26.397962	photos_show	2f2d9e7c-baa3-4d3c-806e-d0e4b7f3cb60
47521	2019-10-25 15:43:35.808185	tips_show	46451fb9-4660-454f-b116-ec905d03356e
42908	2019-10-23 22:10:41.082169	tips_show	0a6dc4b7-8cec-4624-81e0-5b559e50f418

Создадим функцию для вывода общей информации по датафрейму: размер, типы данных и количество объектов/пропусков в солбцах,количество полных дубликатов, наименования столбцов отдельно

In [6]:
```
def describe(df: pd.DataFrame):
    display(f'Общая информация: {df.shape}')
    display(df.info())
    print('Дубликаты в массиве:', df.duplicated().sum(), 'в процентах:', round(df.duplicated().mean()*100, 2))
    display(f'Названия столбцов: {df.columns}')
```

In [7]:
```
describe(mobile_sources)
```

```
'Общая информация: (4293, 2)'
<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
Дубликаты в массиве: 0 в процентах: 0.0
"Названия столбцов: Index(['userId', 'source'], dtype='object')"
```

In [8]:
```
describe(mobile_dataset)
```

```
'Общая информация: (74197, 3)'
<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
Дубликаты в массиве: 0 в процентах: 0.0
"Названия столбцов: Index(['event.time', 'event.name', 'user.id'], dtype='object')"
```

Количество строк-дубликатов равно 0 в обоих массивах.

Переименуем столбцы массивов:

In [9]:
```
mobile_sources = mobile_sources.rename(columns={'userId': 'user_id', 'source': 'source'})
mobile_sources.head(2)
```

Out[9]:

	user_id	source
0	020292ab-89bc-4156-9acf-68bc2783f894	other
1	cf7eda61-9349-469f-ac27-e5b6f5ec475c	yandex

In [10]:
```
mobile_dataset = mobile_dataset.rename(columns={'event.time': 'event_time', 'event.name': 'event_name', 'user.id': 'user_id'})
mobile_dataset.head(2)
```

Out[10]:

	event_time	event_name	user_id
0	2019-10-07 00:00:00.431357	advert_open	020292ab-89bc-4156-9acf-68bc2783f894
1	2019-10-07 00:00:01.236320	tips_show	020292ab-89bc-4156-9acf-68bc2783f894

Приведем данные в столбце с датами к соответствующему формату:

In [11]:
```
mobile_dataset['event_time'] = pd.to_datetime(mobile_dataset['event_time'])
mobile_dataset.info()
```

```
<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  datetime64[ns]
 1   event_name  74197 non-null  object        
 2   user_id     74197 non-null  object        
dtypes: datetime64[ns](1), object(2)
memory usage: 1.7+ MB
```

Проверим значения столбца 'event_name':

In [12]:
```
mobile_dataset['event_name'].value_counts()
```

Out[12]:
```
tips_show        40055
photos_show      10012
advert_open       6164
contacts_show     4450
map               3881
search_1          3506
favorites_add     1417
search_5          1049
tips_click         814
search_4           701
contacts_call      541
search_3           522
search_6           460
search_2           324
search_7           222
show_contacts       79
Name: event_name, dtype: int64
```

Видим 2 способа написания одного и того же события contacts_show и show_contacts. Переименуем второй тип в contacts_show, чтобы объединить значения:

In [13]:
```
mobile_dataset.loc[mobile_dataset['event_name'] == 'show_contacts', 'event_name'] = 'contacts_show'
mobile_dataset['event_name'].value_counts()
```

Out[13]:
```
tips_show        40055
photos_show      10012
advert_open       6164
contacts_show     4529
map               3881
search_1          3506
favorites_add     1417
search_5          1049
tips_click         814
search_4           701
contacts_call      541
search_3           522
search_6           460
search_2           324
search_7           222
Name: event_name, dtype: int64
```

Также заменим все виды поиска с индексами от 1 до 7 на search, так как мы не знаем, в чем разница. Но хотелось бы узнать конверсию из поиска в целевое действие в дальнейшем.

In [14]:
```
mobile_dataset['event_name'] = mobile_dataset['event_name'].replace(
    ['search_1', 'search_2','search_3','search_4','search_5','search_6','search_7'], 'search')
mobile_dataset['event_name'].value_counts()
```

Out[14]:
```
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
```

Проверим нет ли событий, совершенных пользователем в одно и то же время:

In [15]:
```
mobile_dataset[mobile_dataset.duplicated(['event_time', 'user_id'])]
```

Out[15]:

	event_time	event_name	user_id

Округлим до секунд и проверим еще раз:

In [16]:
```
mobile_dataset['event_time'] = mobile_dataset['event_time'].dt.round('1s')
mobile_dataset.head()
```

Out[16]:

	event_time	event_name	user_id
0	2019-10-07 00:00:00	advert_open	020292ab-89bc-4156-9acf-68bc2783f894
1	2019-10-07 00:00:01	tips_show	020292ab-89bc-4156-9acf-68bc2783f894
2	2019-10-07 00:00:02	tips_show	cf7eda61-9349-469f-ac27-e5b6f5ec475c
3	2019-10-07 00:00:07	tips_show	020292ab-89bc-4156-9acf-68bc2783f894
4	2019-10-07 00:00:56	advert_open	cf7eda61-9349-469f-ac27-e5b6f5ec475c

In [17]:
```
mobile_dataset[mobile_dataset.duplicated(subset=['event_time', 'event_name', 'user_id'], keep=False)]
```

Out[17]:

	event_time	event_name	user_id
395	2019-10-07 11:00:20	tips_show	fb667205-a708-4693-832d-363a30022cfc
396	2019-10-07 11:00:20	tips_show	fb667205-a708-4693-832d-363a30022cfc
421	2019-10-07 11:10:40	map	ed13f6f0-08f4-4561-852e-456580f7a40d
422	2019-10-07 11:10:40	map	ed13f6f0-08f4-4561-852e-456580f7a40d
423	2019-10-07 11:10:40	map	ed13f6f0-08f4-4561-852e-456580f7a40d
...	...	...	...
73678	2019-11-03 21:10:40	photos_show	06edf71c-b725-47dc-acfe-0c78f079fe8f
73838	2019-11-03 21:45:22	photos_show	1af9ffcd-2c77-4de0-9d35-3ff30604c9bd
73839	2019-11-03 21:45:22	photos_show	1af9ffcd-2c77-4de0-9d35-3ff30604c9bd
74027	2019-11-03 22:41:01	tips_show	16a5371c-152f-48d8-86fe-5636a931316b
74028	2019-11-03 22:41:01	tips_show	16a5371c-152f-48d8-86fe-5636a931316b

2183 rows × 3 columns

Обнаружено 2183 строки с повторами, где один и тот же пользователь совершает в течение секунды одно и то же действие несколько раз. Возможно, это связано с тем, что события фиксируются системой несколько раз в секунду. Пользователь может за секунду пролистать несколько фото или просмотреть несколько рекомендаций. Так как у нас нет id объявлений или событий, мы не будем считать эти повторы дубликатами.

Проверим наименования и количество источников установки:

In [18]:
```
mobile_sources['source'].value_counts()
```

Out[18]:
```
yandex    1934
other     1230
google    1129
Name: source, dtype: int64
```

Проверим количество уникальных пользователей в массивах:

In [19]:
```
mobile_sources['user_id'].nunique()
```

Out[19]:
```
4293
```

In [20]:
```
mobile_dataset['user_id'].nunique()
```

Out[20]:
```
4293
```

Объединим массивы по user_id:

In [21]:
```
mobile = mobile_dataset.merge(mobile_sources, on='user_id', how='left')
mobile.sample(10)
```

Out[21]:

	event_time	event_name	user_id	source
16529	2019-10-14 12:00:11	tips_show	5bffcd16-5312-4137-8dd6-e87c9c40c7e9	other
65931	2019-10-31 17:45:19	tips_show	4060ea41-7ee3-4eb3-9c69-8cb63cd09747	other
68297	2019-11-01 16:06:08	photos_show	cbe34f8a-8f3c-498f-b164-00014a99f8ff	yandex
3249	2019-10-08 13:04:11	tips_show	75996fc0-d7f1-4b35-ab6d-1a1ee3d8cd13	yandex
25043	2019-10-17 11:39:39	tips_show	e999cd93-34e8-4e91-b5f4-444439e72fec	yandex
51803	2019-10-26 22:58:40	photos_show	1af9ffcd-2c77-4de0-9d35-3ff30604c9bd	google
66783	2019-10-31 23:08:54	photos_show	2a816e6f-1357-4df5-b1e5-748807810bcd	other
47646	2019-10-25 16:26:49	advert_open	9ed40b63-d9d1-4154-a992-92816330d7e3	other
48127	2019-10-25 19:56:43	search	1620a0fa-9d2a-4cc5-bd33-e9c209e4a9f0	yandex

	event_time	event_name	user_id	source
54142	2019-10-27 20:05:53	tips_show	475f8fc3-d48c-4bf6-9ebf-f41c6427aa69	yandex

In [22]:
```
mobile.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 74197 entries, 0 to 74196
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   event_time  74197 non-null  datetime64[ns]
 1   event_name  74197 non-null  object
 2   user_id     74197 non-null  object
 3   source      74197 non-null  object
dtypes: datetime64[ns](1), object(3)
memory usage: 2.8+ MB
```

Вывод:

В ходе предобработки данных:

- проверены массивы на пропуски и явные дубликаты;
- откорректированы названия столбцов;
- удалены неявные дубликаты в значениях столбца "event_name";
- объединены в одно 7 событий "search";
- изменен тип данных в столбце времени событий, время округлено до секунд;
- объединены события пользователей, произошедшие в один момент времени;
- массивы объединены в один по идентификаторам пользователей.

Исследовательский анализ данных

Уточнение данных

Подсчитаем количество уникальных пользователей и событий в массиве после обработки:

In [23]:
```
all_us = mobile['user_id'].nunique()
all_us
```

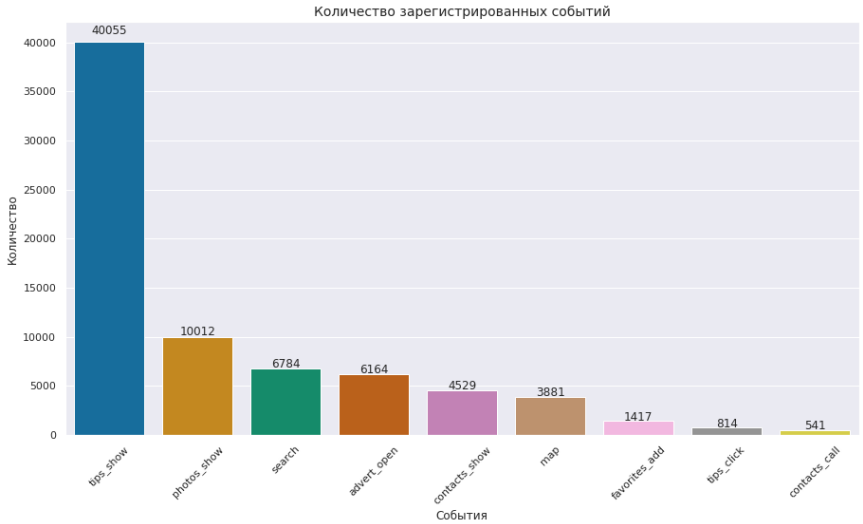
Out[23]: 4293

In [24]:
```
mobile_ev = mobile['event_name'].value_counts().reset_index()
mobile_ev.columns = ['events','total']
mobile_ev
```

Out[24]:

	events	total
0	tips_show	40055
1	photos_show	10012
2	search	6784
3	advert_open	6164
4	contacts_show	4529
5	map	3881
6	favorites_add	1417
7	tips_click	814
8	contacts_call	541

In [25]:
```
sns.set(rc={'figure.figsize':(15, 8)})
sns.set_palette('colorblind')
ax = sns.barplot(x='events', y='total', data=mobile_ev)
ax.set_title('Количество зарегистрированных событий', fontsize=14)
plt.xticks(rotation=45)
plt.xlabel('События')
plt.ylabel('Количество')
for p in ax.patches:
    _x = p.get_x() + p.get_width() / 2
    _y = p.get_y() + p.get_height() + (p.get_height()*0.02)
    value = '{:.0f}'.format(p.get_height())
    ax.text(_x, _y, value, ha="center")
plt.show()
```

Больше всего среди зарегистрированных событий - рекомендаций. Видимо, это связано с тем, что рекомендации выдаются на любом этапе работы с системой, их показ инициируется не самим пользователем. На втором месте по частоте - просмотр фото, который уже зависит от действий самого пользователя. Далее все действия по поиску. И уже потом открытие карточки конкретного объявления и просмотр контактов.

Уточним период, за который предоставлены данные:

In [26]:
```
min = mobile['event_time'].dt.date.min()
display('Первая дата активности пользователей {}'.format(min))
```

```
'Первая дата активности пользователей 2019-10-07'
```

In [27]:
```
max = mobile['event_time'].dt.date.max()
display('Последняя дата активности пользователей {}'.format(max))
```

```
'Последняя дата активности пользователей 2019-11-03'
```

У нас есть данные с 7 октября по 3 ноября 2019 года - примерно за месяц.

Посчитаем количество пользователей, установивших приложение из разных источников:

In [28]:
```
user_source = mobile.pivot_table(index=['source'], values='user_id', aggfunc='nunique').reset_index()
user_source.rename(columns={'user_id': 'user_sum'}, inplace=True)
user_source.sort_values(by='user_sum', ascending = False)
```

Out[28]:

	source	user_sum
2	yandex	1934
1	other	1230
0	google	1129

Яндекс лидирует по количеству установок, Гугл на 3 месте - интересно как отличается конверсия пользователей из этих источников в целевое действие. Есть ли статистически значимая разница в конверсии пользователе, установивших приложение из Гугл и из Яндекс. Проверим на этапе стат. гипотез.
Довольно много установок из других источников, для более глубокого анализа неплохо бы узнать, что это за источники, возможно вреди них есть перспективные, стоит обратить на них больше внимания.

Проверим пересечение установок приложения одним пользователем из разных источников:

In [29]:
```
mobile.groupby('user_id').agg({'source' : 'nunique'}).query('source>1')
```

Out[29]:

	source
user_id	

Пересечений нет. Каждый пользователь установил приложение только из одного источника.

Проанализируем количество событий на пользователя в среднем и по медиане, посмотрим статистику.

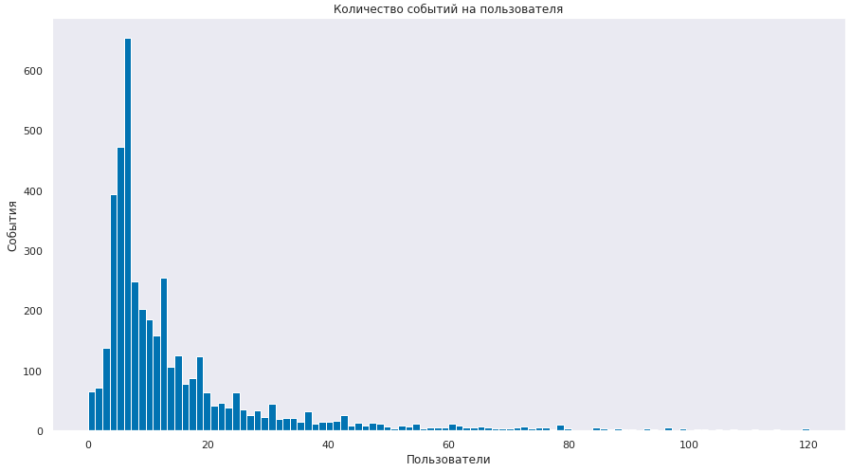
In [30]:
```
visits = mobile.groupby('user_id')['event_name'].count()
```

In [31]:
```
visits.describe()
```

Out[31]:
```
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
```

Видим, что за один месяц пользователь совершает около 9 действий в приложении (по медиане), в целом от 5 до 17, а в среднем без учета выбросов 17 действий. Минимально - 1 действие, максимум 472. Значения в списке отличаются от среднего почти на 29. Активность пользователей довольно сильно отличается. Стоит рассмотреть количество событий в рамках сессий пользователя.
Построим гистограмму количества событий на пользователя по общим данным.

In [32]:
```
plt.figure(figsize=(15,8))
plt.hist(visits, bins=100, range=(0,120))
plt.xlabel('Пользователи')
plt.grid()
plt.ylabel('События')
plt.title('Количество событий на пользователя')
plt.show()
```



Получили распределение Пуассона, которое обрезано на нуле и имеет длинный «хвост» в положительную сторону. При небольшом количестве пользователей есть пользователи с высокой активностью и есть с очень низкой.

Построим гистограмму, чтобы увидеть распределение данных во времени.

In [33]:
```
mobile['event_time'].hist(bins=28*24, figsize=(14, 5))
plt.title('Распределение по датам')
plt.xticks(rotation=45)
plt.show()
```

Активность пользователей чуть меньше в районе 09-10.10.19. Но в целом распределена равномерно по имеющемуся временному интервалу.
Пики и спады приходятся на определенное время активности (день-вечер) и сна (ночь).

Отсортируем массив для выделения сессий:

In [34]:
```
mobile = mobile.sort_values(['user_id', 'event_time'])
mobile.head()
```

Out[34]:

	event_time	event_name	user_id	source
805	2019-10-07 13:39:46	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other
806	2019-10-07 13:40:31	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other
809	2019-10-07 13:41:06	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other
820	2019-10-07 13:43:21	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other
830	2019-10-07 13:45:31	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other

Посчитаем разницу между временем текущего и следующего действия в минутах и секундах. Условно ее можно считать временем совершения текущего действия.

In [35]:
```
mobile['diff'] = mobile.groupby('user_id')['event_time'].diff(1).shift(-1)
mobile.head()
```

Out[35]:

	event_time	event_name	user_id	source	diff
805	2019-10-07 13:39:46	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other	0 days 00:00:45
806	2019-10-07 13:40:31	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other	0 days 00:00:35
809	2019-10-07 13:41:06	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other	0 days 00:02:15
820	2019-10-07 13:43:21	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other	0 days 00:02:10
830	2019-10-07 13:45:31	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other	0 days 00:00:12

In [36]:
```
mobile['diff_s'] = mobile['diff'].astype('timedelta64[s]', errors = 'ignore').astype('int64', errors = 'ignore').fillna(0)
mobile.head()
```

Out[36]:

	event_time	event_name	user_id	source	diff	diff_s
805	2019-10-07 13:39:46	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other	0 days 00:00:45	45.0
806	2019-10-07 13:40:31	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other	0 days 00:00:35	35.0
809	2019-10-07 13:41:06	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other	0 days 00:02:15	135.0
820	2019-10-07 13:43:21	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other	0 days 00:02:10	130.0
830	2019-10-07 13:45:31	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other	0 days 00:00:12	12.0

In [37]:
```
mobile['diff_m'] = round((mobile['diff_s']/60), 2)
mobile.head()
```

Out[37]:

	event_time	event_name	user_id	source	diff	diff_s	diff_m
805	2019-10-07 13:39:46	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other	0 days 00:00:45	45.0	0.75
806	2019-10-07 13:40:31	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other	0 days 00:00:35	35.0	0.58
809	2019-10-07 13:41:06	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other	0 days 00:02:15	135.0	2.25
820	2019-10-07 13:43:21	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other	0 days 00:02:10	130.0	2.17
830	2019-10-07 13:45:31	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other	0 days 00:00:12	12.0	0.20

Посчитаем количество действий, время которых не удалось вычислить, или длящихся меньше секунды.

In [38]:
```
mobile[mobile['diff_s'] == 0]['diff_s'].count()
```

Out[38]:
```
5550
```

Удалим их, чтобы корректно оценить действия по времени.

In [39]:
```
mob = mobile.query('diff_s > 0')
```

Посмотрим на разброс значений по времени действий в минутах.

In [40]:
```
mob['diff_m'].describe()
```

Out[40]:
```
count    68647.000000
mean       237.286377
std       1546.598621
min          0.020000
25%          0.430000
50%          1.220000
75%          3.070000
max      38269.920000
Name: diff_m, dtype: float64
```

Выбросы в данных объясняются временным периодом в месяц, а не в сессию, то есть включают в себя длительные перерывы между посещениями приложения. Почистим данные от выбросов времени для графического анализа продолжительности действий. Огрничим

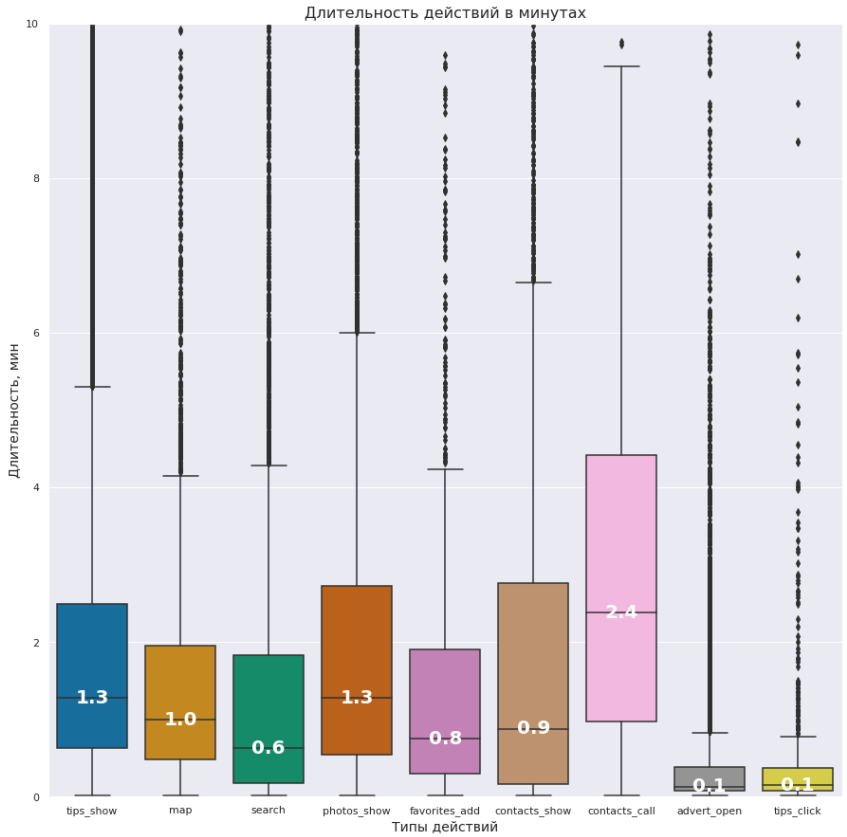
предполагаемую максимальную длительность действия 30 минутами, возьмем значения:

In [41]:
```
mob = mob.query('diff_m < 30')
```

Мы получили временной интервал между действиями пользователя, который условно можно считать продолжительностью каждого действия. Сравним действия по этим интервалам.

In [42]:
```
def add_median_labels(boxplot, fmt='.1f'):
    lines = boxplot.get_lines()
    boxes = [c for c in boxplot.get_children() if type(c).__name__ == 'PathPatch']
    lines_per_box = int(len(lines) / len(boxes))
    for median in lines[4:len(lines):lines_per_box]:
        x, y = (data.mean() for data in median.get_data())
        # choose value depending on horizontal or vertical plot orientation
        value = x if (median.get_xdata()[1] - median.get_xdata()[0]) == 0 else y
        text = boxplot.text(x, y, f'{value:{fmt}}', ha='center', va='center',
                            fontweight='bold', size=20, color='white')
```

In [43]:
```
sns.set(rc={'figure.figsize':(15, 15)})
boxplot = sns.boxplot(x='event_name', y='diff_m', data=mob, palette='colorblind')
boxplot.axes.set_title("Длительность действий в минутах", fontsize=16)
boxplot.set_xlabel("Типы действий", fontsize=14)
boxplot.set_ylabel("Длительность, мин", fontsize=14)
boxplot.set_ylim(0,10)
add_median_labels(boxplot)
plt.show()
```



Медианное время действия колеблется около минуты: от 1 секунды у кликов по рекомендациям и открытий карточек, до 2,4 минут у звонков. Просмотр фото и рекомендаций по медиане 1,3 минуты, поиск довольно быстрый - меньше минуты. Максимальное время звонка самое большое - 9 минут. Поэтому ограничим время на графике 10 минутами. Т.к. большая часть событий ниже этого предела.

In [44]:
```
mob['diff_m'].describe()
```


Out[44]:
```
count    62571.000000
mean         2.133428
std          3.498424
min          0.020000
25%          0.370000
50%          1.050000
75%          2.300000
max         29.970000
Name: diff_m, dtype: float64
```

Установим интервал между сессиями в 30 минут (за это время можно было бы совершить каждое действие и все действия в совокупности, если брать их среднее время) и выделим группы действий пользователя через каждые 30 и более минут в отдельные сессии.

In [45]:
```
#identify difference 5Min for each group with cumulative sum
g = (mobile.groupby('user_id')['event_time'].diff() > pd.Timedelta('30Min')).cumsum()
#create counter of groups
mobile['session_id'] = mobile.groupby(['user_id', g], sort=False).ngroup() + 1
mobile.head()
```

Out[45]:

	event_time	event_name	user_id	source	diff	diff_s	diff_m	session_id
805	2019-10-07 13:39:46	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other	0 days 00:00:45	45.0	0.75	1
806	2019-10-07 13:40:31	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other	0 days 00:00:35	35.0	0.58	1
809	2019-10-07 13:41:06	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other	0 days 00:02:15	135.0	2.25	1
820	2019-10-07 13:43:21	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other	0 days 00:02:10	130.0	2.17	1
830	2019-10-07 13:45:31	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other	0 days 00:00:12	12.0	0.20	1

In [46]:
```
u = mobile['user_id'].nunique()
s = mobile['session_id'].nunique()
su = round(s/u)
print(f'Количество пользователей = {u}. Количество сессий = {s}. Среднее количество сессий на пользователя за 28 дней = {su}.')
```

```
Количество пользователей = 4293. Количество сессий = 10368. Среднее количество сессий на пользователя за 28 дней = 2.
```

In [47]:
```
mobile.groupby('user_id')['session_id'].nunique().describe()
```

Out[47]:
```
count    4293.000000
mean        2.415094
std         3.536466
min         1.000000
25%         1.000000
50%         1.000000
75%         3.000000
max        99.000000
Name: session_id, dtype: float64
```

Количество сессий на пользователя по медиане = 1. Максимум, правда, очень велик. 98 сессий совершил какой-то очень активный пользователь.

In [48]:
```
mobile.groupby('session_id')['event_name'].count().describe()
```

Out[48]:
```
count    10368.000000
mean         7.156346
std          9.581106
min          1.000000
25%          2.000000
50%          4.000000
75%          9.000000
max        149.000000
Name: event_name, dtype: float64
```

С учетом повторяемости действий в рамках сессии пользователь совершает по медиане 4 действия, в среднем без учета выбросов - 7. В большинстве случаев от 2 до 8.

Добавим дни в массив и посмотрим на распределение сессий пользователей по дням.

In [49]:
```
mobile['day'] = mobile['event_time'].dt.date
```

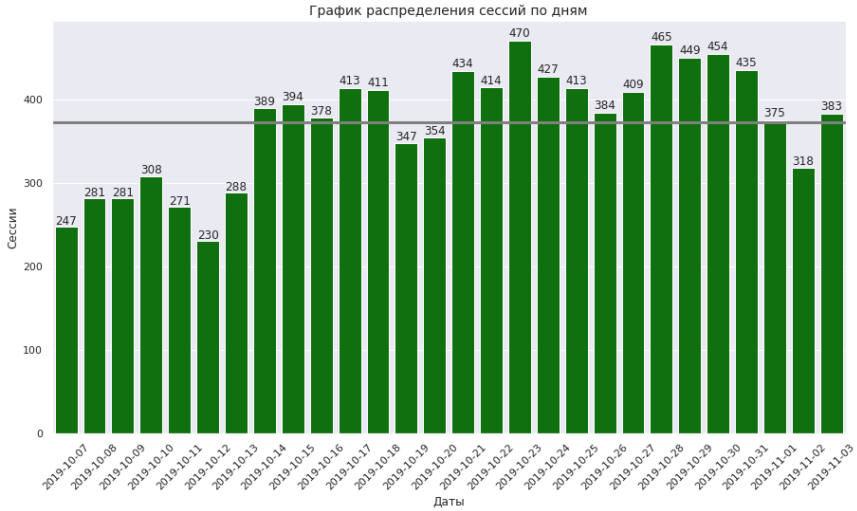
In [50]:
```
sessions = mobile.groupby('day')['session_id'].nunique().reset_index()
sessions.columns = ['day','sessions']
sessions.describe()
```

Out[50]:

	sessions
count	28.000000
mean	372.214286
std	68.963413
min	230.000000
25%	315.500000
50%	386.500000
75%	417.250000
max	470.000000

In [51]:
```
average = sessions['sessions'].mean()
```

In [52]:
```
sns.set(rc={'figure.figsize':(15, 8)})
ax = sns.barplot(x='day', y='sessions', data=sessions, color='green')
ax.set_title('График распределения сессий по дням', fontsize=14)
plt.xticks(rotation=45)
plt.xlabel('Даты')
plt.ylabel('Сессии')
plt.axhline(y=average, color = 'grey', linewidth=3)
for p in ax.patches:
    _x = p.get_x() + p.get_width() / 2
    _y = p.get_y() + p.get_height() + (p.get_height()*0.01)
    value = '{:.0f}'.format(p.get_height())
    ax.text(_x, _y, value, ha="center")
plt.show()
```



Количество сессий наибольшее примерно с середины до конца октября. Есть тенденция к спаду активности в начале месяца. Но чтобы убедиться в этом нужны данные за несколько месяцев и более. В среднем в день 372 сессии.

Подготовим массив для выделения сценариев. В рамках сценария нас не интересуют повторяющиеся в одной сессии действия. Удалим их.

In [53]:
```
m = mobile.drop_duplicates(subset=['event_name', 'user_id', 'session_id'], keep='first').reset_index(drop=True)
m.head()
```

Out[53]:

	event_time	event_name	user_id	source	diff	diff_s	diff_m	session_id	day
0	2019-10-07 13:39:46	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other	0 days 00:00:45	45.0	0.75	1	2019-10-07
1	2019-10-09 18:33:56	map	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other	0 days 00:01:32	92.0	1.53	2	2019-10-09
2	2019-10-09 18:40:29	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other	0 days 00:01:54	114.0	1.90	2	2019-10-09
3	2019-10-21 19:52:31	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other	0 days 00:00:46	46.0	0.77	3	2019-10-21
4	2019-10-21 19:53:39	map	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	other	0 days 00:01:06	66.0	1.10	3	2019-10-21

Вывод

- у нас есть данные с 7 октября по 3 ноября 2019 года - примерно за месяц;
- больше всего среди зарегистрированных событий - рекомендаций. Видимо, это связано с тем, что рекомендации выдаются на любом этапе работы с системой, их показ инициируется не самим пользователем. На втором месте по частоте - просмотр фото, который уже зависит от действий самого пользователя. Далее все действия по поиску. И уже потом открытие карточки конкретного объявления и просмотр контактов;
- 3 источника установок: яндекс, другие и гугл - в порядке количества установок из ник. Пользователи по источникам не пересекаются;
- за один месяц пользователь совершает около 9 действий в приложении (по медиане), в целом от 5 до 17, а в среднем без учета выбросов 17 действий. Минимально - 1 действие, максимум 472. Значения в списке отличаются от среднего почти на 29. Активность пользователей довольно сильно отличается;
- с учетом повторяемости действий в рамках сессии пользователь совершает по медиане 4 действия, в среднем без учета выбросов - 7. В большинстве случаев от 2 до 8;
- медианное время действия колеблется около минуты: от 1 секунды у кликов по рекомендациям и открытий карточек, до 2,4 минут у звонков. Просмотр фото и рекомендаций по медиане 1,3 минуты, поиск довольно быстрый - меньше минуты. Максимальное время звонка самое большое - 9 минут;

- количество пользователей = 4293. Количество сессий = 10368. Среднее количество сессий на пользователя за 28 дней = 2, по медиане 1;
- количество сессий наибольшее примерно с середины до конца октября. Есть тенденция к спаду активности в начале месяца. Но чтобы убедиться в этом нужны данные за несколько месяцев и более. В среднем в день 352 сессии.

Выделение сценариев использования приложения в рамках сессий

Построим диаграмму Сэнкей для выделения сценариев действий. Определим пары исходное - целевое событие.

In [54]:
```
def add_features(df):

    """Функция генерации новых столбцов для исходной таблицы

    Args:
        df (pd.DataFrame): исходная таблица.
    Returns:
        pd.DataFrame: таблица с новыми признаками.
    """

    # сортируем по id и времени
    sorted_df = df.sort_values(by=['session_id', 'event_time']).copy()
    # добавляем шаги событий
    sorted_df['step'] = sorted_df.groupby('session_id').cumcount() + 1

    # добавляем узлы-источники и целевые узлы
    # узлы-источники - это сами события
    sorted_df['source_event'] = sorted_df['event_name']
    # добавляем целевые узлы
    sorted_df['target'] = sorted_df.groupby('session_id')['source_event'].shift(-1)

    # возврат таблицы без имени событий
    return sorted_df.drop(['event_name', 'source', 'diff', 'diff_m'], axis=1)

# преобразуем таблицу
table = add_features(m)
table.head()
```

Out[54]:

	event_time	user_id	diff_s	session_id	day	step	source_event	target
0	2019-10-07 13:39:46	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	45.0	1	2019-10-07	1	tips_show	NaN
1	2019-10-09 18:33:56	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	92.0	2	2019-10-09	1	map	tips_show
2	2019-10-09 18:40:29	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	114.0	2	2019-10-09	2	tips_show	NaN
3	2019-10-21 19:52:31	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	46.0	3	2019-10-21	1	tips_show	map
4	2019-10-21 19:53:39	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	66.0	3	2019-10-21	2	map	NaN

Следующее, что нужно сделать - это выбрать количество шагов на нашей будущей диаграмме. На основании подсчета шагов видим, что максимальное их количество: 6. Но большинство пользователей ограничивается 3-мя. Достаточно будет оставить 4 шага в диаграмме, чтобы не удлинять ее излишне,к тому же у нас нет цели выделить все паттерны, а только основные и самые распространенные.

In [55]:
```
table['step'].value_counts()
```

Out[55]:
```
1    10368
2     5076
3     1818
4      503
5       83
6        6
Name: step, dtype: int64
```

In [56]:
```
df_comp = table.rename(columns={'source_event':'source'})
df_comp = df_comp[df_comp['step'] <= 4].copy().reset_index(drop=True)
```

Создадим словарь, в котором ключи - это шаги, а значения - словари со списком названий source_event и соответствующих им индексов.

In [57]:
```
def get_source_index(df):

    """Функция генерации индексов source

    Args:
        df (pd.DataFrame): исходная таблица с признаками step, source, target.
    Returns:
        dict: словарь с индексами, именами и соответсвиями индексов именам source.
    """

    res_dict = {}

    count = 0
    # получаем индексы источников
    for no, step in enumerate(df['step'].unique().tolist()):
        # получаем уникальные наименования для шага
        res_dict[no+1] = {}
        res_dict[no+1]['sources'] = df[df['step'] == step]['source'].unique().tolist()
        res_dict[no+1]['sources_index'] = []
        for i in range(len(res_dict[no+1]['sources'])):
            res_dict[no+1]['sources_index'].append(count)
            count += 1

    # соединим списки
```

```
    for key in res_dict:
        res_dict[key]['sources_dict'] = {}
        for name, no in zip(res_dict[key]['sources'], res_dict[key]['sources_index']):
            res_dict[key]['sources_dict'][name] = no
    return res_dict


# создаем словарь
source_indexes = get_source_index(df_comp)
```

Для более наглядного представления можно разукрасить каждый source-target в разные цвета. Цвета выберем в цветовой модели RGBA. Цвет будем генерировать для каждого уникального источника. Для этого создадим еще один словарь, в котором будут храниться соответствия source:color.

In [58]:
```
def generate_random_color():

    """Случайная генерация цветов rgba

    Args:

    Returns:
        str: Строка со сгенерированными параметрами цвета
    """

    # сгенерим значение для каждого канала
    r, g, b = np.random.randint(255, size=3)
    return f'rgba({r}, {g}, {b}, 1)'
```

In [59]:
```
def colors_for_sources(mode):

    """Генерация цветов rgba

    Args:
        mode (str): сгенерировать случайные цвета, если 'random', а если 'custom' -
                    использовать заранее подготовленные
    Returns:
        dict: словарь с цветами, соответствующими каждому индексу
    """
    # словарь, в который сложим цвета в соответствии с индексом
    colors_dict = {}

    if mode == 'random':
        # генерим случайные цвета
        for label in df_comp['source'].unique():
            r, g, b = np.random.randint(255, size=3)
            colors_dict[label] = f'rgba({r}, {g}, {b}, 1)'

    elif mode == 'custom':
        # присваиваем ранее подготовленные цвета
        colors = requests.get('https://raw.githubusercontent.com/rusantsovsv/senkey_tutorial/main/json/colors_senkey.json').json()
        for no, label in enumerate(df_comp['source'].unique()):
            colors_dict[label] = colors['custom_colors'][no]

    return colors_dict

colors_dict = colors_for_sources(mode='random')
```

Расчет количества уникальных пользователей в процентах

In [60]:
```
def percent_users(sources, targets, values):

    """
    Расчет уникальных id в процентах (для вывода в hover text каждого узла)

    Args:
        sources (list): список с индексами source.
        targets (list): список с индексами target.
        values (list): список с "объемами" потоков.

    Returns:
        list: список с "объемами" потоков в процентах
    """

    # объединим источники и метки и найдем пары
    zip_lists = list(zip(sources, targets, values))

    new_list = []

    # подготовим список словарь с общим объемом трафика в узлах
    unique_dict = {}

    # проходим по каждому узлу
    for source, target, value in zip_lists:
        if source not in unique_dict:
            # находим все источники и считаем общий трафик
            unique_dict[source] = 0
            for sr, tg, vl in zip_lists:
                if sr == source:
                    unique_dict[source] += vl

    # считаем проценты
```



```
    for source, target, value in zip_lists:
        new_list.append(round(100 * value / unique_dict[source], 1))

    return new_list
```

Создание словаря с данными для отрисовки диаграммы

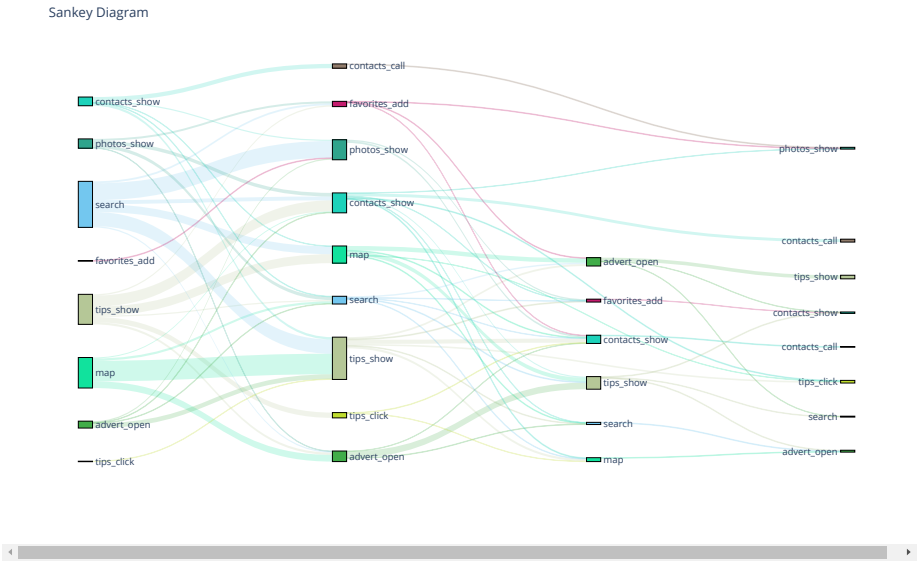
In [61]:
```
def lists_for_plot(source_indexes=source_indexes, colors=colors_dict, frac=10):

    """
    Создаем необходимые для отрисовки диаграммы переменные списков и возвращаем
    их в виде словаря

    Args:
        source_indexes (dict): словарь с именами и индексами source.
        colors (dict): словарь с цветами source.
        frac (int): ограничение на минимальный "объем" между узлами.

    Returns:
        dict: словарь со списками, необходимыми для диаграммы.
    """

    sources = []
    targets = []
    values = []
    labels = []
    link_color = []
    link_text = []

    # проходим по каждому шагу
    for step in tqdm(sorted(df_comp['step'].unique()), desc='Шаг'):
        if step + 1 not in source_indexes:
            continue

        # получаем индекс источника
        temp_dict_source = source_indexes[step]['sources_dict']

        # получаем индексы цели
        temp_dict_target = source_indexes[step+1]['sources_dict']

        # проходим по каждой возможной паре, считаем количество таких пар
        for source, index_source in tqdm(temp_dict_source.items()):
            for target, index_target in temp_dict_target.items():
                # делаем срез данных и считаем количество id
                temp_df = df_comp[(df_comp['step'] == step)&(df_comp['source'] == source)&(df_comp['target'] == target)]
                value = len(temp_df)
                # проверяем минимальный объем потока и добавляем нужные данные
                if value > frac:
                    sources.append(index_source)
                    targets.append(index_target)
                    values.append(value)
                    # делаем поток прозрачным для лучшего отображения
                    link_color.append(colors[source].replace(', 1)', ', 0.2)'))

    labels = []
    colors_labels = []
    for key in source_indexes:
        for name in source_indexes[key]['sources']:
            labels.append(name)
            colors_labels.append(colors[name])

    # посчитаем проценты всех потоков
    perc_values = percent_users(sources, targets, values)

    # добавим значения процентов для howertext
    link_text = []
    for perc in perc_values:
        link_text.append(f"{perc}%")

    # возвратим словарь с вложенными списками
    return {'sources': sources,
            'targets': targets,
            'values': values,
            'labels': labels,
            'colors_labels': colors_labels,
            'link_color': link_color,
            'link_text': link_text}


# создаем словарь
data_for_plot = lists_for_plot()
```

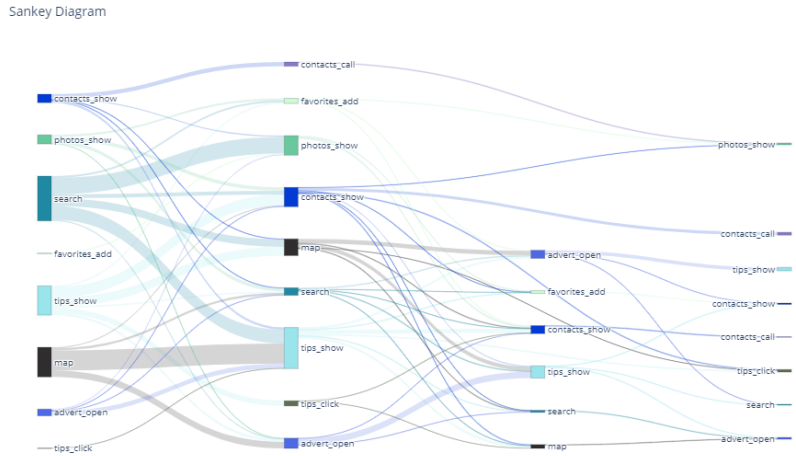
Создание объекта диаграммы

In [62]:
```
def plot_senkey_diagram(data_dict=data_for_plot):

    """
    Функция для генерации объекта диаграммы Сенкей

    Args:
        data_dict (dict): словарь со списками данных для построения.
```

```
    Returns:
        plotly.graph_objs._figure.Figure: объект изображения.
    """

    fig = go.Figure(data=[go.Sankey(
        domain = dict(
          x =  [0,1],
          y =  [0,1]
        ),
        orientation = "h",
        valueformat = ".0f",
        node = dict(
          pad = 50,
          thickness = 15,
          line = dict(color = "black", width = 0.1),
          label = data_dict['labels'],
          color = data_dict['colors_labels']
        ),
        link = dict(
          source = data_dict['sources'],
          target = data_dict['targets'],
          value = data_dict['values'],
          label = data_dict['link_text'],
          color = data_dict['link_color']
      ))])
    fig.update_layout(title_text="Sankey Diagram", font_size=10, width=980, height=600)

    # возвращаем объект диаграммы
    return fig


# сохраняем диаграмму в переменную
senkey_diagram = plot_senkey_diagram()
```

In [63]:
```
senkey_diagram.show()
```





Видим, что большая часть пользователей начинает с поиска, в целевой просмотр контактов они попадают из поиска напрямую, через фотопросмотр и просмотр рекомендаций. То есть 1: search-contacts_show; 2: search-photos_show-contacts_show; 3: tips_show-map-contacts_show. Также популярны сценарии tips_show-contacts_show и photos_show-contacts_show. Если рассматривать действия в 4 шага наиболее перспективным по диаграмме кажется сценарий map-advert_open-tips_show-contacts_show. Но большая часть пользователей явно останавливается на 2-3 шагах. Их и рассмотрим далее.

Выявление самых распространенных событий, построение воронок по основным сценариям и определение конверсии перехода в целевое действие по сценарию

В распределении количества событий на пользователя лидируют: просмотр рекомендаций, поиск и просмотр на карте. Просмотр фото и контактов - замыкают топ-5. Открытие карточки события на 6 месте.

In [64]:
```
user_event= mobile.pivot_table(index=['event_name'], values='user_id', aggfunc='nunique').sort_values(by='user_id', ascending = False
user_event.rename(columns={'user_id': 'user_sum'}, inplace=True)
user_event
```

Out[64]:

	event_name	user_sum
0	tips_show	2801
1	search	1666
2	map	1456
3	photos_show	1095
4	contacts_show	981
5	advert_open	751
6	favorites_add	351
7	tips_click	322
8	contacts_call	213

Самые распростренные действия пользователей:

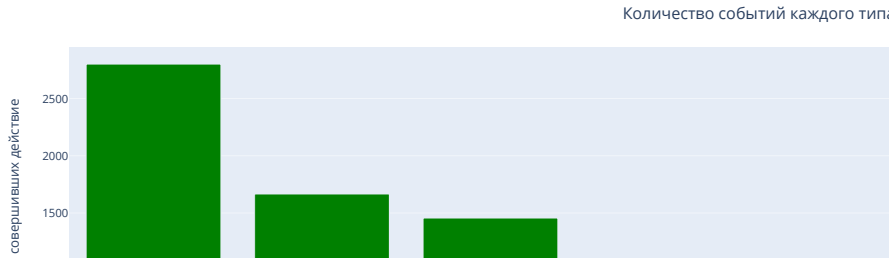
In [65]:
```
sum_event = mobile.groupby('event_name')['event_time'].count()
sum_event = sum_event.reset_index().rename(columns={'event_time': 'sum_event'}).sort_values(by='sum_event', ascending = False)
sum_event
```

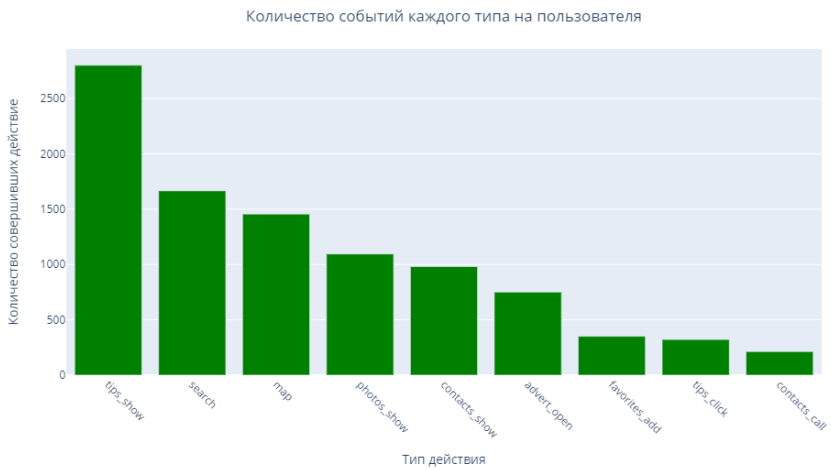
Out[65]:

	event_name	sum_event
8	tips_show	40055
5	photos_show	10012
6	search	6784
0	advert_open	6164
2	contacts_show	4529
4	map	3881
3	favorites_add	1417

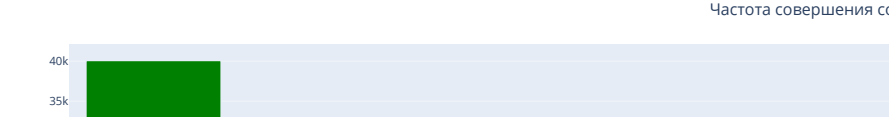
	event_name	sum_event
7	tips_click	814
1	contacts_call	541

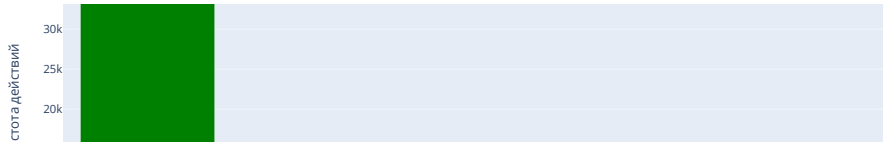
In [66]:
```
fig = px.bar(user_event, x='event_name', y='user_sum', color_discrete_sequence=["green"])
fig.update_xaxes(tickangle=45)
fig.update_layout(title='Количество событий каждого типа на пользователя', title_x = 0.5,
                  xaxis_title='Тип действия',
                  yaxis_title='Количество совершивших действие')
fig.show()
```

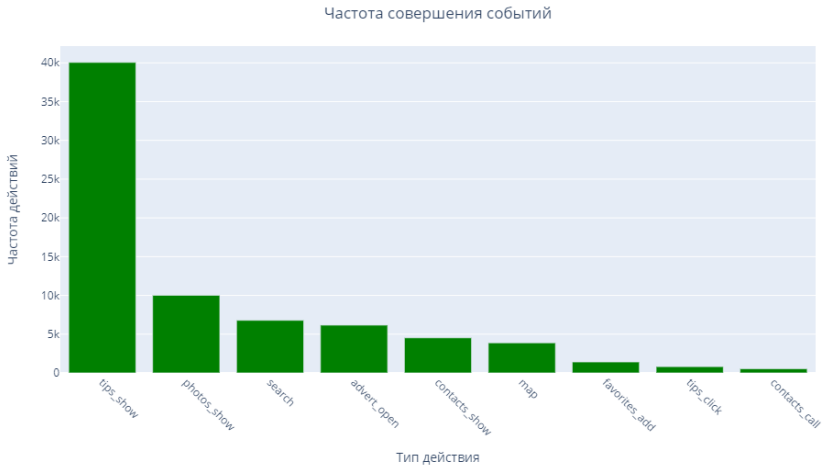




In [67]:
```
fig = px.bar(sum_event, x='event_name', y='sum_event', color_discrete_sequence=["green"])
fig.update_xaxes(tickangle=45)
fig.update_layout(title='Частота совершения событий', title_x = 0.5,
                  xaxis_title='Тип действия',
                  yaxis_title='Частота действий')
fig.show()
```







Количество уникальных пользователей и событий по типам событий. Их количество на пользователя:

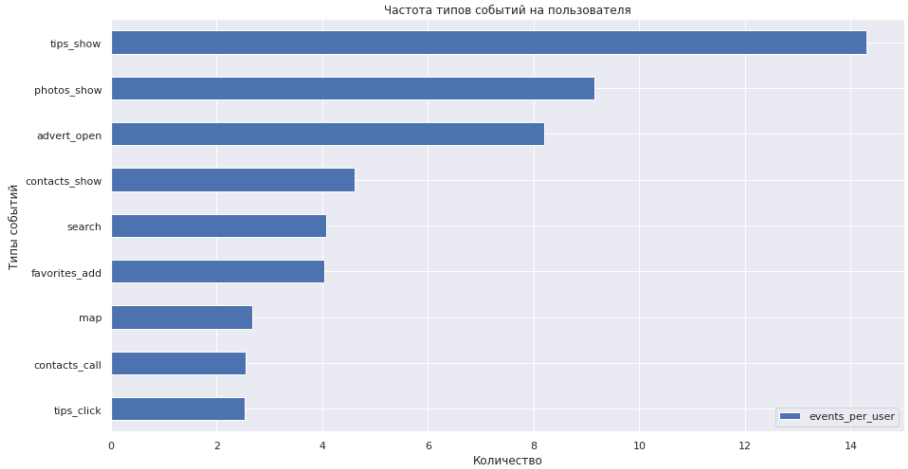
In [68]:
```
t = mobile.groupby('event_name').agg(
    events_cnt = ('user_id', 'count'),
    users = ('user_id', 'nunique'))
t['events_per_user'] = t['events_cnt'] / t['users']
```

In [69]:
```
t = t.sort_values('events_per_user')
t
```

Out[69]:

event_name	events_cnt	users	events_per_user
tips_click	814	322	2.527950
contacts_call	541	213	2.539906
map	3881	1456	2.665522
favorites_add	1417	351	4.037037
search	6784	1666	4.072029
contacts_show	4529	981	4.616718
advert_open	6164	751	8.207723
photos_show	10012	1095	9.143379
tips_show	40055	2801	14.300250

In [70]:
```
t.plot(kind='barh', y='events_per_user')
plt.title('Частота типов событий на пользователя')
plt.xlabel('Количество')
plt.ylabel('Типы событий')
plt.show()
```



Выведем таблицу переходов из события в событие в рамках сессии, чтобы учесть их последовательность:

In [71]:
```
def add_target(df):

    """Функция генерации новых столбцов для исходной таблицы

    Args:
        df (pd.DataFrame): исходная таблица.
    Returns:
        pd.DataFrame: таблица с новыми признаками.
    """

    # сортируем по id и времени
    sorted_df = df.sort_values(by=['session_id', 'event_time']).copy()
    # добавляем шаги событий
    sorted_df['step'] = sorted_df.groupby('session_id').cumcount() + 1

    # добавляем узлы-источники и целевые узлы
    # узлы-источники - это сами события
    sorted_df['source_event'] = sorted_df['event_name']
    # добавляем целевые узлы
    sorted_df['target'] = sorted_df.groupby('session_id')['source_event'].shift(-1)
    # возврат таблицы без имени событий
    return sorted_df.drop(['source', 'diff', 'diff_m'], axis=1)

# преобразуем таблицу
target = add_target(mobile)
target.head()
```

Out[71]:

	event_time	event_name	user_id	diff_s	session_id	day	step	source_event	target
805	2019-10-07 13:39:46	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	45.0	1	2019-10-07	1	tips_show	tips_show
806	2019-10-07 13:40:31	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	35.0	1	2019-10-07	2	tips_show	tips_show
809	2019-10-07 13:41:06	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	135.0	1	2019-10-07	3	tips_show	tips_show
820	2019-10-07 13:43:21	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	130.0	1	2019-10-07	4	tips_show	tips_show
830	2019-10-07 13:45:31	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	12.0	1	2019-10-07	5	tips_show	tips_show

Таблица переходов в просмотр контактов:

In [72]:
```
target1 = target.query('target == "contacts_show"')
target1.head()
```

Out[72]:

	event_time	event_name	user_id	diff_s	session_id	day	step	source_event	target
33528	2019-10-20 19:11:47	search	00157779-810c-4498-9e05-a1e9e3cedf93	332.0	6	2019-10-20	6	search	contacts_show
33540	2019-10-20 19:20:42	photos_show	00157779-810c-4498-9e05-a1e9e3cedf93	150.0	6	2019-10-20	10	photos_show	contacts_show
33545	2019-10-20 19:23:14	contacts_call	00157779-810c-4498-9e05-a1e9e3cedf93	438.0	6	2019-10-20	12	contacts_call	contacts_show
33615	2019-10-20 20:04:17	photos_show	00157779-810c-4498-9e05-a1e9e3cedf93	36.0	6	2019-10-20	16	photos_show	contacts_show
60711	2019-10-29 21:26:06	photos_show	00157779-810c-4498-9e05-a1e9e3cedf93	34.0	8	2019-10-29	7	photos_show	contacts_show

Посчитаем, какое количество уникальных пользователей целевого события совершили до этого разные типы действий:

In [73]:
```
user_target = target1.pivot_table(index=['event_name'], values='user_id', aggfunc='nunique').sort_values(by='user_id', ascending = Fa
user_target.rename(columns={'user_id': 'user_sum'}, inplace=True)
```


In [73]:
```
user_target
```

Out[73]:

	event_name	user_sum
0	tips_show	461
1	contacts_show	330
2	photos_show	211
3	search	129
4	contacts_call	81
5	advert_open	41
6	map	37
7	favorites_add	33
8	tips_click	7

Объединим массивы с пользователями, совершившими каждое действие, и пользователями, перешедшими из этих действий в целевое.

In [74]:
```
conv = user_event.merge(user_target, on='event_name')
conv['convers'] = round(conv['user_sum_y']*100/conv['user_sum_x'])
conv = conv.sort_values(by ='convers', ascending = False).reset_index(drop=True)
conv
```

Out[74]:

	event_name	user_sum_x	user_sum_y	convers
0	contacts_call	213	81	38.0
1	contacts_show	981	330	34.0
2	photos_show	1095	211	19.0
3	tips_show	2801	461	16.0
4	favorites_add	351	33	9.0
5	search	1666	129	8.0
6	advert_open	751	41	5.0
7	map	1456	37	3.0
8	tips_click	322	7	2.0

Видим, что чаще всего пользователи переходят из просмотра контактов и звонков обратно в контакты. А вот по остальным действиям конверсия еще ниже: 19 % из просмотра фото, 16 % из рекомендаций (которым нельзя особо доверять, это не клики, а просмотры), 9 и 8 соответственно из избранного и поиска.

Рассчитаем конверсию и построим воронки для выделенных по диаграмме Сэнкей сценариям.

Конверсия перехода в целевое действие по сценарию tips_show - contacts_show

In [75]:
```
tips_show = target1.query('event_name == "tips_show"').pivot_table(index=['event_name'], values='user_id', aggfunc='nunique').sort_va
tips_show
```

Out[75]:

	event_name	user_id
0	tips_show	461

Построим датафрейм для отображения воронки и конверсии по шагам:

In [76]:
```
df1 = pd.DataFrame([['tips_show', 2801], ['contacts_show', 461]], columns=['event_name', 'user_sum'])
df1
```

Out[76]:

	event_name	user_sum
0	tips_show	2801
1	contacts_show	461

Конверсия перехода в целевое действие по сценарию photos_show - contacts_show

In [77]:
```
photos_show = target1.query('event_name == "photos_show"').pivot_table(index=['event_name'], values='user_id', aggfunc='nunique').sort
photos_show
```

Out[77]:

	event_name	user_id
0	photos_show	211

In [78]:
```
df2 = pd.DataFrame([['photos_show', 1095], ['contacts_show', 211]], columns=['event_name', 'user_sum'])
df2
```

Out[78]:

	event_name	user_sum
0	photos_show	1095

	event_name	user_sum
1	contacts_show	211

Конверсия перехода в целевое действие по сценарию tips_show-map-contacts_show

Создадим датафрейм с целевым событием просмотр карты. Далее для данного сценария выполним те же действия: посчитаем количество пользователей каждого шага, просмотревших контакты.

In [79]:
```
target2 = target.query('target == "map"')
```

In [80]:
```
tips = target2.query('event_name == "tips_show"')
```

In [81]:
```
user_event_fun_m = tips.pivot_table(index=['event_name'], values='user_id', aggfunc='nunique').sort_values(by='user_id', ascending = F
user_event_fun_m
```

Out[81]:

	event_name	user_id
0	tips_show	593

In [82]:
```
user_event_fun_mc = target1.query('event_name == "map"').pivot_table(index=['event_name'], values='user_id', aggfunc='nunique').sort_v
user_event_fun_mc
```

Out[82]:

	event_name	user_id
0	map	37

In [83]:
```
df5 = pd.DataFrame([['tips_show', 2801], ['map', 593], ['contacts_show', 37]], columns=['event_name', 'user_sum'])
df5
```

Out[83]:

	event_name	user_sum
0	tips_show	2801
1	map	593
2	contacts_show	37

Конверсия перехода в целевое действие по сценарию search-contacts_show

Пользователи, совершившие поиск и просмотр контактов:

In [84]:
```
user_event_s = target1.query('event_name == "search"').pivot_table(index=['event_name'], values='user_id', aggfunc='nunique').sort_va
user_event_s
```

Out[84]:

	event_name	user_id
0	search	129

In [85]:
```
df3 = pd.DataFrame([['search', 1666], ['contacts_show', 129]], columns=['event_name', 'user_sum'])
df3
```

Out[85]:

	event_name	user_sum
0	search	1666
1	contacts_show	129

Конверсия перехода в целевое действие по сценарию search-photos_show-contacts_show

Создадим датафрейм с целевым событием просмотр фото. Далее для данного сценария выполним те же действия: посчитаем количество пользователей каждого шага, просмотревших контакты.

In [86]:
```
target3 = target.query('target == "photos_show"')
```

In [87]:
```
user_event_fun_p = target3.query('event_name == "search"').pivot_table(index=['event_name'], values='user_id', aggfunc='nunique').sor
user_event_fun_p
```

Out[87]:

	event_name	user_id
0	search	514

In [88]:
```
user_event_fun_pc = target1.query('event_name == "photos_show"').pivot_table(index=['event_name'], values='user_id', aggfunc='nunique
user_event_fun_pc
```

Out[88]:

	event_name	user_id
0	photos_show	211

In [89]:
```
df4 = pd.DataFrame([['search', 1666], ['photos_show', 514], ['contacts_show', 211]], columns=['event_name', 'user_sum'])
df4
```

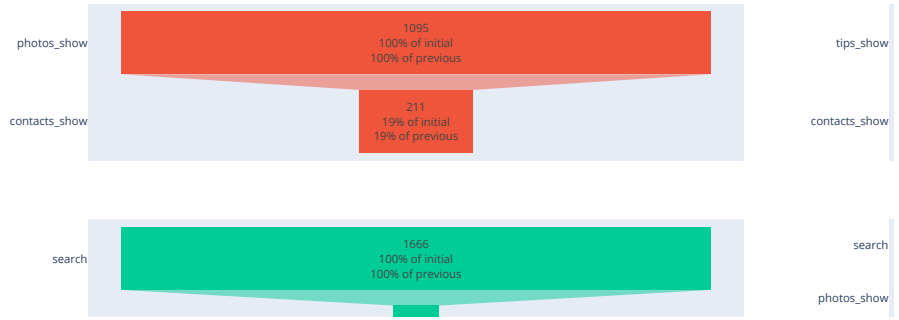
Out[89]:

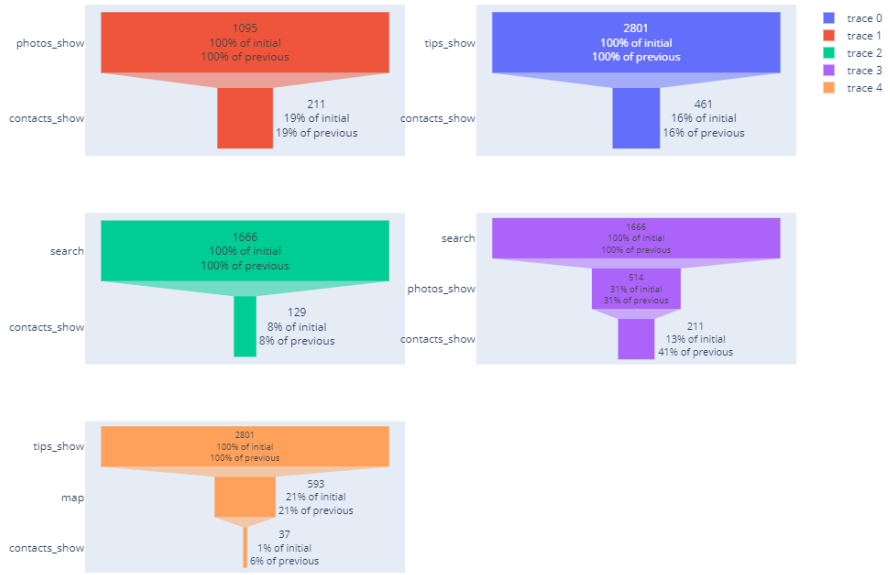
	event_name	user_sum
0	search	1666
1	photos_show	514
2	contacts_show	211

Воронки перехода в целевое действие contacts_show по выделенным сценариям

Построим воронки по рассчитанным данным:

In [90]:
```
fig = make_subplots(rows=3, cols=2)

fig.add_trace(go.Funnel(x = df1['user_sum'], y = df1['event_name'], textinfo = "value+percent initial+percent previous"),row=1, col=2
fig.add_trace(go.Funnel(x = df2['user_sum'], y = df2['event_name'], textinfo = "value+percent initial+percent previous"),row=1, col=1
fig.add_trace(go.Funnel(x = df3['user_sum'], y = df3['event_name'], textinfo = "value+percent initial+percent previous"),row=2, col=1
fig.add_trace(go.Funnel(x = df4['user_sum'], y = df4['event_name'], textinfo = "value+percent initial+percent previous"),row=2, col=2
fig.add_trace(go.Funnel(x = df5['user_sum'], y = df5['event_name'], textinfo = "value+percent initial+percent previous"),row=3, col=1
fig.update_layout(height=800)
fig.show()
```





In [91]:
```
conv
```

Out[91]:

	event_name	user_sum_x	user_sum_y	convers
0	contacts_call	213	81	38.0
1	contacts_show	981	330	34.0
2	photos_show	1095	211	19.0
3	tips_show	2801	461	16.0
4	favorites_add	351	33	9.0
5	search	1666	129	8.0
6	advert_open	751	41	5.0
7	map	1456	37	3.0
8	tips_click	322	7	2.0

По расчетам и воронкам можно сделать вывод, что конверсия в просмотр контактов оставляет желать лучшего:

- по сценарию tips_show - contacts_show конверсия перехода в целевое действие составляет 16 %;
- по сценарию tips_show-map-contacts_show: 21 % пользователей из просмотра рекомендаций переходят к просмотру карты. С шага просмотра карты до целевого действия доходит 6 % пользователей. А от просмотра рекомендаций к просмотру карты, а затем к просмотру контактов переходит только 1 % пользователей;
- по сценарию search-contacts_show конверсия пользователей из поиска в просмотр контактов 8 %;
- по сценарию search-photos_show-contacts_show: 31 % пользователей переходят от поиска к просмотру фото. 41 % смотревших фото переходят к контактам. А по цепочке: поиск-фото-контакты переходит 13 % пользователей;
- по сценарию photos_show-contacts_show: больше всего пользователей переходят к просмотру контактов из просмотра фото - 19 %.

Среднее время перехода пользователей к целевому действию в рамках сессии

Среднее время перехода пользователя от search к contacts_show в рамках сессии

Найдем среднее время конверсии из поиска в просмотр контактов в рамках сессии. Для этого отфильтруем значения в массиве m, где остались события только по первому вхождению, по нужным нам типам событий и вычислим разницу во времени:

In [92]:
```
ms=m.query('event_name == "search"').rename(columns = {'event_time':'time1'})
ms = ms.drop(['user_id', 'source', 'diff', 'diff_m', 'diff_s'], axis=1)
ms.shape
```


Out[92]: (2974, 4)

In [93]:
```
mc=m.query('event_name == "contacts_show"').rename(columns = {'event_time':'time2'})
mc = mc.drop(['user_id', 'source', 'diff', 'diff_m', 'diff_s'], axis=1)
mc.shape
```

Out[93]: (1703, 4)

In [94]:
```
msc= ms.merge(mc, on='session_id')
msc.sample()
```

Out[94]:

	time1	event_name_x	session_id	day_x	time2	event_name_y	day_y
309	2019-10-28 13:06:27	search	8107	2019-10-28	2019-10-28 13:06:45	contacts_show	2019-10-28

In [95]:
```
def make_delta(row):
    """
    Возвращает разницу search и contacts_show
    """
    if row['time2'] > row['time1']:
        delta = round(((row['time2'] - row['time1']).seconds/60), 2)
        return delta
    return 0
```

In [96]:
```
msc['delta_sc'] = msc.apply(make_delta, axis=1)
msc['delta_sc'].describe()
```

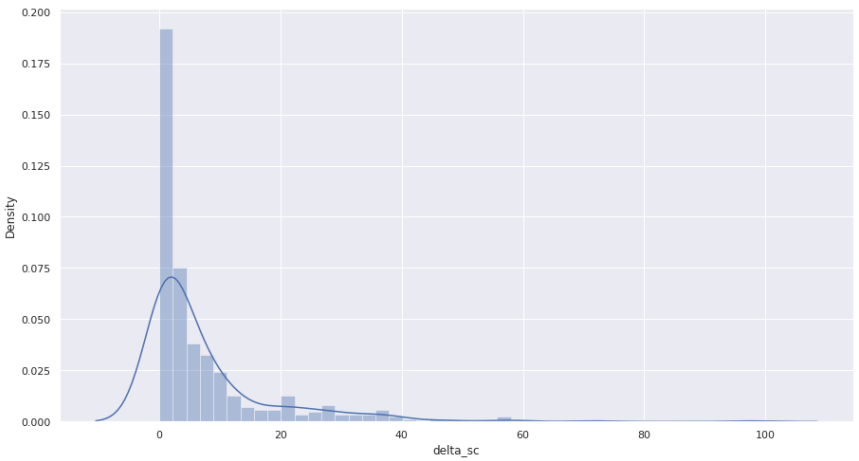
Out[96]:
```
count    388.000000
mean       7.460902
std       11.502744
min        0.000000
25%        0.677500
50%        3.090000
75%        8.880000
max       98.100000
Name: delta_sc, dtype: float64
```

In [97]:
```
mean = msc['delta_sc'].mean().round()
display('Средняя длительность поиска до перехода в просмотр контактов в минутах = {}'.format(mean))
```

```
'Средняя длительность поиска до перехода в просмотр контактов в минутах = 7.0'
```

In [98]:
```
sns.distplot(msc['delta_sc'])
plt.show()
```

```
/opt/conda/lib/python3.9/site-packages/seaborn/distributions.py:2557: FutureWarning:

`distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
```



Максимальный выброс в 98 минут дает среднее в 7 минут, тогда как медиана равна 3 минуты.

Среднее время перехода пользователя от photos_show к contacts_show в рамках сессии

In [99]:
```
mp=m.query('event_name == "photos_show"').rename(columns = {'event_time':'time2'})
mp = mp.drop(['user_id', 'source', 'diff', 'diff_m', 'diff_s'], axis=1)
mp.shape
```

Out[99]: (2526, 4)

In [100…
```
mpc = mp.merge(mc, on='session_id')
mpc = mpc.rename(columns = {'time2_x':'time1', 'time2_y':'time2'})
```

In [101…
```
mpc['delta_pc'] = mpc.apply(make_delta, axis=1)
```

In [102…
```
mean = mpc['delta_pc'].mean().round()
display('Средняя длительность перехода из фотопросмотра в просмотр контактов в минутах = {}'.format(mean))
```

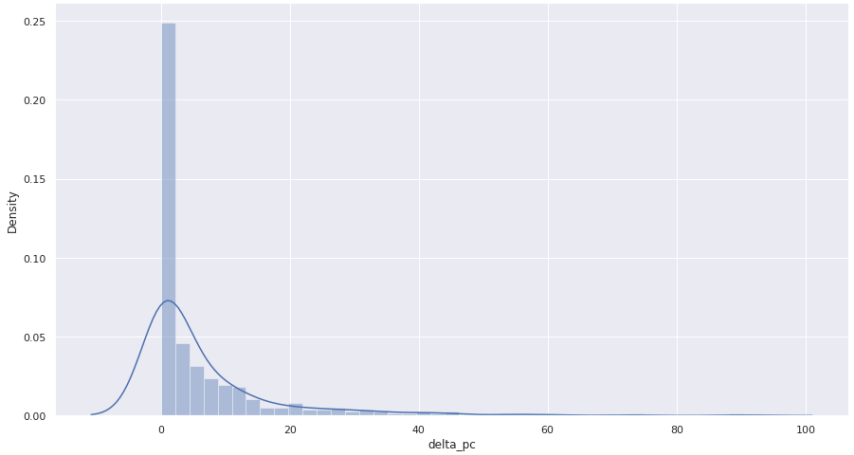
'Средняя длительность перехода из фотопросмотра в просмотр контактов в минутах = 6.0'

In [103…
```
mpc['delta_pc'].describe()
```

Out[103…
```
count    347.000000
mean       6.488934
std       11.611226
min        0.000000
25%        0.000000
50%        1.450000
75%        7.885000
max       90.250000
Name: delta_pc, dtype: float64
```

In [104…
```
sns.distplot(mpc['delta_pc'])
plt.show()
```

```
/opt/conda/lib/python3.9/site-packages/seaborn/distributions.py:2557: FutureWarning:

`distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
```



Здесь из-за большого максимума при среднем времени в 6,5 минут, медиана равна 1,5 минуты.

Среднее время перехода пользователя от tips_show к contacts_show в рамках сессии

In [105…
```
mt=m.query('event_name == "tips_show"').rename(columns = {'event_time':'time2'})
mt = mt.drop(['user_id', 'source', 'diff', 'diff_m', 'diff_s'], axis=1)
mt.shape
```

Out[105… (6035, 4)

In [106…
```
mtc = mt.merge(mc, on='session_id')
mtc = mtc.rename(columns = {'time2_x':'time1', 'time2_y':'time2'})
```

In [107…
```
mtc['delta_tc'] = mtc.apply(make_delta, axis=1)
```

In [108…
```
mean = mtc['delta_tc'].mean().round()
display('Средняя длительность перехода от рекомендаций в просмотр контактов в минутах = {}'.format(mean))
```

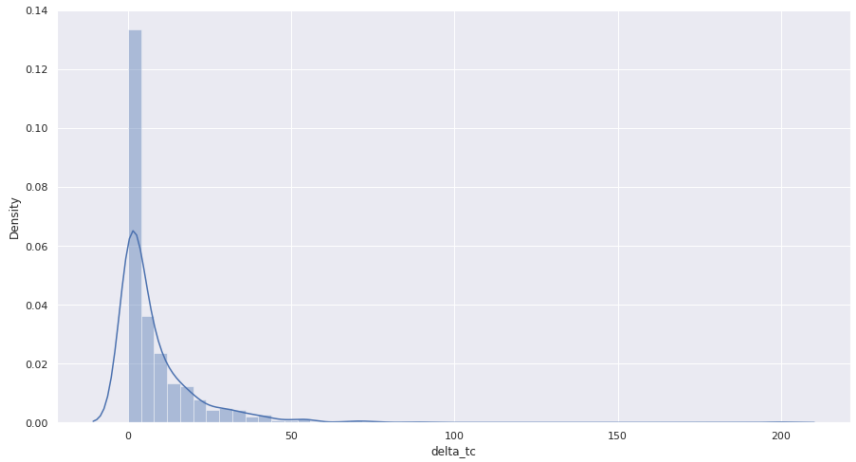
'Средняя длительность перехода от рекомендаций в просмотр контактов в минутах = 8.0'

In [109…
```
mtc['delta_tc'].describe()
```


Out[109…
```
count    849.000000
mean       8.405889
std       13.699454
min        0.000000
25%        0.420000
50%        3.330000
75%       10.950000
max      199.520000
Name: delta_tc, dtype: float64
```

In [110…
```
sns.distplot(mtc['delta_tc'])
plt.show()
```

```
/opt/conda/lib/python3.9/site-packages/seaborn/distributions.py:2557: FutureWarning:

`distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
```



Максимум в 200 минут говорит о какой-то проблеме в регистрации времени событий просмотра рекомендаций. Среднее время 8,4 при медиане 3,3 минуты.

Вывод

Среднее время перехода пользователя к целевому действию (просмотру контактов) в рамках сессии в минутах:

- от поиска: 7 в среднем и 3 по медиане;
- от просмотра фото: 6,5 в среднем и 1,5 по медиане;
- от просмотра рекомендаций: 8,4 при медиане 3,3 минуты.
  Видим, что фото быстрее всего помогают определиться с выбором. Рекомендации похоже работают не так, как ожидалось, судя по времени перехода в целевое действие и редкости кликов в рекомендациях. Возможно, недостаточно персонализированы.

Анализ пользовательских метрик

Рассчитаем метрики пользовательской активности — количество уникальных активных пользователей за определённое время. На основании имеющихся данных мы можем считать за дни и неделю.

In [111…
```
mobile['week'] = mobile['event_time'].dt.isocalendar().week
mobile.sample(5)
```

Out[111…

	event_time	event_name	user_id	source	diff	diff_s	diff_m	session_id	day	week
24078	2019-10-16 21:56:18	advert_open	8270324d-d1ec-49cf-ad44-a4d3edc40f93	google	0 days 00:00:06	6.0	0.10	5201	2019-10-16	42
8332	2019-10-10 15:31:31	tips_show	5622bed3-de57-4e44-851a-89cb6a2e7596	yandex	0 days 00:02:13	133.0	2.22	3452	2019-10-10	41
47954	2019-10-25 18:52:06	tips_show	28af1ea4-9be3-4e9e-be07-1838cfe0e1cb	other	0 days 00:01:02	62.0	1.03	1797	2019-10-25	43
39996	2019-10-23 09:41:01	map	691518e6-4781-4afd-80e2-e8180af6c93b	yandex	0 days 00:00:21	21.0	0.35	4213	2019-10-23	43
13991	2019-10-13 14:12:19	advert_open	c140f88a-c544-4ce6-a6bd-578a1a0d1b18	yandex	0 days 00:00:42	42.0	0.70	7783	2019-10-13	41

DAU — количество уникальных пользователей в день:

In [112…
```
dau = round(mobile.groupby('day').agg({'user_id': 'nunique'}))
dau_m = dau.mean()
dau.describe()
```

Out[112…

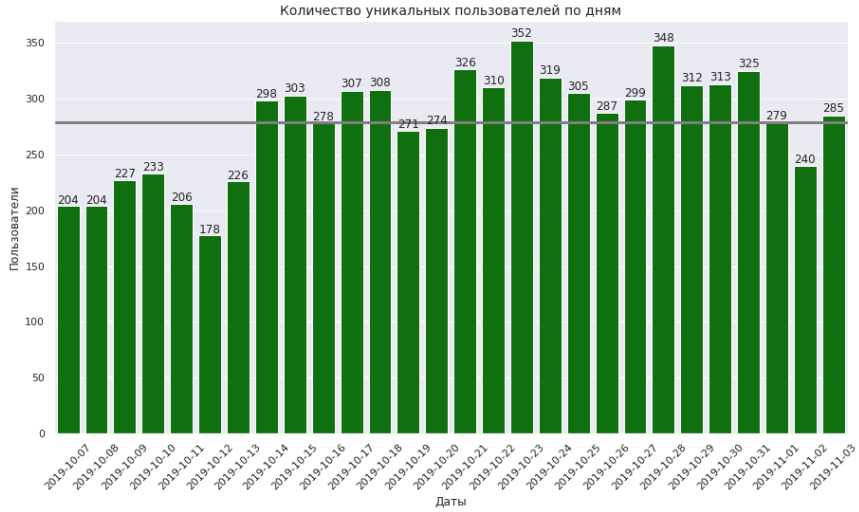
	user_id
count	28.000000

	user_id
mean	279.178571
std	46.737291
min	178.000000
25%	238.250000
50%	292.500000
75%	310.500000
max	352.000000

In [113…
```
dau = dau.reset_index()
```

In [114…
```
average = dau['user_id'].mean()
```

In [115…
```
sns.set(rc={'figure.figsize':(15, 8)})
ax = sns.barplot(x='day', y='user_id', data=dau, color='green')
ax.set_title('Количество уникальных пользователей по дням', fontsize=14)
plt.xticks(rotation=45)
plt.xlabel('Даты')
plt.ylabel('Пользователи')

plt.axhline(y=average, color = 'grey', linewidth=3)

for p in ax.patches:
    _x = p.get_x() + p.get_width() / 2
    _y = p.get_y() + p.get_height() + (p.get_height()*0.01)
    value = '{:.0f}'.format(p.get_height())
    ax.text(_x, _y, value, ha="center")
plt.show()
```



WAU — количество уникальных пользователей в неделю:

In [116…
```
wau = round(mobile.groupby(['week']).agg({'user_id': 'nunique'}))
wau_m = wau.mean()
wau.describe()
```


Out[116…

	user_id
count	4.000000
mean	1382.500000
std	177.661663
min	1130.000000
25%	1344.500000
50%	1427.000000
75%	1465.000000
max	1546.000000

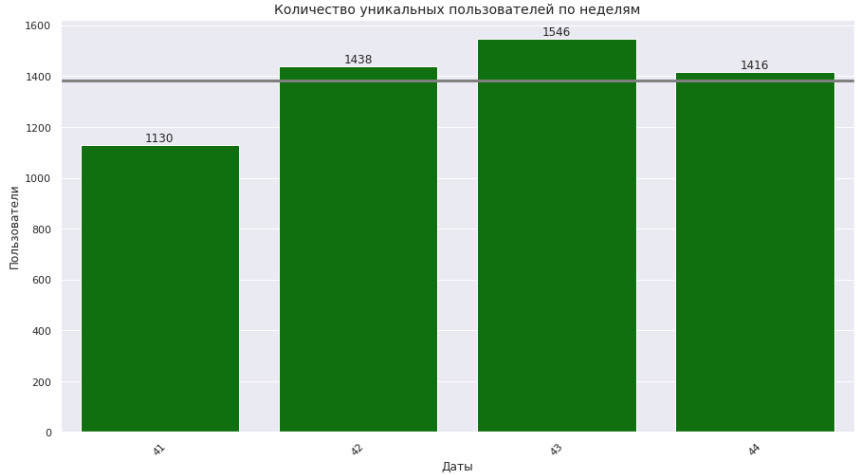
In [117…
```
wau = wau.reset_index()
wau
```

Out[117…

	week	user_id
0	41	1130
1	42	1438
2	43	1546
3	44	1416

In [118…
```
average = wau['user_id'].mean()
```

In [119…
```
sns.set(rc={'figure.figsize':(15, 8)})
ax = sns.barplot(x='week', y='user_id', data=wau, color='green')
ax.set_title('Количество уникальных пользователей по неделям', fontsize=14)
plt.xticks(rotation=45)
plt.xlabel('Даты')
plt.ylabel('Пользователи')

plt.axhline(y=average, color = 'grey', linewidth=3)

for p in ax.patches:
    _x = p.get_x() + p.get_width() / 2
    _y = p.get_y() + p.get_height() + (p.get_height()*0.01)
    value = '{:.0f}'.format(p.get_height())
    ax.text(_x, _y, value, ha="center")
plt.show()
```



Количество уникальных пользователей в день и за неделю достаточно стабильно. В среднем приходит 279 пользователей в день и 1382 в неделю. Как и в распределении по сессиям, количество уникальных пользователей по дням и неделям максимально на 2 и 3 неделе, то есть в середине месяца и ближе к завершению.

Sticky factor отражает регулярность использования приложения для недельной аудитории:

In [120…
```
sticky_factor = round(dau_m/wau_m * 100)
sticky_factor
```

Out[120…
```
user_id    20.0
dtype: float64
```

20 % клиентов возвращаются в приложение в течение недели.

Посчитаем конверсию в целевое событий в зависимости от вида источника установки приложения, чтобы проверить в дальнейшем, есть ли статистически значимая разница конверсий из источников Яндекс и Гугл. Посчитаем количество уникальных пользователей, дошедших до целевого действия, по источникам установки.

In [121…
```
user_source2 = mobile.query('event_name == "contacts_show"').pivot_table(index=['source'], values='user_id', aggfunc='nunique').reset_
user_source2.rename(columns={'user_id': 'cont_sum'}, inplace=True)
```

In [122…
```
source_conv = user_source.merge(user_source2, on='source')
source_conv['part'] = round(((source_conv['cont_sum']/source_conv['user_sum'])*100), 2)
source_conv = source_conv.sort_values(by='part', ascending = False)
source_conv
```

Out[122…

	source	user_sum	cont_sum	part
2	yandex	1934	478	24.72
0	google	1129	275	24.36
1	other	1230	228	18.54

In [123…
```
plt.subplot(2,1,1)
#This will create the bar graph for poulation
ax1 = plt.bar(source_conv['source'], source_conv['user_sum'])
plt.ylabel('Установки')
plt.xticks([],[])
#The below code will create the second plot.
plt.subplot(2,1,2)
#This will create the bar graph for gdp i.e gdppercapita divided by population.
ax2 = plt.bar(source_conv['source'], source_conv['part'])
plt.ylabel('Конверсия')
plt.xticks(source_conv['source'], rotation='vertical')
plt.show()
```

Конверсия из основных источников отличается меньше, чем на 1 %. При этом установок из Яндекс больше, чем из гугл. В следующем пункте попробуем выяснить, есть ли значимая разница. Также четко прослеживается, что конверсия из Гугл, лучше, чем из остальных неименованных источников, хотя первоначальных установок в Гугл - наименьшее количество.

Вывод

- количество уникальных пользователей в день и за неделю достаточно стабильно. В среднем приходит 279 пользователей в день и 1382 в неделю. Но было бы правильнее измерять пользователей в месяц, т.к. наше приложение большинством пользователей используется не часто и не регулярно. Для этого нужны данные за более длительный период;
- 20 % клиентов возвращаются в приложение в течение недели, это может быть не так уж плохо, т.к. наше приложение не предполагает регулярного использования большинством пользователей. Но для корректных выводов интересно посмотреть лояльность пользователей за месяц и в динамике;
- конверсия из основных источников: Яндекс и Гугл отличается меньше, чем на 1 %. При этом установок из Яндекс больше, чем из гугл. А конверсия из Гугл, лучше, чем из остальных неименованных источников, хотя первоначальных установок в Гугл - наименьшее количество.

Проверка статистических гипотез

Проверка гипотезы № 1

Проверить статистическую гипотезу 1.
Одни пользователи совершают действия `tips_show` и `tips_click` , другие — только `tips_show` .
Проверим гипотезу:
H0 - конверсия в просмотры контактов не различается у этих двух групп.
H1 - конверсия в просмотры контактов у этих двух групп различается.

Воспользуемся проверкой гипотезы о равенстве долей. Зададим уровень статистической значимости 5 %.

Выделим две экспериментальные группы 'tips_sc' и 'tips_s'. И рассчитаем конверсию перехода в целевое действие в каждой из них (для уникальных пользователей).

сначала нужно получить список юзеров, которые совершали действие tips_show. Потом из них оставить только тех юзеров, которые так же совершали действия tips_click. Тогда количество этих юзеров, это будет размер группы, которые совершали tips_show и tips_clcik. Теперь из уже этого списка юзеров, мы оставим только тех юзеров, которые совершали действие contacts_show. С размером первой группы и количеством успехов разобрались. Теперь ко второй группе: Снова получаем список юзеров, которые совершали действие tips_show, но в этот раз из него нужно исключить юзеров, которые совершали действие tips_click. Это будет размер группы, которые совершали только tips_show. Теперь уже из них нужно оставить только тех юзеров, которые совершали contacts_show. Таким образом найдем размер второй группы и количество успехов

Количество пользователей, посмотревших рекомендации (среди них есть и кликнувшие по рекомендациям):

In [124…
```
tips_show = mobile.query('event_name == "tips_show"')
tips_show2 = tips_show['user_id'].nunique()
tips_show2
```

Out[124… 2801

Добавим их идентификаторы в список:

In [125…
```
show = tips_show['user_id'].tolist()
```

In [126…
```
tips_show_click = mobile.query('event_name == "tips_click" and user_id in @show')
tips_show_click2 = tips_show_click['user_id'].nunique()
tips_show_click2
```

Out[126… 297

In [127…
```
show_click = tips_show_click['user_id'].tolist()
```

297 пользователей просмотрели и кликнули рекомендации. Сколько из них из них смотрели контакты:

In [128…
```
contacts_show_click = mobile.query('event_name == "contacts_show" and user_id in @show_click')
contacts_show_click = contacts_show_click['user_id'].nunique()
contacts_show_click
```

Out[128… 91

Посчитаем, сколько пользователей только смотрели, но не кликнули рекомендации, для этого из смотревших вычтем тех, кто смотрел и кликнул:

In [129…
```
show_only =  mobile.query('event_name == "tips_show" and user_id not in @show_click')
show_only2 = show_only['user_id'].nunique()
show_only2
```

Out[129… 2504

2504 пользователя смотрели рекомендации (но не кликали). Выделим сколько из них перешли в просмотр контактов:

In [130…
```
so = show_only['user_id'].tolist()
contacts_show = mobile.query('event_name == "contacts_show" and user_id in @so')['user_id'].nunique()
contacts_show
```

Out[130… 425

2504 пользователя смотрели рекомендации (но не кликали). 425 из них смотрели контакты.

Соберем данные по тестовым группам в таблицу:

In [131…
```
data = {'event_name': ['tips', 'contact'], 'tips_sc': [tips_show_click2, contacts_show_click], 'tips_s': [show_only2, contacts_show]}

# DataFrame
data = pd.DataFrame(data)
data
```

Out[131…

	event_name	tips_sc	tips_s
0	tips	297	2504
1	contact	91	425

In [132…
```
alpha = 0.05 # критический уровень статистической значимости
# пропорция успехов в первой группе tips_show_cont:
p1 = data.iloc[1][1] / data.iloc[0][1]
# пропорция успехов во второй группе tips_show_click_cont:
p2 = data.iloc[1][2] / data.iloc[0][2]
# пропорция успехов в комбинированном датасете:
p_combined = ((data.iloc[1][1]  + data.iloc[1][2]) /
                      (data.iloc[0][1] + data.iloc[0][2]))
# разница пропорций в датасетах
difference = p1 - p2
# считаем статистику в ст.отклонениях стандартного нормального распределения
z_value = difference / mth.sqrt(p_combined * (1 - p_combined) *
                                (1/data.iloc[0][1] + 1/data.iloc[0][2]))
# задаем стандартное нормальное распределение (среднее 0, ст.отклонение 1)
distr = st.norm(0, 1)
p_value = (1 - distr.cdf(abs(z_value))) * 2
print('p-значение: {}'.format(p_value))
if(p_value < alpha):
    print("Отвергаем нулевую гипотезу: между долями есть значимая разница")
else:
    print("Не получилось отвергнуть нулевую гипотезу, нет оснований считать доли разными")
print('')
```

```
p-значение: 9.218316554537864e-09
Отвергаем нулевую гипотезу: между долями есть значимая разница
```

Проверим конверсию этих двух групп расчетом:

In [133…
```
p1 = round(((data.iloc[1][1] / data.iloc[0][1])*100), 2)
p2 = round(((data.iloc[1][2] / data.iloc[0][2])*100), 2)
display('Конверсия группы tips_show_cont', p1, 'Конверсия группы tips_show_click_cont', p2 )
```

```
'Конверсия группы tips_show_cont'
30.64
'Конверсия группы tips_show_click_cont'
16.97
```

Вывод по проверке гипотезы

Конверсия группы просмотров и кликов в контакты существенно отличается от конверсии группы просмотров рекомендаций. По расчетам конверсия кликов и просмотров почти в 2 раза меньше конверсии просмотров рекомендаций в целевое действие.

Проверка гипотезы № 2

Предполагаем, что основные источники установки приложения - Яндекс и Гугл - равнопопулярны, значит и конверсия пользователей из этих источников примерно одинакова. Воспользуемся проверкой гипотезы о равенстве долей. Зададим уровень статистической значимости 5 %.
Проверим гипотезу:
H0 - конверсия в целевое событие (от начального события) у пользователей из Яндекса не отличается от конверсии пользователей, установивших приложение из Гугл.
H1 - конверсии пользователей, установивших приложение из этих источников, различаются.

Вывод источника данных для проверки. Проверка гипотезы.

In [134…
```
source_conv.set_index('source').T
```

Out[134…

source	yandex	google	other
user_sum	1934.00	1129.00	1230.00
cont_sum	478.00	275.00	228.00
part	24.72	24.36	18.54

In [135…
```
alpha = 0.05 # критический уровень статистической значимости
# пропорция успехов в первой группе:
p1 = source_conv.loc[0][2] / source_conv.loc[0][1]
# пропорция успехов во второй группе:
p2 = source_conv.loc[2][2] / source_conv.loc[2][1]
# пропорция успехов в комбинированном датасете:
p_combined = ((source_conv.loc[0][2]  + source_conv.loc[2][2]) /
                      (source_conv.loc[0][1] + source_conv.loc[2][1]))
# разница пропорций в датасетах
difference = p1 - p2
# считаем статистику в ст.отклонениях стандартного нормального распределения
z_value = difference / mth.sqrt(p_combined * (1 - p_combined) *
                                (1/source_conv.loc[0][1] + 1/source_conv.loc[2][1]))
# задаем стандартное нормальное распределение (среднее 0, ст.отклонение 1)
distr = st.norm(0, 1)
p_value = (1 - distr.cdf(abs(z_value))) * 2
print('p-значение: {}'.format(p_value))
if(p_value < alpha):
    print("Отвергаем нулевую гипотезу: между долями есть значимая разница")
else:
    print("Не получилось отвергнуть нулевую гипотезу, нет оснований считать доли разными")
print('')
```

```
p-значение: 0.8244316027993777
Не получилось отвергнуть нулевую гипотезу, нет оснований считать доли разными
```


In [136…
```
source_conv
```

Out[136…

	source	user_sum	cont_sum	part
2	yandex	1934	478	24.72
0	google	1129	275	24.36
1	other	1230	228	18.54

Вывод по проверке гипотезы

Подтвердилась нулевая гипотеза, о том, что конверсия пользователей из источников Яндекс и Гугл примерно одинакова. Что подтверждается ранее проведенными расчетами.

ВЫВОДЫ И РЕКОМЕНДАЦИИ:

1. В ходе предобработки данных:

- проверены массивы на пропуски и явные дубликаты;
- откорректированы названия столбцов;
- удалены неявные дубликаты в значениях столбца "event_name";
- объединены в одно 7 событий "search";
- изменен тип данных в столбце времени событий, время округлено до секунд;
- объединены события пользователей, произошедшие в один момент времени;
- массивы объединены в один по идентификаторам пользователей.

Обнаружено 2183 строки с дубликатами, где один и тот же пользователь совершает в течение секунды одно и то же действие несколько раз. Возможно, это связано с тем, что события фиксируются системой несколько раз в секунду. Пользователь вряд ли сможет совершить больше одного уникального действия в секунду.

2. Анализ данных

2.1. Уточнение данных:

- у нас есть данные с 7 октября по 3 ноября 2019 года - примерно за месяц;
- больше всего среди зарегистрированных событий - рекомендаций. Видимо, это связано с тем, что рекомендации выдаются на любом этапе работы с системой, их показ инициируется не самим пользователем. На втором месте по частоте - просмотр фото, который уже зависит от действий самого пользователя. Далее все действия по поиску. И уже потом открытие карточки конкретного объявления и просмотр контактов;
- 3 источника установок: яндекс, другие и гугл - в порядке количества установок из ник. Пользователи по источникам не пересекаются;
- за один месяц пользователь совершает около 9 действий в приложении (по медиане), в целом от 5 до 17, а в среднем без учета выбросов 17 действий. Минимально - 1 действие, максимум 472. Значения в списке отличаются от среднего почти на 29. Активность пользователей довольно сильно отличается;
- с учетом повторяемости действий в рамках сессии пользователь совершает по медиане 4 действия, в среднем без учета выбросов - 7. В большинстве случаев от 2 до 8;
- медианное время действия колеблется около минуты: от 1 секунды у кликов по рекомендациям и открытий карточек, до 2,4 минут у звонков. Просмотр фото и рекомендаций по медиане 1,3 минуты, поиск довольно быстрый - меньше минуты. Максимальное время звонка самое большое - 9 минут;
- количество пользователей = 4293. Количество сессий = 10368. Среднее количество сессий на пользователя за 28 дней = 2, по медиане 1;
- количество сессий наибольшее примерно с середины до конца октября. Есть тенденция к спаду активности в начале месяца. Но чтобы убедиться в этом нужны данные за несколько месяцев и более. В среднем в день 352 сессии.

2.2. Выделение сценариев использования приложения в рамках сессий

Построена диаграмма Сэнкей для выделения сценариев действий. По диаграмме видно, что большая часть пользователей начинает с поиска, в целевой просмотр контактов они попадают из поиска напрямую, через фотопросмотр и просмотр рекомендаций. Также популярны сценарии tips_show-contacts_show и photos_show-contacts_show. Большая часть пользователей явно останавливается на 2-3 шагах.

2.3. Выявление самых распространенных событий, определение конверсии перехода в целевое действие по сценарию

Самое большое количество событий на пользователя: просмотр рекомендаций, поиск и просмотр на карте. Просмотр фото и контактов - замыкают топ-5. Открытие карточки события на 6 месте.Эти события рассчитаны в разрезе совершения каждого из них пользователем без учета повторений. Поэтому просмотр фото и открытие карточки сместились ниже по сравнению с зарегистрированным количеством событий в периоде. То есть мы не учитываем, что пользователь просматривает фото, открывает карточки и осуществляет поиск не по одному разу за время работы с системой. Чаще всего также показываются рекомендации, затем осуществляется просмотр фото, поиск, открытие карточек и просмотр контактов. Самая высокая прямая конверсия из просмотра фото и рекомендаций.

Конверсия по выделенным сценариям:

- по сценарию tips_show - contacts_show конверсия перехода в целевое действие составляет 16 %;
- по сценарию tips_show-map-contacts_show: 21 % пользователей из просмотра рекомендаций переходят к просмотру карты. С шага просмотра карты до целевого действия доходит 6 % пользователей. А от просмотра рекомендаций к просмотру карты, а затем к просмотру контактов переходит только 1 % пользователей;
- по сценарию search-contacts_show конверсия пользователей из поиска в просмотр контактов 8 %;
- по сценарию search-photos_show-contacts_show: 31 % пользователей переходят от поиска к просмотру фото. 41 % смотревших фото переходят к контактам. А по цепочке: поиск-фото-контакты переходит 13 % пользователей;
- по сценарию photos_show-contacts_show: больше всего пользователей переходят к просмотру контактов из просмотра фото - 19 %.

2.4. Среднее время перехода пользователей к целевому действию в рамках сессии

Среднее время перехода пользователя к целевому действию (просмотру контактов) в рамках сессии в минутах:

- от поиска: 7;
- от просмотра фото: 6;
- от просмотра рекомендаций: 8. Видим, что фото быстрее всего помогают определиться с выбором. Рекомендации похоже работают не так, как ожидалось, судя по времени перехода в целевое действие и редкости кликов в рекомендациях. Возможно, недостаточно персонализированы.

2.5. Анализ пользовательских метрик

- количество уникальных пользователей в день и за неделю достаточно стабильно. В среднем приходит 279 пользователей в день и 1382 в неделю. Но было бы правильнее измерять пользователей в месяц, т.к. наше приложение большинством пользователей используется не часто и не регулярно. Для этого нужны данные за более длительный период;
- 20 % клиентов возвращаются в приложение в течение недели, это может быть не так уж плохо, т.к. наше приложение не предполагает регулярного использования большинством пользователей. Но для корректных выводов интересно посмотреть лояльность пользователей за период в полгода в динамике;
- конверсия из основных источников: Яндекс и Гугл отличается меньше, чем на 1 %. При этом установок из Яндекс больше, чем из гугл. А конверсия из Гугл, лучше, чем из остальных неименованных источников, хотя первоначальных установок в Гугл - наименьшее количество.

3. Проверка гипотез

Гипотеза 1: Одни пользователи совершают действия tips_show и tips_click, другие — только tips_show. Проверим гипотезу: H0 - конверсия в просмотры контактов не различается у этих двух групп.
В результате проверки выяснилось: конверсия группы просмотров и кликов в контакты существенно отличается от конверсии группы просмотров рекомендаций. По расчетам конверсия кликов и просмотров почти в 2 раза меньше конверсии просмотров рекомендаций в целевое действие.

Гипотеза 2: Предполагаем, что основные источники установки приложения - Яндекс и Гугл - равнопопулярны, значит и конверсия пользователей из этих источников примерно одинакова. Воспользуемся проверкой гипотезы о равенстве долей. Зададим уровень статистической значимости 5 %. Проверим гипотезу: H0 - конверсия в целевое событие (от начального события) у пользователей из Яндекса не отличается от конверсии пользователей, установивших приложение из Гугл.
В результате проверки подтвердилась нулевая гипотеза, о том, что конверсия пользователей из источников Яндекс и Гугл примерно одинакова. Что подтверждается ранее проведенными расчетами.

РЕКОМЕНДАЦИИ

- получить данные за более длительный период, чтобы рассмотреть показатели успешности приложения в динамике;
- уточнить варианты поиска, вдруг они кардинально разные, может потребоваться более глубокий анализ;
- уточнить, как производится фиксация действий пользователя в системе, с какой периодичностью. Возможны ошибки логирования, связанные с большим количеством повторов в рамках сессии;
- уточнить источники установок "другие", возможно, среди них есть перспективные для развития;
- проверить удобство поиска, может search1-7 не варианты, а последовательность? Длительность перехода к цели в 7 минут, при том что сам поиск в среднем 0,6 минуты. Максимум в 98 минут. Конверсия из поиска в контакты 8%. Есть над чем работать;
- проработать удобство совершения звонков напрямую из приложения, это удобнее, чем копировать контакты, даст больше информации для анализа;
- уточнить, каким образом контакты, избранное и просмотр фото могут быть доступны в качестве первого шага сессии пользователя;
- выделить вход в приложение отдельным стартовым действием, это поможет установить более точные границы и длительность сессии.

Большая часть переходов к просмотру контактов осуществляется из просмотра рекомендаций, а также просмотра фото. При этом переход от просмотра рекомендаций к просмотру контактов занимает 8,4 минуты в среднем при медиане 3,3 минуты. Очень велик максимум, что может говорить об ошибках в регистрации времени событий. Для увеличения вовлеченности пользователей стоит персонализировать просмотр рекомендаций (очень мало кликов по ним), увеличить количество объявлений с фото (самое меньшее время перехода к контактам из рассмотренных). Стоит поработать над добавлением в избранное, очень мало кто пользуется этой функцией, хотя это очевидные переход в покупки. Переход от карты к контактам - посмотреть, можно ли сделать его более удобным и быстрым.