Мобильные приложения — Анализ поведения пользователей

Цели исследования:

- управление вовлеченностью клиентов(адаптация приложений по целевой и смежной аудитории). Для этого необходимо понять какая аудитория в принципе
- получить на основе поведения пользователей гипотезы о том, как можно было бы улучшить приложение с т.зр. пользовательского опыта. Дать рекомендации.

Задачи:

1. Проанализируйте связь целевого события — просмотра контактов — и других действий пользователей.
2. Оцените, какие действия чаще совершают те пользователи, которые просматривают контакты.

- Проведите исследовательский анализ данных
- Проанализируйте влияние событий на совершение целевого события
- Проверьте статистические гипотезы
    1. Одни пользователи совершают действия tips_show и tips_click , другие — только tips_show . Проверьте гипотезу: конверсия в просмотры контактов различается у этих двух групп.
    2. Сформулируйте собственную статистическую гипотезу. Дополните её нулевой и альтернативной гипотезами. Проверьте гипотезу с помощью статистического теста.

Материалы

- Презентация

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.

Колонки в _mobile*sources.csv* :
`userId` — идентификатор пользователя,
`source` — источник, с которого пользователь установил приложение.
Колонки в _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.

Выполнение проекта

Шаг 1. Открыть _mobile*dataset.csv* и _mobile*sources.csv*, изучить общую информацию.

- проверить на наличие пропусков, явных и неявных дубликатов. Откорректировать, если нужно
- проверить, что заявленные данные соответствуют предоставленному описанию
- заменить названия столбов на удобные
- сколько всего уникальных пользователей в каждом датасете?
- посмотреть на события столбца `event_name` . По результатам найти прожатие обычной кнопки поиска в _search*1..7*

Шаг 2. Предобработка

- удалить _tips*show* из `event_name` для шага анализа данных, сохранить копию исходного датасета для проверки гипотез
- разделить _search*1..7* на простой и остальной поиск

Шаг 3. Анализ данных

- оценить активность по дням недели через гистограмму
- посчитать retantional rate этого месяца
- посчитать DAU. Построить график.
- оценить количесво приходящих новых пользователей за каждый день. Построить график.
- скопировать датасет. Оценить тайм-аут, не являющийся выбросом. Создать столбец с `session_id` на основании тайм-аута - каждой новой сессии присваивается порядковый номер
- построить диаграмму Сэнкей по `session_id`. Выбрать от 3 до 5 наиболее распространенных сценария прохода пользователей по приложению.
- по выбранным сценариям по `user_id` построить воронки. Сделать выводы
- Оценить какие действия чаще всего совершают те пользователи, которые просматривают контакты:рассчитать относительную частоту событий в разрезе двух групп пользователей: группа, которые смотрели contacts_show и не смотрели

Шаг 4. Проверка гипотез

- Одни пользователи совершают действия _tips*show* и _tips*click* , другие — только _tips*show* . Проверить гипотезу: конверсия в просмотры контактов различается у этих двух групп. Для этого выделить пользователей, которые совершили и _tips*show*, и _tips*click*, и _contacts*show* в группу A, _tips*show* и _contacts*show* в группу B.
- аналогично проверить, что конверсия пользователей, просмотревших фотографии, в просмотр контактов больше, чем конверсия пользователей, не просмотревших

Импорт библиотек

In [1]:
```
import pandas as pd
import numpy as np
import math as mth # для рассчета статистики ст.отклонениях стандартного нормального распределения
from scipy import stats as st #для проверки гипотезы 1
from datetime import datetime, timedelta
import locale # для переключения дат на русский (н-р, пн/вт/октябрь)
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm # помощь при построении диаграммы Сэнкей
from plotly import graph_objects as go # графическая часть построения диаграммы Сэнкей
```

In [2]:
```
# если раскомментировать строку - Даты будут выведены на русском
#locale.setlocale(locale.LC_ALL, 'ru_RU.UTF-8')
```

Общий обзор дынных

In [3]:
```
def watch_basics(df):
    """
    Отображает базовую информацию о датасете
    """
    p = 20
    print('*'*p, 'Общая информация', '*'*p)
    display(df.head())
    df.info()
    print('*'*p, 'Пропуски', '*'*p)
    if not (df.isna().sum() > 0).any():
        print('*'*p, 'Пропусков не найдено', '*'*p)
    else:
        display(df.isna().sum())
    print('*'*p, 'Явные дубликаты', '*'*p)
    display(df[df.duplicated()].count())
```

Выгрузка _mobile*dataset.csv*

In [4]:
```
dataset = pd.read_csv('mobile_dataset.csv', dtype={'user.id': 'string', 'event.name': 'string'}, parse_dates=['eve
watch_basics(dataset)
```

```
********** Общая информация **********
```

	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
2	2019-10-07 00:00:02.245341	tips_show	cf7eda61-9349-469f-ac27-e5b6f5ec475c
3	2019-10-07 00:00:07.039334	tips_show	020292ab-89bc-4156-9acf-68bc2783f894
4	2019-10-07 00:00:56.319813	advert_open	cf7eda61-9349-469f-ac27-e5b6f5ec475c

```
<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  string
 2   user.id     74197 non-null  string
dtypes: datetime64[ns](1), string(2)
memory usage: 1.7 MB
******************** Пропуски ********************
******************** Пропусков не найдено ********************
******************** Явные дубликаты ********************
event.time    0
event.name    0
user.id       0
dtype: int64
```

Очень удачно, что нет пропусков или явных дубликатов. Заранее указали некоторый более экономный(с т.зр. памяти) тип данных. Но есть неявные дубликаты в виде _contacts*show* и _show*contacts*, образованные, скорее всего, из-за слияния нескольких баз данных. Избавимся от них.

In [5]:
```
# сначала переименуем столбцы на более удобные
dataset = dataset.rename(columns={'event.time': 'event_time', 'event.name': 'event_name', 'user.id': 'user_id'})
# избавляемся от неявных дубликатов
dataset.loc[dataset['event_name'] == 'show_contacts', 'event_name'] = 'contacts_show'
```

In [6]:
```
print('Минимальная дата:', dataset['event_time'].min().strftime('%d %b %Y (%a)'))
print('Максимальная дата:', dataset['event_time'].max().strftime('%d %b %Y (%a)'))
# у заведения может быть несколько точек. Посмотрим тогда только на уникальные
print('В датасете представлено', len(dataset['user_id'].unique()), 'уникальных пользователей.')
```

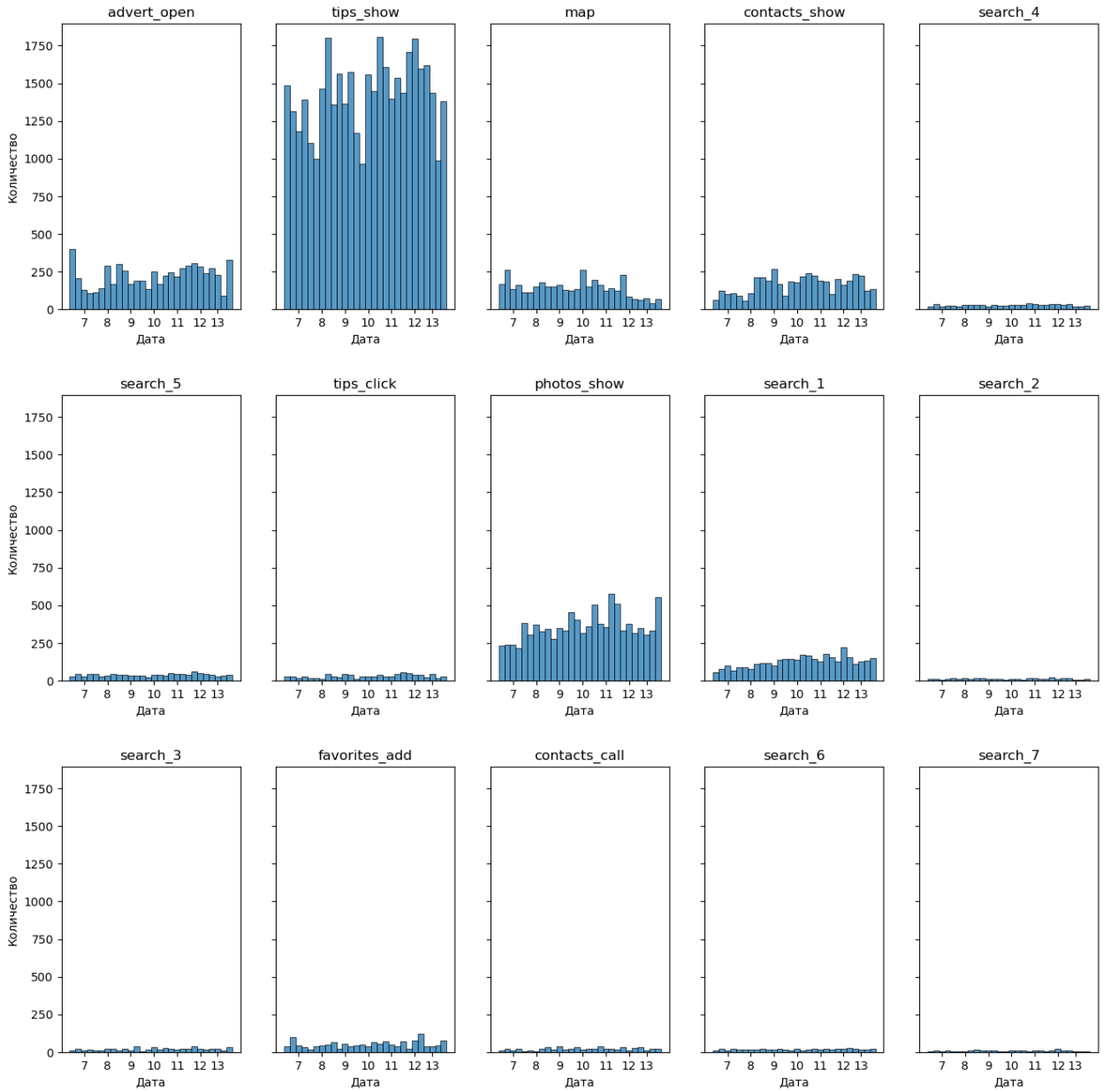
```
Минимальная дата: 07 Oct 2019 (Mon)
Максимальная дата: 03 Nov 2019 (Sun)
В датасете представлено 4293 уникальных пользователей.
```

In [7]:
```
# создадим столбец с датами
dataset['date'] =  pd.DatetimeIndex(dataset['event_time']).date
```

In [8]:
```
import warnings
warnings.filterwarnings('ignore')

events_list = list(dataset['event_name'].unique())
# в датасете 15 возможных событий - разделим
fig,axes = plt.subplots(nrows=3,ncols=5,figsize=(16,16), sharey=True)
num = 0
for i in range(3):
    for j in range(5):
        sett = dataset[dataset['event_name'] == events_list[num]]
        sns.histplot(ax=axes[i][j],data=sett,x='date')
        axes[i][j].set_title(events_list[num])
        axes[i][j].set_xlabel('Дата')
        axes[i][j].set_ylabel('Количество')
        axes[i][j].set_xticklabels(list(pd.DatetimeIndex(dataset['date'].unique()).day))
        #for tick in axes[i][j].get_xticklabels():
        #    tick.set_rotation(35)
        num += 1
# С помощью wspace,hspace можно регулировать расстояния между наборами осей
fig.subplots_adjust(hspace=0.3)
fig.suptitle('Гистограммы по событиям',fontsize=16);
```

Гистограммы по событиям



По результатам доработки ТЗ было выявлено, что tips_show появляется автоматически и может появиться на любом из этапов. Не удивительно, что его так много, к тому же он влияет на весь датасет и последующий анализ. Однако странно, что не логируется заглавная страница: это могло бы стать более точной метрикой количества заходов в приложение, могло бы стать стартовой позицией для большинства воронок, стало бы проще отслеживать пики популярности по загодам в разные дни месяца и/или недели. Рекомендуется вести счет. Так как tips_show выставляется автоматически, а в ТЗ указано провести анализ пользователей, то этап не характеризует действия человека и будет исключен.

Можно заметить, что количество `advert_open` (открыл карточки объявления) гораздо больше, чем `tips_click` (кликнул по рекомендованному объявлению) - похоже, рекомендательной системе стоит уделить внимание.

search_1..7 являются результатами поисковой строки, причем скоре всего бОльшая часть - с фильтрами. Тогда логично предположить, что наибольшее количество - это прожатие строки поиска после ввода запроса - это search_1. Разъединим данные на обычный поиск и специальный.

В contacts_show 3 "гребня волны". В месяц как раз умещается 3-4 недели - стоит в дальнейшем проверить не является ли это регулярной активностью по неделям

Выгрузка _mobile*sources.csv*

In [12]:
```
sources = pd.read_csv('mobile_sources.csv', dtype={'userId': 'string', 'source': 'string'})
watch_basics(sources)
```

```
******************** Общая информация ********************
```

	userId	source
0	020292ab-89bc-4156-9acf-68bc2783f894	other
1	cf7eda61-9349-469f-ac27-e5b6f5ec475c	yandex
2	8c356c42-3ba9-4cb6-80b8-3f868d0192c3	yandex
3	d9b06b47-0f36-419b-bbb0-3533e582a6cb	other
4	f32e1e2a-3027-4693-b793-b7b3ff274439	google

```
<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   string
 1   source  4293 non-null   string
dtypes: string(2)
memory usage: 67.2 KB
******************** Пропуски ********************
******************** Пропусков не найдено ********************
******************** Явные дубликаты ********************
userId    0
source    0
dtype: int64
```

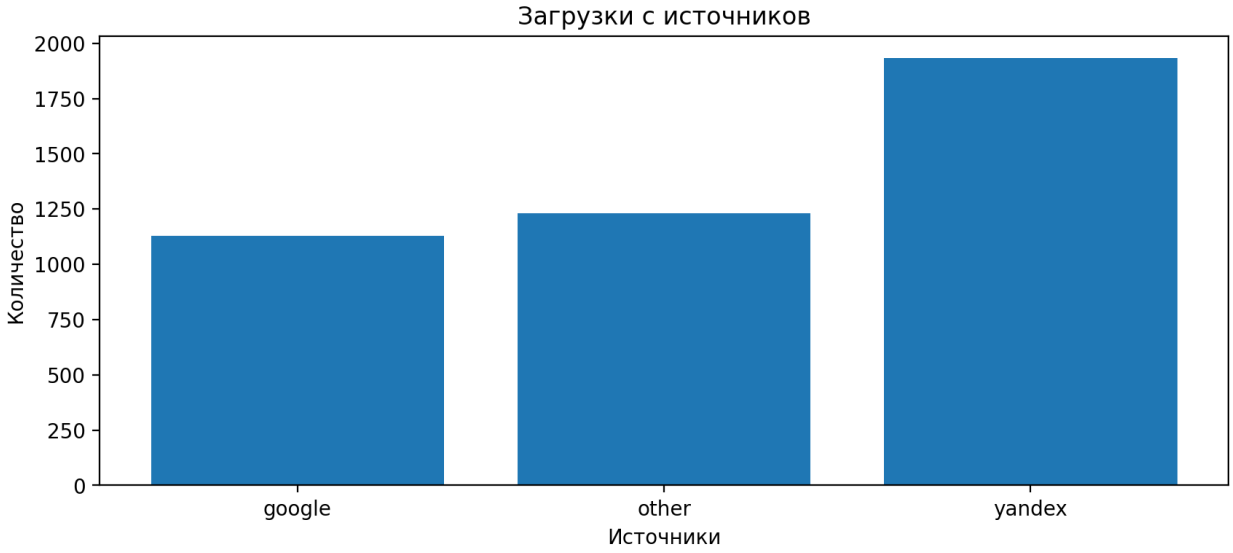
Пропусков и явных дубликатов нет. Привели данные к типу string.

In [13]:
```
sources['source'].unique()
```

Out[13]:
```
<StringArray>
['other', 'yandex', 'google']
Length: 3, dtype: string
```

In [14]:
```
sources = sources.rename(columns={'userId': 'user_id'})
```

In [15]:
```
y = sources.groupby('source')['source'].count()
x = list(y.index)
y = list(y.values)
plt.figure(figsize=(10,4),dpi=200)
plt.bar(x,y)
plt.title('Загрузки с источников')
plt.xlabel("Источники")
plt.ylabel("Количество");
```



К сожалению, имеется только 3 источника. Необычно, что не используются PlayMarket/AppStore/AppGalary или другие платформы для скачивания - мобильные версии на уже исзветсных платформах намного эффективнее, поскольку пользователи им доверяют. К тому же, если ведется запись прихода пользователей с реклам, рекомендуется провести аналитику и с точки зрения эффективности привлечения новых пользователей.

In [16]:
```
print('Количество уникальных пользователей в датасете:', len(sources['user_id'].unique()))
```

```
Количество уникальных пользователей в датасете: 4293
```

Возможно, приложение только запустилось, поскольку количество скачиваний уникальными пользователями и поличество уникальных пользователей в _mobile*dataset.csv* идеально совпадают. В этом также видно хорошие новости: ни

один из них не удалил и не скачал приложение заново(иначе в последних было бы меньше)

Краткий вывод

Переименовали столбцы в обоих датафреймах. Привели столбцы к типу данных string, а *event.time* к datetime. Устранили неявные дубликаты в виде _contacts*show* и _show*contacts*, обозначив как _contacts*show*. Такое произошло, скорее всего, из-за слияния двух баз данных.

Рекомендация: Логировать заглавную страницу. Счет пользователей бы стать более точной метрикой количества заходов в приложение, а также стартовой позицией для большинства воронок, стало бы проще отслеживать как часто пользователи заходят в приложение в разные дни месяца и/или недели.

Можно заметить, что количество `advert_open` (открыл карточки объявления) гораздо больше, чем `tips_click` (кликнул по рекомендованному объявлению) - похоже, рекомендательной системе стоит уделить внимание.

_search*1..7* являются результатами поисковой строки, причем скоре всего бОльшая часть - с фильтрами. Тогда логично предположить, что наибольшее количество - это прожатие строки поиска после ввода запроса - это _search*1*. В предобработке разъединим данные на обычный поиск и специальный.

Предобработка

Мы уже избавились от неявных дубликатов в _mobile*dataset.csv*.

In [17]:
```
# сохраним копию для проверки гипотез
dataset_with_tips_show = dataset.copy()
dataset = dataset[dataset['event_name'] != 'tips_show']
# объединим поиски в простой и специальный
dataset.loc[dataset['event_name'] == 'search_1', 'event_name'] = 'search_simple'
dataset.loc[dataset['event_name'].isin(['search_2', 'search_3', 'search_4', 'search_5', 'search_6', 'search_7']),\
            'event_name'] = 'search_special'
```

Краткий вывод

Создана копия датафрейма с tips_show для проверки гипотез.Однако, tips_show показывается автоматически и является лишним при анализе данных и поведении пользователей, поскольку поведением непосредственно пользователя не является. Переименовать _search*1* в _search*simple*, остальные объединили в search_special.

Анализ данных

Общая конверсия

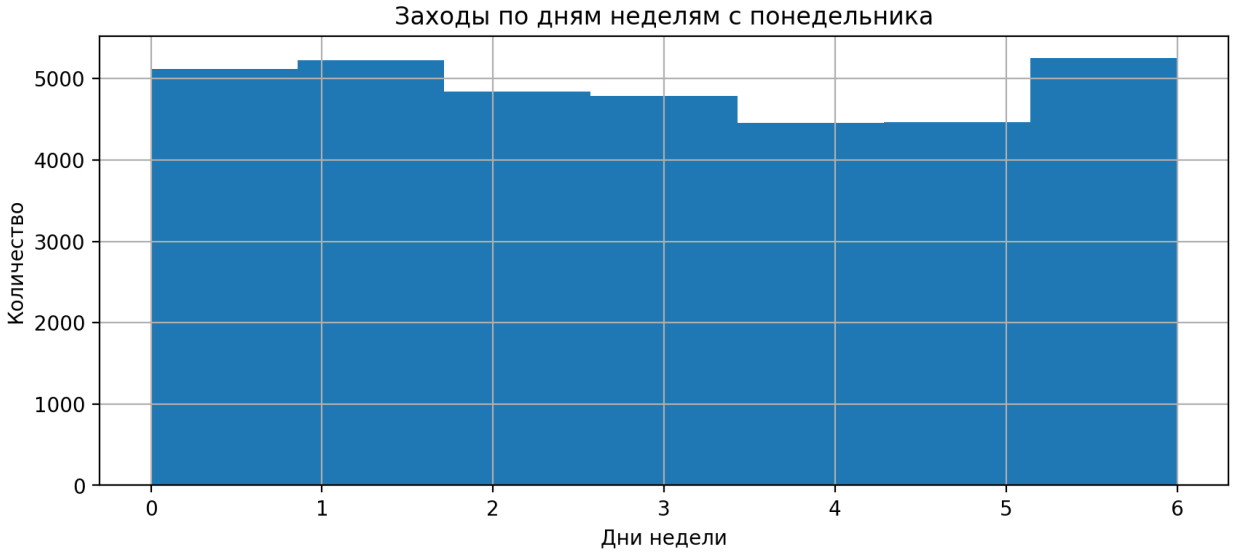
In [18]:
```
print('Общая конверсия пользователей, дошедших к просмотру контактов {0:.2%}'.format(\
        dataset[dataset['event_name'] == 'contacts_show']['user_id'].unique().shape[0] / dataset['user_id'].unique
```

```
Общая конверсия пользователей, дошедших к просмотру контактов 27.36%
```

27% - это весьма неплохой показатель. Кажется, "Ненужные вещи" пользователям нужны.

Активность по дням недели

In [19]:
```
plt.figure(figsize=(10,4),dpi=200)
dataset['event_time'].dt.weekday.hist(bins=7)
plt.title('Заходы по дням неделям с понедельника')
plt.xlabel("Дни недели")
plt.ylabel("Количество");
```



Было бы логично, если бы заходили в выходные и четверг, однако самый популятрный день все же вс, а 2 непопулярных - пт и сб. Видимо, люди откдыхают и не особо заботятся о вещах.

Относительная частота событий в разрезе двух групп

In [20]:
```
def make_table_for_pie_chart(df, contacts_show = True):
    dataset_pie = df.groupby('event_name')['event_name'].count().sort_values(ascending=False)
    if contacts_show:
        dataset_true_pie = pd.Series([dataset_pie['contacts_show']], index=["contacts_show"])
        dataset_true_pie = dataset_true_pie.append(dataset_pie.drop('contacts_show').head())
    else:
        dataset_true_pie =  pd.Series([],dtype=pd.StringDtype())
        dataset_true_pie = dataset_true_pie.append(dataset_pie.head())
    dataset_true_pie = (dataset_true_pie.append(pd.Series([dataset_pie[~dataset_pie.isin(dataset_true_pie)].sum()]
                                                          index=["other"])))
    return dataset_true_pie
```

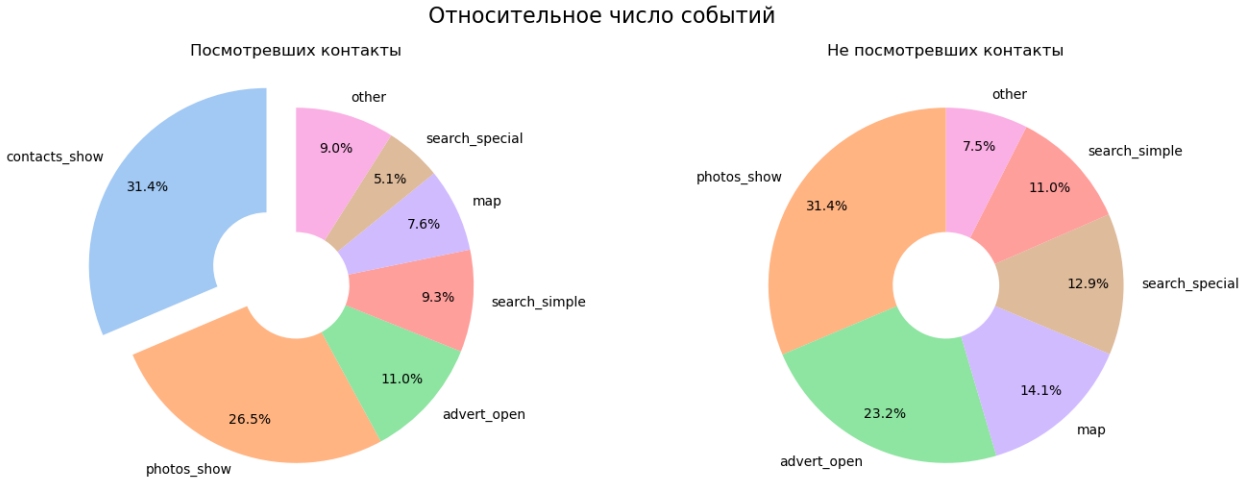
In [21]:
```
users_watched_contacts = dataset[dataset['event_name'] == 'contacts_show']
```

In [22]:
```
users_NOTwatched_contacts = make_table_for_pie_chart(dataset[~dataset['user_id'].isin(users_watched_contacts['user_
                                                     contacts_show=False)
```

In [23]:
```
users_watched_contacts = make_table_for_pie_chart(dataset[dataset['user_id'].isin(users_watched_contacts['user_id'
```

In [24]:
```
fig,axes = plt.subplots(nrows=1,ncols=2,figsize=(16,6))

explode = tuple([0.2] + [0]*(len(users_watched_contacts)-1))
colors = sns.color_palette('pastel')[0:(len(users_watched_contacts)+1)]
labels = list(users_watched_contacts.index)
axes[0].set_title('Посмотревших контакты')
axes[0].pie(users_watched_contacts, labels=labels, \
                autopct='%1.1f%%', startangle=90, colors = colors,wedgeprops=dict(width=0.7), explode= explode,\
                     pctdistance=0.8)
# каждому ивенту поставим в соответствие цвет из предыдущей палитры
tipe = {name:color for name,color in zip(users_watched_contacts.index, colors)}
explode = tuple([0]*(len(users_NOTwatched_contacts)))
colors.clear()
# обозначаем цвета
for name in users_NOTwatched_contacts.index:
    colors.append(tipe[name])
labels = list(users_NOTwatched_contacts.index)
axes[1].set_title('Не посмотревших контакты')
axes[1].pie(users_NOTwatched_contacts, labels=labels, \
                autopct='%1.1f%%', startangle=90, colors = colors,wedgeprops=dict(width=0.7), explode= explode,\
                     pctdistance=0.8)
fig.suptitle('Относительное число событий',fontsize=16);
```



Выделили 2 группы пользователей: достигших ключевого события _contacts*show* и тех, кто не достиг такового. В целом, за исключением самого ключевого события, больших различий нет: сначала просмотр фото, затем - открытие объявлений. Это вполне логично, посмольку именно так обычно происходит просмотр "ненужной вещи". Однако, есть небольшое различие есть: пользователи.ю которые смотрят контакты, предпочитают простой поиск, зато те, что не доходят целевого события пользуются поиском по карте и специальным поиском. Не исключено, что они связаны: например, в одним из изначальных search_2..7 может быть фильтр, располагающий объявления от наиболее близкого к наиболее удаленному по карте. Возможно, разработчикам проще всего было посчитать удаленность по радиусу, а не по времени езды до пункта? В любом случае, стоит присмотреться к карте и выяснить почему пользователи не доходят до целевого события.

Retantional rate этого месяца

In [25]:
```
# функция для создания пользовательских профилей

def get_profiles(sessions):
    sessions['date'] = pd.to_datetime(dataset['date'])
    # находим параметры первых посещений
    profiles = (
        sessions.sort_values(by=['user_id', 'date'])
        .groupby('user_id')
        .agg(
            {
                'date': 'first'
            }
        )
        .rename(columns={'date': 'first_ts'})
        .reset_index()
    )

    # для когортного анализа определяем дату первого посещения
    # и первый день месяца, в который это посещение произошло
    profiles['dt'] = pd.to_datetime(profiles['first_ts'].dt.date)
    profiles['month'] = profiles['first_ts'].astype('datetime64[M]')


    # считаем количество уникальных пользователей
    # с датой привлечения
    new_users = (
        profiles.groupby(['dt'])
        .agg({'user_id': 'nunique'})
        .rename(columns={'user_id': 'unique_users'})
        .reset_index()
    )
    new_users['dt'] = pd.to_datetime(new_users['dt'])

    return profiles
```

In [26]:
```
profiles = get_profiles(dataset)
```

In [27]:
```
def get_retention(
    profiles, sessions, observation_date, horizon_days, 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(
    dataset[['user_id', 'date']], on='user_id', how='left'
    )
    result_raw['lifetime'] = (
    result_raw['date'] - result_raw['first_ts']
    ).dt.days

    """
    Применим метод `pivot_table()`. Чтобы сделать даты первого посещения заголовками строк, лайфтаймы — заголовкам
    а количество уникальных пользователей — значениями, передадим параметру `index` значения столбца `dt`,
    параметру `columns` — значения `lifetime`, параметру `values` — значения `user_id`,
    а параметру `aggfunc` — функцию для подсчёта уникальных значений `nunique`.
    """
    result_grouped = result_raw.pivot_table(
        index=['dt'], columns='lifetime', values='user_id', aggfunc='nunique'
    )
    cohort_sizes = (
        result_raw.groupby('dt')
        .agg({'user_id': 'nunique'})
        .rename(columns={'user_id': 'cohort_size'})
    )
    result_grouped = cohort_sizes.merge(
        result_grouped, on='dt', how='left'
    ).fillna(0)
    result_grouped = result_grouped.div(result_grouped['cohort_size'], axis=0)

    # исключаем все лайфтаймы, превышающие горизонт анализа
    result_grouped = result_grouped[
        ['cohort_size'] + list(range(horizon_days))
    ]

    # восстанавливаем столбец с размерами когорт
    result_grouped['cohort_size'] = cohort_sizes

    # возвращаем таблицу удержания и сырые данные
    # сырые данные пригодятся, если нужно будет отыскать ошибку в расчётах
    return result_raw, result_grouped
```

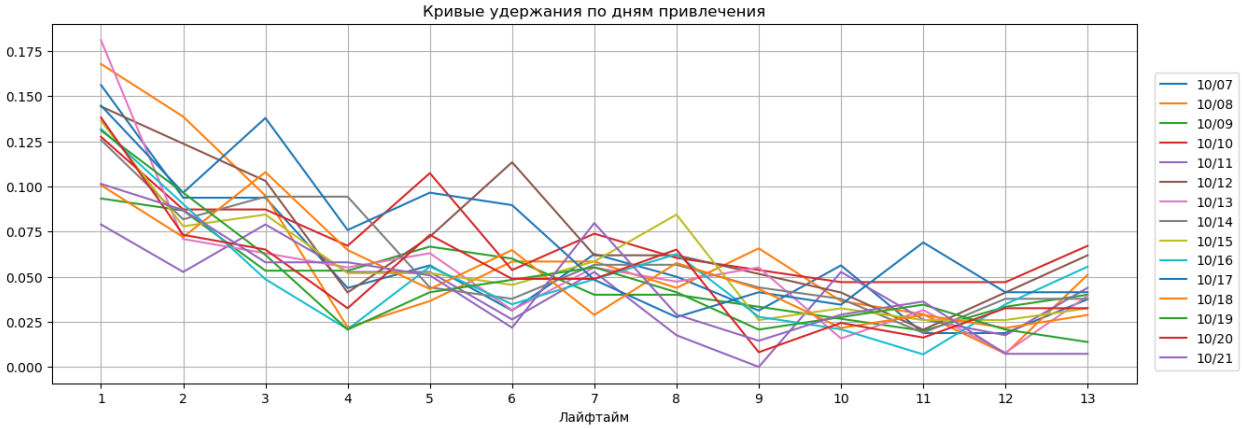
Возьмем за горизонт анализа 2 недели, получится 15 полноценных когорт

In [28]:
```
retention_raw, retention = get_retention(
    profiles, dataset,  dataset['event_time'].max().date(), 14)
retention.index = retention.index.strftime('%m/%d')
```

In [29]:
```
plt.figure(figsize=(10,4),dpi=200)
sns.heatmap(retention.drop(['cohort_size', 0],axis=1),
            annot=True,  # включаем подписи
            fmt = '.2%')  # отображаем значения в виде процентов
plt.title('Тепловая карта коэффициента удержания')
plt.xlabel("День жизни")
plt.ylabel("Дата начала когорты");
```

Показатели когорт не столь радужны: удержание седьмого дня составляет менее 8%, а местами к концу(на 14 день) падает ниже 1%. Самая «мрачная» когорта на графике — пользователи, привлечённые 28 октября. Также заметно, что во вт и вс приходит больше пользователей, чем в другие дни. Однако, тенденции в их удержании не видно. Есть даже 0%(что приходится как раз на сб)

In [30]:
```
retention.drop(['cohort_size', 0],axis=1).T.plot(
    grid=True,  # добавляем сетку
    xticks=list(retention.drop(['cohort_size', 0],axis=1).columns.values),  # отметки на оси X — названия колонок
    figsize=(15, 5),  # размер графика
)
plt.legend(loc='right', bbox_to_anchor=(0.6, 0.2, 0.5, 0.5))
plt.xlabel('Лайфтайм')  # название оси X
plt.title('Кривые удержания по дням привлечения')  # название графика
plt.show()
```



Как и ожидалось, удержание падает. Показатели в целом довольно низкие.

DAU

In [31]:
```
temp = dataset.pivot_table(index='date', values='user_id', aggfunc='nunique')
name_x = 'Дни с ' + str(temp.index[0].strftime('%d %b %Y')) + ' по ' + str(temp.index[-1].strftime('%d %b %Y'))
temp.index = temp.index.strftime('%d')
```

In [32]:
```
plt.figure(figsize=(10,4),dpi=200)
plt.bar(list(temp.index), temp['user_id'])
plt.title('Количество уникальных пользователей в день')
plt.xlabel(name_x)
plt.ylabel("Число пользователей");
```

Что ж, Хотябы общее количество уникальных пользователей в день растет. Это хороший показатель для приложения

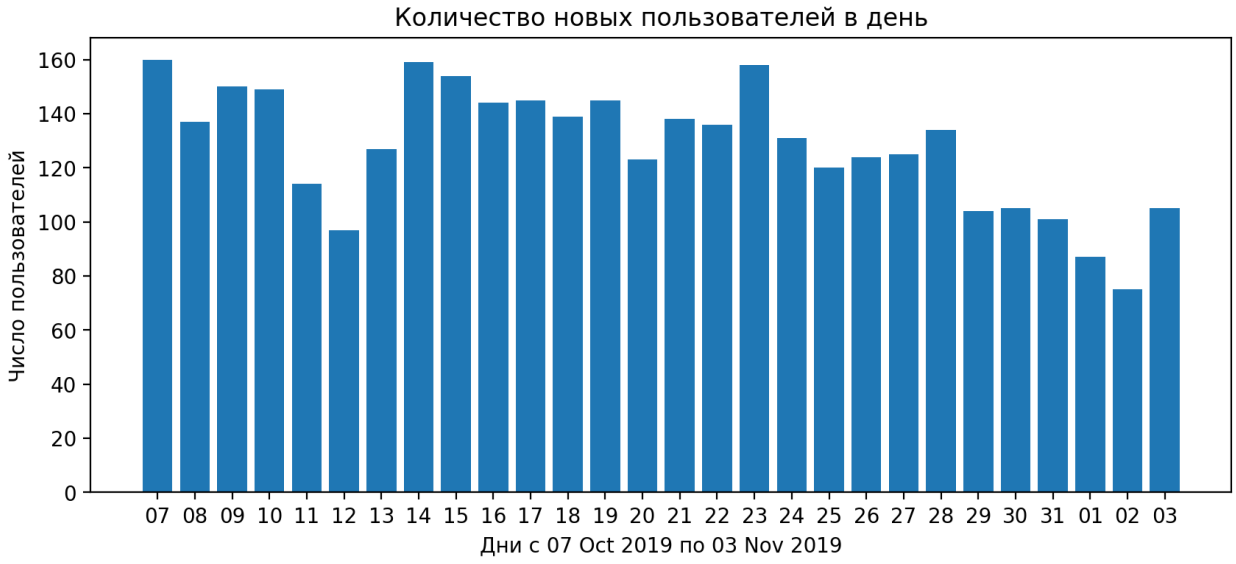
Количество новых пользователей в день

In [33]:
```
was_here = []
def in_previous_days(users):
    count_new_users = 0
```

```
    for user in users:
        if user not in was_here:
            was_here.append(user)
            count_new_users += 1
    return count_new_users
```

In [34]:
```
temp = dataset.pivot_table(index='date', values='user_id', aggfunc=[lambda x: in_previous_days(x)])
name_x = 'Дни с ' + str(temp.index[0].strftime('%d %b %Y')) + ' по ' + str(temp.index[-1].strftime('%d %b %Y'))
temp.index = temp.index.strftime('%d')
del was_here
```

In [35]:
```
plt.figure(figsize=(10,4),dpi=200)
plt.bar(list(temp.index), temp[('<lambda>', 'user_id')])
plt.title('Количество новых пользователей в день')
plt.xlabel(name_x)
plt.ylabel("Число пользователей");
```



Интересно, что больше новых пользователей приходит в начале недели, но чем ближе к концу - тем, как правило, меньше. Это расходится с тем, что больше всего заходов в выходные. Также возможна тенденция к общему уменьшению количества новых пользователей. Это странно, поскольку хорошее приложение со временем обычно привлекает все больше и больше новых. Возможно, для "Ненужные вещи" стоит провести маркетинговую кампанию по привлечению новых. Уже исзвестно, что большинство пришли из Яндекса, поэтому ориентироваться стоит на данный поисковик.

Тайм-аут

Сколько пользователи проводят времени на сайте: https://www.statista.com/statistics/1201901/most-visited-websites-worldwide-time-visit/

Среднее время на сайте Амазон составляло около 13мин. Проверим насколько близко значение в данном датасете

In [36]:
```
new_dataset = dataset.copy().sort_values(['user_id', 'event_time'])
new_dataset['session'] = new_dataset.groupby('user_id')['event_time'].diff()
# проверим подсчет на каком-л одном пользователе
new_dataset[new_dataset['user_id']== '0001b1d5-b74a-4cbf-aeb0-7df5947bf349'].sort_values(by='event_time')
```

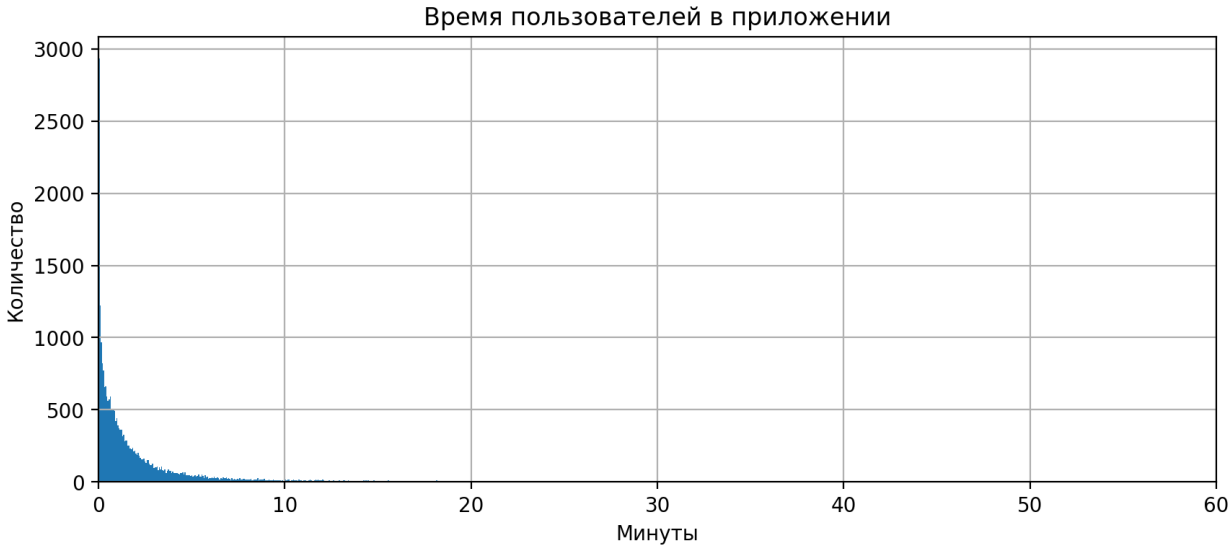
Out[36]:

	event_time	event_name	user_id	date	session
6541	2019-10-09 18:33:55.577963	map	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	2019-10-09	NaT
6546	2019-10-09 18:35:28.260975	map	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	2019-10-09	0 days 00:01:32.683012
36419	2019-10-21 19:53:38.767230	map	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	2019-10-21	12 days 01:18:10.506255
36430	2019-10-21 19:56:49.417415	map	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	2019-10-21	0 days 00:03:10.650185
37556	2019-10-22 11:18:14.635436	map	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	2019-10-22	0 days 15:21:25.218021
37581	2019-10-22 11:25:33.508919	map	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	2019-10-22	0 days 00:07:18.873483

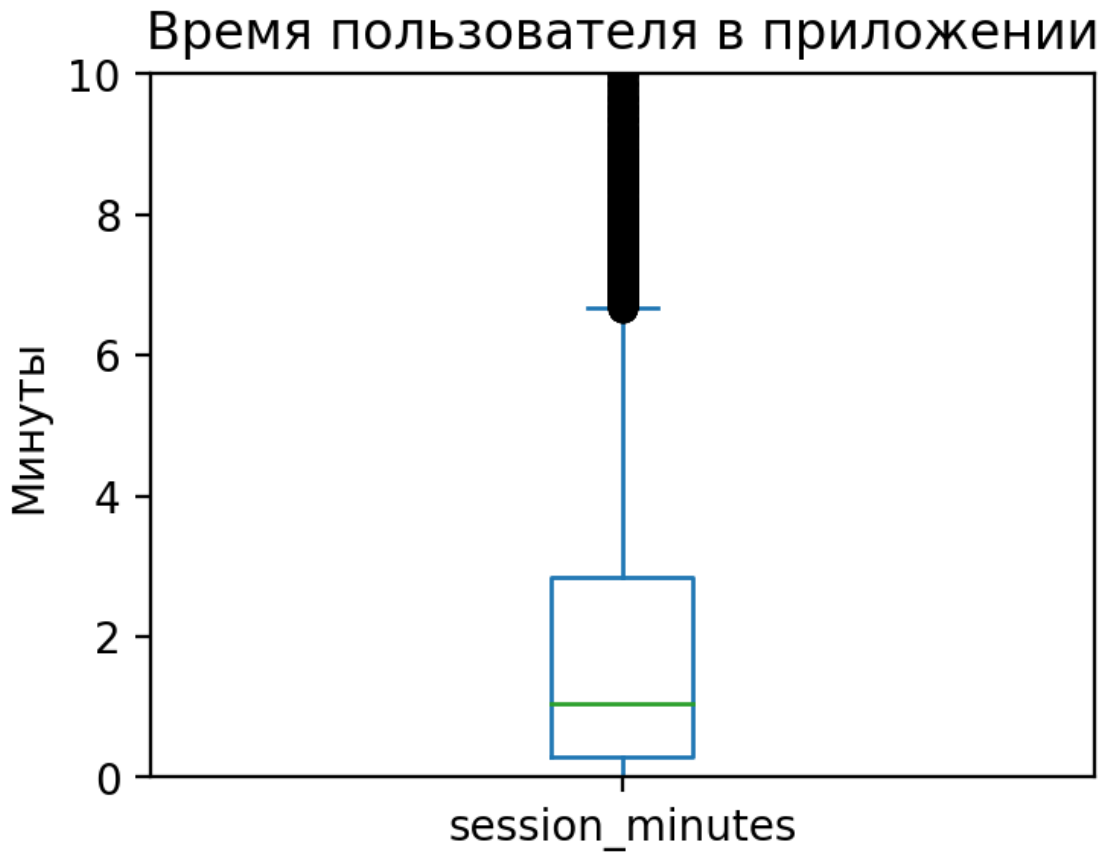
In [37]:
```
plt.figure(figsize=(10,4),dpi=200)
new_dataset['session_minutes'] = new_dataset['session'].dt.total_seconds() / 60
print('Максимально высчитаное время сессии',round(new_dataset['session_minutes'].max()/ 60),'часов')
temp = new_dataset[(new_dataset['session_minutes'] < 60) & (new_dataset['session_minutes'] != 0)]
temp['session_minutes'].hist(bins = 1000)
#посмотрим на час
plt.xlim(0,60)
```

```
plt.title('Время пользователей в приложении')
plt.xlabel("Минуты")
plt.ylabel("Количество");
```

```
Максимально высчитаное время сессии 638 часов
```



In [38]:
```
plt.figure(figsize=(4,3),dpi=200)
temp['session_minutes'].plot.box()
plt.ylim(0,10)
plt.title('Время пользователя в приложении')
plt.xlabel(" ")
plt.ylabel("Минуты");
```

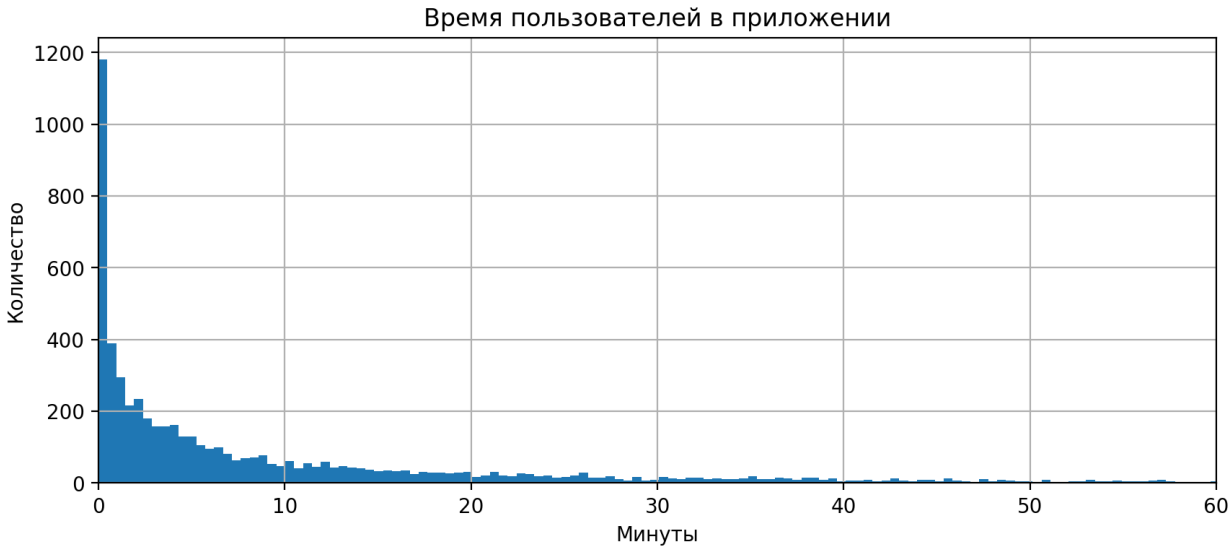


In [39]:
```
median = temp['session_minutes'].median()
print(f'Медиана = {median:.1f} мин.')
```

```
Медиана = 1.0 мин.
```

In [40]:
```
g = (new_dataset['session'] > pd.Timedelta('30Min')).cumsum()
new_dataset['session_id'] = new_dataset.groupby(['user_id', g], sort=False).ngroup() + 1
temp = pd.pivot_table(new_dataset,
                      index=["session_id"],
```

```
                values=["session"],
                aggfunc=['sum','max'])
temp.loc[temp[('max', 'session')] != temp[('sum', 'session')], 'real_session'] = (temp[('sum', 'session')] -
                                                                                   temp[('max', 'session')])
plt.figure(figsize=(10,4),dpi=200)
temp.loc[temp['real_session'].isna(), 'real_session'] = temp[('sum', 'session')]
temp['session_minutes'] = temp['real_session'].dt.total_seconds() / 60
temp = temp[(temp['session_minutes'] < 480) & (temp['session_minutes'] != 0)]
temp['session_minutes'].hist(bins = 1000)
#посмотрим на час
plt.xlim(0,60)
plt.title('Время пользователей в приложении')
plt.xlabel("Минуты")
plt.ylabel("Количество");
```



Видно, что очень много пользователей быстро заходят и уходят. График имеет убывающий гиперболически вид, то есть чем больше количество минут в приложении, тем меньше таких пользователей. Значение тайм-аута, подобранное гугл аналитиками, тоже нам подходит, поэтому диаграмму строим со срезом в 30мин

Диаграмма Сэнкей

In [41]:
```
#пересчитаем session_id
temp = (new_dataset['session'] > pd.Timedelta('30Min')).cumsum()
new_dataset['session_id'] = new_dataset.groupby(['user_id', temp], sort=False).ngroup() + 1
new_dataset = new_dataset[~new_dataset.duplicated(subset=["session_id", "event_name"])]
# некоторые столбцы просто не пригодятся - незачем таскать с собой
new_dataset = new_dataset.drop([ 'date'], axis=1)
```

In [42]:
```
new_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11934 entries, 6541 to 72552
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype         
---  ------           --------------  -----         
 0   event_time       11934 non-null  datetime64[ns]
 1   event_name       11934 non-null  string        
 2   user_id          11934 non-null  string        
 3   session          8348 non-null   timedelta64[ns]
 4   session_minutes  8348 non-null   float64       
 5   session_id       11934 non-null  int64         
dtypes: datetime64[ns](1), float64(1), int64(1), string(2), timedelta64[ns](1)
memory usage: 652.6 KB
```

In [43]:
```
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'] = sorted_df['event_name'].astype(str)
    # добавляем целевые узлы
    sorted_df['target'] = sorted_df.groupby('session_id')['source'].shift(-1)

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

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

Out[43]:

	event_time	user_id	session	session_minutes	session_id	step	source	target
6541	2019-10-09 18:33:55.577963	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	NaT	NaN	1	1	map	NaN
36419	2019-10-21 19:53:38.767230	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	12 days 01:18:10.506255	17358.175104	2	1	map	NaN
37556	2019-10-22 11:18:14.635436	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	0 days 15:21:25.218021	921.420300	3	1	map	NaN
31632	2019-10-19 21:34:33.849769	00157779-810c-4498-9e05-a1e9e3cedf93	NaT	NaN	4	1	search_simple	photos_show
31655	2019-10-19 21:40:38.990477	00157779-810c-4498-9e05-a1e9e3cedf93	0 days 00:03:54.645786	3.910763	4	2	photos_show	NaN

In [44]:
```
df_comp = table.copy() #table[table['step'] <= 6].copy().reset_index(drop=True)
```

In [45]:
```
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)
```

In [46]:
```
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 [47]:
```python
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 = {}

    # 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)'



    return colors_dict


# генерю цвета
colors_dict = colors_for_sources(mode='random')
```

In [48]:
```python
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
```

In [49]:
```python
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'] == ta
                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()
```

```
Шаг:   0%|          | 0/6 [00:00<?, ?it/s]
  0%|          | 0/8 [00:00<?, ?it/s]
 38%|███▊      | 3/8 [00:00<00:00, 28.00it/s]
100%|██████████| 8/8 [00:00<00:00, 27.66it/s]
Шаг:  17%|█▋        | 1/6 [00:00<00:01,  3.42it/s]
  0%|          | 0/9 [00:00<?, ?it/s]
 33%|███▎      | 3/9 [00:00<00:00, 28.95it/s]
 67%|██████▋   | 6/9 [00:00<00:00, 28.22it/s]
100%|██████████| 9/9 [00:00<00:00, 28.27it/s]
Шаг:  33%|███▎      | 2/6 [00:00<00:01,  3.23it/s]
  0%|          | 0/9 [00:00<?, ?it/s]
 33%|███▎      | 3/9 [00:00<00:00, 27.27it/s]
 67%|██████▋   | 6/9 [00:00<00:00, 27.21it/s]
100%|██████████| 9/9 [00:00<00:00, 27.30it/s]
Шаг:  50%|█████     | 3/6 [00:00<00:00,  3.12it/s]
  0%|          | 0/9 [00:00<?, ?it/s]
100%|██████████| 9/9 [00:00<00:00, 41.25it/s]
Шаг:  67%|██████▋   | 4/6 [00:01<00:00,  3.55it/s]
100%|██████████| 6/6 [00:00<00:00, 97.10it/s]
Шаг: 100%|██████████| 6/6 [00:01<00:00,  4.85it/s]
```

In [50]:
```
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.
```

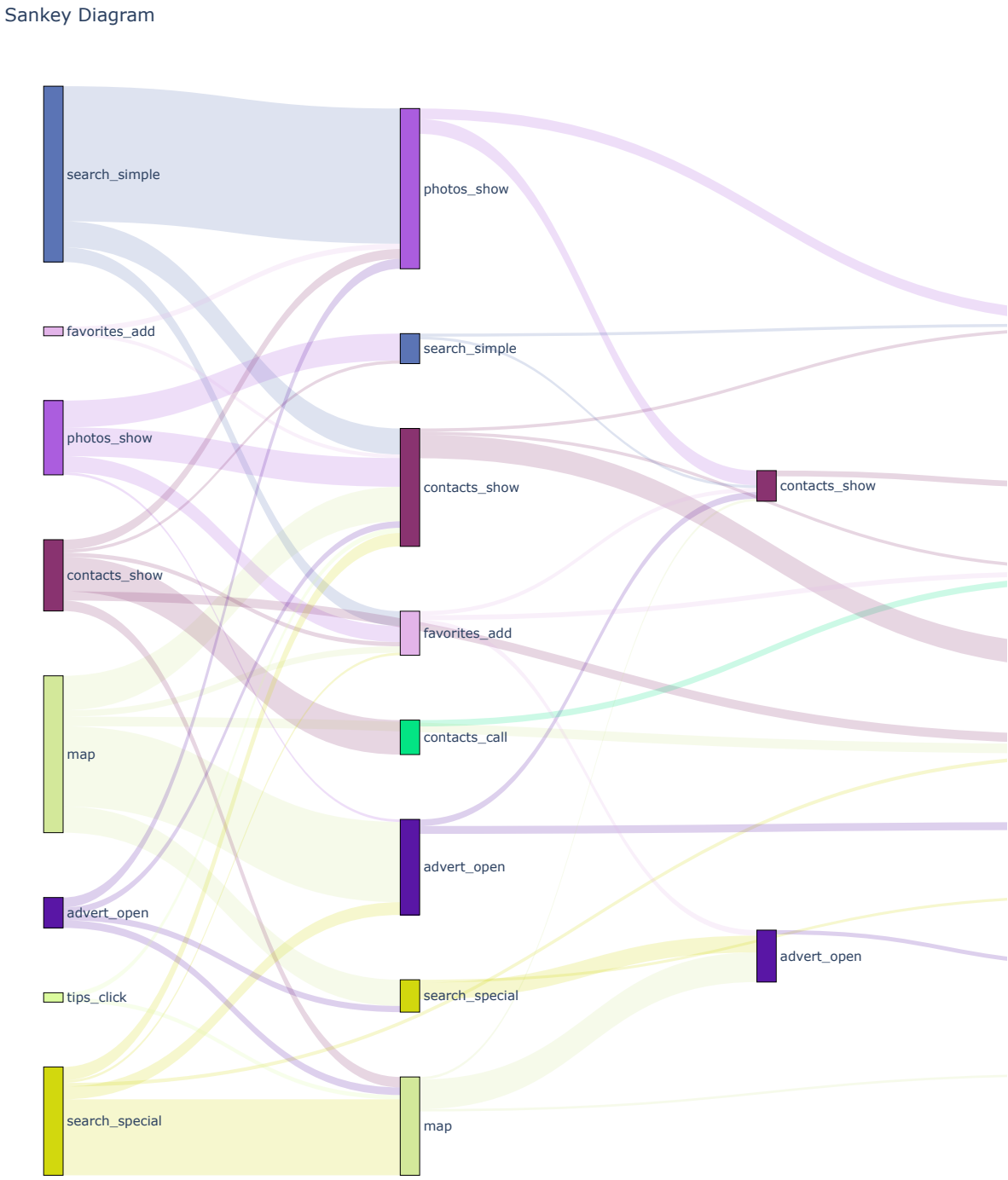
```
    Returns:
        plotly.graph_objs._figure.Figure: объект изображения.
    """

    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=1000, height=1020)

    # возвращаем объект диаграммы
    return fig


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

In [51]:
```
senkey_diagram.show()
```



Много пользователей в принципе идет из простого поиска и с карт. Также в карты заходят из специального поиска. Интересно, что отображается не более 4 этапов, а очень многие достигают ключевого на втором. Выберем для воронок:

1. search_simple -> photos_show -> contacts_show;
2. photos_show -> contacts_show;
3. map -> contacts_show;
4. search_special -> map -> advert_open -> contacts_show

Воронки

Задание функций для построения

In [52]:
```
def filtred_data(df, *args):
    new_df = df[df['event_name'] == args[0]]
    users_id = new_df['user_id'].unique()
```

```
    temp = pd.DataFrame({'unique_users':len(new_df['user_id'].unique()), \
                         ('general_users',        ''):new_df.shape[0]
                         }, index=[args[0]])
    for i in range(1, len(args)):
        new_df = df[(df['user_id'].isin(users_id)) & (df['event_name'] == args[i])]
        users_id = new_df['user_id'].unique()
        new_row = pd.DataFrame({'unique_users':len(new_df['user_id'].unique()), \
                         ('general_users',        ''):new_df.shape[0]
                         }, index=[args[i]])
        new_row['general_users'] = new_df['user_id'].nunique()
        temp = pd.concat([temp,new_row.loc[:]])
    return temp

table_temp = filtred_data(dataset, 'search_simple', 'photos_show', 'contacts_show')
table_temp
```

Out[52]:

	unique_users	(general_users,)	general_users
search_simple	787	3506	NaN
photos_show	643	5798	643.0
contacts_show	191	577	191.0

In [53]:

```
def build_funnel(df, n=''):
    # Для дальнейшей визулизации создадим список с именами воронки
    events = list(df.index)
    # а также со значениями уникальных пользователей
    x_val = list(df['unique_users'])
    # построение воронки
    fig = go.Figure(go.Funnel(
    y = events[:4],
    x = x_val[:4],
    textposition = "inside",
    textinfo = "value+percent initial+percent previous",
    opacity = 0.65, marker = {"color": ["deepskyblue", "lightsalmon", "tan", "teal", "silver"],
    "line": {"width": [4, 2, 2, 3, 1, 1], "color": ["wheat", "wheat", "blue", "wheat", "wheat"]}},
    connector = {"line": {"color": "royalblue", "dash": "dot", "width": 3}})
    )
    fig.update_layout(title=dict(text='Воронка событий ' + str(n), x=0.5, font_family="Times New Roman"),
                      yaxis_title='Этапы',
                      yaxis = dict(
                        tickfont = dict(family = 'Times New Roman'), # оформление самих значений на оси Y
                        title_font = dict(family = 'Times New Roman') # оформление подписи Y
                      )
    )
    # для подсчета процента относительно начального количества пользователей можно заменить percent previous на pe
    fig.show()
```

search_simple -> photos_show -> contacts_show

In [54]:

```
build_funnel(table_temp, 1)
```

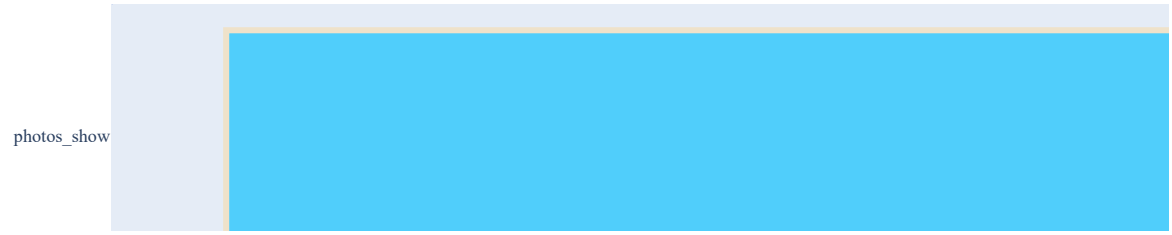
Вор

Здесь хороший переход от простого поиска к показу фото, но куда меньше людей смотрят потом контакты: относительно предыдущего шага 30% - чуть более, чем в 3 раза уменьшилось количество пользователей с предыдушего шага.

photos_show -> contacts_show

In [55]:
```
table_temp = filtred_data(dataset,'photos_show', 'contacts_show')
build_funnel(table_temp, 2)
```

Вор

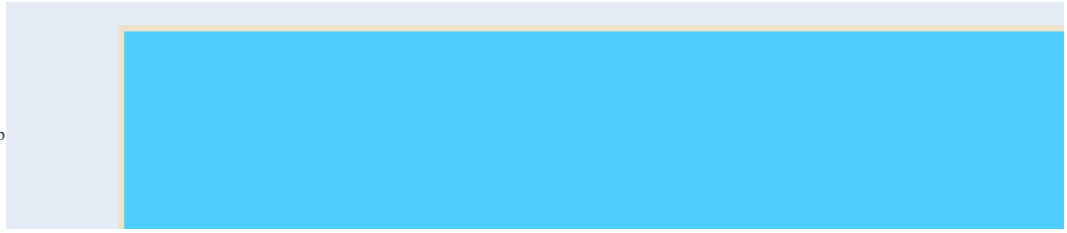


С показа фото сразу к контактам переходит 31%

map -> contacts_show

In [56]:
```
table_temp = filtred_data(dataset,'map', 'contacts_show')
build_funnel(table_temp, 3)
```

Вор

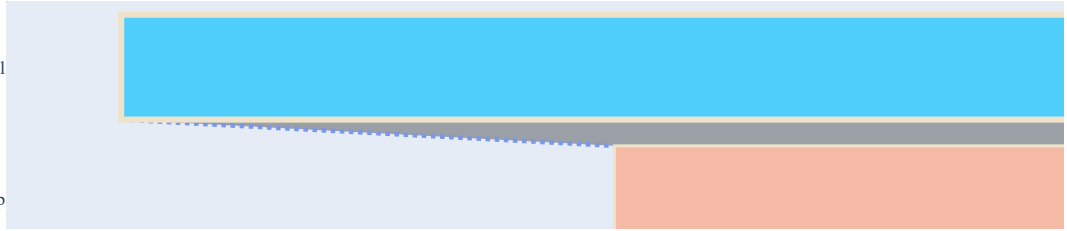


Здесь Чуть меньше 20%

search_special -> map -> advert_open -> contacts_show

In [57]:
```
table_temp = filtred_data(dataset,'search_special', 'map', 'advert_open', 'contacts_show')
build_funnel(table_temp, 4)
```

Вор



Данная воронка была выбранна из-за четырех этапов, и наглядно демонстрирует, что пользователи, которые проходят больше 2 этапов, сильно теряют в конверссии со ступени на ступень. К концу осталось не более 6% от первоначального числа.

Краткий вывод

- Общая конверсия в период в 7 октября по 3 ноября 2019 составила 27.36%
- самый популярный день - вс, а 2 непопулярных - пт и сб
- разделили на 2 группы пользователей: достигших _contacts*show* и нет. В обоих группах наиболее популятны сначала просмотр фото, затем - открытие объявлений. Однако, есть небольшое различие есть: пользователи, которые смотрят контакты, предпочитают простой поиск, зато те, что не доходят целевого события пользуются поиском по карте и специальным
- коэффициент удержания этого месяца равен 74.42%
- общее количество уникальных пользователей в день растет
- количество новых пользователей в день падает
- медианное время в приложении:1мин; не являющееся выбросом взято 30мин
- по результатам диаграммы Сэнкей построены 5 воронок:
  1. search_simple -> photos_show -> contacts_show;
  2. photos_show -> contacts_show;
  3. map -> contacts_show;
  4. photos_show -> contacts_show
  5. search_special -> map -> advert_open -> contacts_show

  Конверсия лучше всего у двуэтапных воронок. Пользователи не ходят больше, чем по 4 этапам, неспотря на возможные 15шт в датасете. Причем, чем больше воронка, тем хуже абсолютная конверсия. Лучшиq результат у photos_show -> contacts_show.

Проверка гипотез

Гипотеза №1

Пользователи группы А совершают действия _tips*show* и _tips*click* , группы В - только _tips*show* . Необходимо проверить гипотезу: конверсия в просмотры контактов различается у этих двух групп. Сформулируем гипотезы:

Нулевая: нет различий в конверсии между группами А и В.
Альтернативная: различия между группами есть.

In [58]:
```
# формируем данные по группе А - пользователи совершили действия:
# tips_show, tips_click
# множества быстро работают, поэтому будем пользоваться ими

def unique_users(df, event):
    return set(df[df['event_name'] == event]['user_id'])

# пользователи, которые совершили действие tips_show
users_tips_show = unique_users(dataset_with_tips_show, 'tips_show')
# пользователи, которые совершили действие tips_click
users_tips_click = unique_users(dataset_with_tips_show, 'tips_click')
# пользователи, которые совершили действие contacts_show
users_contacts_show = unique_users(dataset_with_tips_show, 'contacts_show')

# поскольку нам нужно только количество, далее будем пользоваться только длиной
# пользователи, которые совершили и contacts_show, и tips_show, и tips_click - группа А изначальная
usersA1_contacts_show = len(users_tips_show.intersection(users_tips_click,users_contacts_show))
usersA1 = len(users_tips_show.intersection(users_tips_click))

#итак, нам понадобится конверсия группы А: пользователи, которые совершили просто  и tips_show, и tips_click
# к пользователям, которые совершили и contacts_show, и tips_show, и tips_click
# т.о. пропорция успехов в первой группе:
p1A1 = usersA1_contacts_show / usersA1
print('Пропорция успехов в первой группе:{0:.2%}'.format(p1A1))

# конверсия группы В: пользователи, которые совершили только tips_show, без tips_click
# к пользователям, которые совершили и contacts_show, tips_click
usersB1 = users_tips_show.difference(users_tips_click)
# пользователи, которые совершили и contacts_show, и tips_show - группа В
usersB1_contacts_show = len(usersB1.intersection(users_contacts_show))
usersB1 = len(usersB1)
# пропорция успехов во второй группе:
p2B1 = usersB1_contacts_show / usersB1
print('Пропорция успехов во второй группе:{0:.2%}'.format(p2B1))

# пропорция успехов в комбинированном датасете:
p_combined = (usersA1_contacts_show + usersB1_contacts_show) / (usersA1 + usersB1)
```

```
Пропорция успехов в первой группе:30.64%
Пропорция успехов во второй группе:16.97%
```

In [59]:
```
alpha = .05 # критический уровень статистической значимости

# разница пропорций в датасетах
```

```
difference = p1A1 - p2B1

# считаем статистику в ст.отклонениях стандартного нормального распределения
z_value = difference / mth.sqrt(p_combined * (1 - p_combined) * (1/usersA1 + 1/usersB1))

# 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: ', 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')
```

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

Такая разница в пропорциях при указанных размерах выборок достаточна, чтобы говорить о статистически значимом различии. Придется отвергнуть нулевую гипотезу в пользу альтернативной.

Гипотеза №2

Пользователи группы А совершают действия _photos*show*, группы В - только _tips*show* . Необходимо проверить гипотезу: конверсия в просмотры контактов различается у этих двух групп. Сформулируем гипотезы:

Нулевая: нет различий в конверсии между группами А и В.
Альтернативная: различия между группами есть.

In [60]:
```
# пользователи, которые совершили действие photos_show
users_photos_show = unique_users(dataset_with_tips_show, 'photos_show')
# пользователи, которые не смотрели фото
temp = dataset_with_tips_show[~dataset_with_tips_show['user_id'].isin(users_photos_show)]
users_without_photos_show = set(temp['user_id'])
# пользователи, которые совершили действие contacts_show
users_contacts_show = unique_users(dataset_with_tips_show, 'contacts_show')

# поскольку нам нужно только количество, далее будем пользоваться только длиной
# пользователи, которые совершили и contacts_show, и photos_show - группа А
usersA2_contacts_show = len(users_photos_show.intersection(users_contacts_show))
usersA2 = len(users_photos_show)

#итак, нам понадобится конверсия группы А: пользователи, которые совершили просто photos_show
# к пользователям, которые совершили и contacts_show, и photos_show
# т.о. пропорция успехов в первой группе:
p1A2 = usersA2_contacts_show / usersA2
print('Пропорция успехов в первой группе:{0:.2%}'.format(p1A2))

# конверсия группы В: пользователи, которые совершили не смотрели фото
# к пользователям, которые совершили и видели контакты, совершив целевое действие, и не видели фото
# пользователи, которые совершили и contacts_show, и tips_show - группа В
usersB2_contacts_show = len(users_without_photos_show.intersection(users_contacts_show))
usersB2 = len(users_without_photos_show)
# пропорция успехов во второй группе:
p2B2 = usersB2_contacts_show / usersB2
print('Пропорция успехов во второй группе:{0:.2%}'.format(p2B2))

# пропорция успехов в комбинированном датасете:
p_combined = (usersA2_contacts_show + usersB2_contacts_show) / (usersA2 + usersB2)
```

```
Пропорция успехов в первой группе:30.96%
Пропорция успехов во второй группе:20.08%
```

In [61]:
```
alpha = .01 # критический уровень статистической значимости

# разница пропорций в датасетах
difference = p1A2 - p2B2

# считаем статистику в ст.отклонениях стандартного нормального распределения
z_value = difference / mth.sqrt(p_combined * (1 - p_combined) * (1/usersA2 + 1/usersB2))

# 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: ', 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')
```

```
p-значение:  1.3278267374516872e-13
Отвергаем нулевую гипотезу: между долями есть значимая разница
```

Казалось бы, между 31% и 20% не такая большая разница, однако она статически значимая. Пользователи, которые смотрят фото, чаще доходят до целевого действия.

Краткий вывод

Различие в конверсии пользователей, которые совершают действия _tips*show* и _tips*click* и пользователей, совершающих только _tips*show* является статичтически значимой, причем в первой группе выше. Также различие в конверсии пользователей, просматривающих фото и не смотревших, тоже статичтически значима. Дополнительно известно, что в первой группе 31%, а во второй 20%.

Вывод

По общему обзору :

- переименовали столбцы
- привели столбцы к типу данных string и к datetime
- устранили неявные дубликаты в виде _contacts*show* и _show*contacts*

Рекомендации:

- логировать заглавную страницу. Плюсы: счет пользователей бы стать более точной метрикой количества заходов в приложение, а также стартовой позицией для большинства воронок, стало бы проще отслеживать как часто пользователи заходят в приложение в разные дни месяца и/или недели
- уделить внимание системе рекомендаций
- рассмотреть возможность выйти на рынок мобильных приложений: PlayMarket/AppStore/AppGalary
- привести к общему виду базы данных, чтобы избежать дубликатов _contacts*show* и _show*contacts*
- рассмотреть возможность хранения части данных в string, поскольку это более экономно с т.зр. памяти, чем в object

По предобработке:

- глава анализа данных проходила без _tips*show*, поскольку действие является автоматическим и не отображает поведелния пользователей, глава проверки гипотез - с _tips*show*
- переименовали _search*1* в _search*simple*(поиск по прожатию кнопки), остальные объединили в search_special(возможно, предполагающие фильтры)

Рекомендации:

- переименовать _search*1..7* на более интуитивно понятные параметры поиска или прописать их значение в документации

По анализу данных:

- отношение уникальных пользователей, дошедщих до целевого событи - просмотр контактов - к общему числу в период в 7 октября по 3 ноября 2019 составила 27.36%
- люди чаще заходят во вторник и воскресенье, меньше всего в пятницу и субботу
- разделили пользователей на 2 группы: достигших ключевого события(группа 1) и нет(группа 2). В целом, больших различий нет. Интересно, что группа 1 предпочитает простой поиск, группа 2 пользуется поиском по карте и специальным
- разделили пользователей на когорты, посчитали retantionl rate. Показатели когорт не столь радужны: удержание седьмого дня составляет менее 8%, а местами к концу(на 14 день) падает ниже 1%
- количество новых пользователей в день имеет тенденцию уменьшаться
- медиана времени, которое пользователи проводят в приложении составляет около 5мин,
- построена диаграмма Сэнкей с тайм-аутом 50мин, на основании выделены сценарии:

    1. search_simple -> photos_show -> contacts_show
       Вывод: сценарий соотносится с результатами раздела рассчета относительной частоты и достаточно популярен. Однако за переход от 2 этапа к 3 количество пользователей уменьшется почти в 3 раза 2.photos_show -> contacts_show
       Вывод: конверсия 31%
    2. map -> contacts_show
       Вывод: конверсия 20%

3. search_special -> map -> advert_open -> contacts_show

   Вывод: Данная воронка была выбранна из-за четырех этапов, и наглядно демонстрирует, что пользователи, которые проходят больше 2 этапов, сильно теряют в конверссии со ступени на ступень. К концу осталось не более 6% от первоначального числа

Конверсия лучше всего у двуэтапных воронок. Пользователи не ходят больше, чем по 4 этапам, несмотря на возможные 15шт в датасете. Причем, чем больше воронка, тем хуже абсолютная конверсия. Лучшие результат у photos_show -> contacts_show.

Рекомендации:

- акции, скидки, дополнительные предложения по продвижению и другую подобную активную деятельность лучше планировать на вс или вт - в это время больше всего пользователей заходит в приложение
- собрать фокус-группу, выяснить почему retantionl rate такой низкий

По проверке гипотез

- различие в конверсии пользователей, которые совершают действия _tips*show* и _tips*click* и пользователей, совершающих только _tips*show* является статичтически значимой, причем в первой группе выше.
- различие в конверсии пользователей, просматривающих фото и не смотревших, тоже статичтически значима. Дополнительно известно, что в первой группе 31%, а во второй 20%.

In [ ]: