Выделение групп пользователей на основе их поведения в мобильном приложении

Содержание

Контекст исследования:

Мобильное приложение "Ненужные вещи" представляет собой площадку, на которой пользователи могут продавать и покупать ненужные бывшие в употреблении вещи. Продавцы публикуют в приложении объявления, покупатели - ищут нужный товар, связываются с продавцом (просматривают контактный номер телефона). Созвон покупателя и продавца возможен как внутри, так и вне приложения.

Монетизация в приложении основана на продвижении объявлений (платное поднятие) в поисковых запросах, что должно влиять на конверсию, вероятность продажи и "уровень удовлетворённости" продавцов и покупателей.

Цели исследования:

1. Разделить пользователей на несколько различающихся по поведению в приложении групп, на которые проджект-менеджмент может влиять для управления вовлечённостью (адаптировать приложение под целевую и смежные аудитории).
2. Получить на основе сегментации гипотезы о путях улучшения пользовательского опыта (UX) в приложении.

Задача исследования: получить ответы на следующие основные вопросы продакт-менеджера:

1. Пользователи какой группы склонны часто возвращаться в мобильное приложение (retention rate)?
2. Пользователи какой группы часто совершают целевое событие (конверсия в целевое событие)?
3. Как различается время, проводимое в приложении, для пользователей разных групп?

Заказчик исследования: продакт-менеджер, желающий влиять на вовлечённость различных пользователей в приложение.

Использование результатов исследования: полученные в исследовании результаты предполагается использовать для:

- разработки мероприятий по влиянию на группы пользователей;
- проработки внутренних метрик продуктового отдела.

Исходные данные исследования: В исследовании использованы 2 датасета:

- `mobile_dataset.csv` - содержит данные о событиях, совершенных пользователями в мобильном приложении;
- `mobile_soures.csv` - содержит информацию об источниках привлечения пользователей.

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

Замечание: в названии датасета с источниками допущена орфографическая ошибка - обратить внимание инженеров по данным.

План исследования:

1. Знакомство с данными, обзор.
2. Предварительная обработка по результатам обзора.
3. Исследовательский анализ данных (EDA).
4. Сегментация пользователей.
5. Проверка гипотез:
   - гипотеза 1: пользователи, установившие приложение по ссылке из yandex и из google демонстрируют разную конверсию в просмотры контактов;
   - гипотеза 2: среднее время сессии для пользователей, пришедших в начале недели (пн-чт) и в конце недели (пт-вс) отличается.
6. Итоговые выводы исследования и базовые рекомендации для продакт-менеджера.
7. Подготовка презентации.

Замечание: в будущем предполагается проведение таких исследований на регулярной основе.

Библиотеки, используемые в исследовании, и вспомогательные функции

In [1]:
```
# основные библиотеки DA
import pandas as pd
import numpy as np
import math as mth

# библиотеки работы с датой и временем
from datetime import timedelta
from datetime import datetime

# библиотеки визуализации
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

# вспомогательные функции ML
from sklearn.preprocessing import StandardScaler

# функции ML-кластеризации
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.cluster import KMeans

# метрики ML-кластеризации
from sklearn.metrics import silhouette_score

# статистические библиотеки
from scipy import stats as st
```

Вспомогательные функции визуализации

In [2]:
```
# определение  функции подписей для pie chart
# =====================================================
# на вход подаётся:
#     values - значения для построения pie chart
# на выходе - список строк в формате "pct% (value)"
# =====================================================
def make_autopct(values):
    def my_autopct(pct):
        total = sum(values)
        val = int(round(pct*total/100.0))
        return '{p:.2f}%  ({v:d})'.format(p=pct,v=val)
    return my_autopct
```

Загрузка и обзор данных

Определение функций загрузки и обзора данных

In [3]:
```
# определение функции загрузки csv-данных по http
# =====================================================
# на вход подаётся:
#     url - http-путь к файлу
#     file_name - имя файла
# на выходе - датафрейм с загруженными данными
# в случае отсутствия http-доступа к файлу бросается
# исключение ValueError
# =====================================================
def http_open_csv(url, file_name, sep=','):
    pth1 = url             # http-путь к файлу

    try:
        df = pd.read_csv(pth1 + file_name, sep=',')
        return df
    except:
        raise ValueError(
            "ERROR: CSV-file " + pth1 + file_name + " is unreachable ..."
        )
```

In [4]:
```
# определение функции обзора данных
# ================================================
# на вход подаётся датафрейм df
# на выходе:
#     - 10 случайных строк df
#     - информация df.info()
#     - количество явных дубликатов в строках df
#     - процент пропусков данных в столбцах df
# ================================================
def data_observe(df):
    row_num = 5    # количество отображаемых строк таблицы

    print('Размерность данных (row, col):', df.shape)
    print('============================\n')
```

```
    print('Произвольные строки таблицы:')
    print('============================')
    if len(df) >= row_num:
        display(df.sample(row_num))
    else:
        display(df)

    print('\nИнформация о таблице:')
    print('=====================')
    df.info()

    print('\nКоличество явных дубликатов в таблице:')
    print('======================================')
    print(df.duplicated().sum())

    print('\nПроцент пропусков в столбцах:')
    print('============================')
    display(pd.DataFrame(
                round((df.isna().mean()*100),2), columns=['NaNs, %'])
                .sort_values(by='NaNs, %', ascending=False
            )
        .style.format('{:.2f}')
        .background_gradient('coolwarm')
    )
```

HTTP-путь к файлам данных

In [5]:
```
data_url = 'https://code.s3.yandex.net/datasets/'
```

Обзор файла `mobile_dataset.csv`

Откроем и изучим содержимое файла `mobile_dataset.csv`:

In [6]:
```
try:
    mobile_dataset = http_open_csv(data_url, 'mobile_dataset.csv', sep=',')
    data_observe(mobile_dataset)
except ValueError as err:
    print(err)
```

```
Размерность данных (row, col): (74197, 3)
==========================

Произвольные строки таблицы:
============================
```

	event.time	event.name	user.id
49261	2019-10-26 11:14:48.905162	tips_show	e387d029-59eb-41b9-9be5-5548389c079c
318	2019-10-07 10:17:19.127817	advert_open	21b9ef95-e152-47e6-bb4b-284525c38064
42849	2019-10-23 21:46:28.884054	tips_show	4a692d22-992c-47d5-9676-9e6ffb11ca7d
39589	2019-10-22 22:10:35.529534	search_6	dc179afe-96e0-4330-a09f-8626a193e09f
41331	2019-10-23 15:33:31.062755	search_5	6309f885-0268-4944-8ef2-26ca83a14c49

```
Информация о таблице:
=====================
<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

Количество явных дубликатов в таблице:
======================================
0

Процент пропусков в столбцах:
============================
```

	NaNs, %
event.time	0.00
event.name	0.00
user.id	0.00

В таблице `mobile_dataset` 74197 строк и 3 столбца. Столбцы поименованы через точку. Целесообразно привести наименования к стилю snake_case.

Согласно описанию данных, датасет **`mobile_dataset.csv`** содержит колонки:

- `event.time` — время совершения события;
- `event.name` — название события;
- `user.id` — идентификатор пользователя, совершившего событие.

Все столбцы имеют тип `object` и содержат:

- строковое представление даты и времени;
- название события;
- идентификатор пользователя.

Целесообразно на этапе предобработки привести дату и время к типу `datetime`, а также на этапе EDA оценить:

- временной диапазон данных;
- множество событий;
- количество уникальных пользователей.

В данных отсутствуют явные дубликаты и пропуски.

Целочисленные и вещественные данные, за счёт которых можно было бы сократить использование памяти, в твблице отсутствуют.

Обзор файла `mobile_soures.csv`

Откроем и изучим содержимое файла `mobile_soures.csv`:

In [7]:
```
try:
    mobile_sources = http_open_csv(data_url, 'mobile_soures.csv', sep=',')
    data_observe(mobile_sources)
except ValueError as err:
    print(err)
```

```
Размерность данных (row, col): (4293, 2)
============================

Произвольные строки таблицы:
============================
```

	userId	source
2564	698c1208-8dc1-4f88-aff6-eab0bc6a3462	other
2746	f7e95cba-1566-47b0-a4f8-7f0dcc6d1060	yandex
837	ea17a900-2c22-42f9-9d6f-d86fabbe58cf	yandex
1762	0eb1474c-5e6b-4d59-acca-ad9863e608ab	google
37	1a3be56d-501d-4178-9f1a-059684f0b510	yandex

```
Информация о таблице:
====================
<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

Количество явных дубликатов в таблице:
======================================
0

Процент пропусков в столбцах:
=============================
```

	NaNs, %
userId	0.00
source	0.00

В таблице `mobile_sources` 4293 строки и 2 столбца. Столбцы поименованы в стиле camelCase. Целесообразно привести наименования к стилю snake_case.

Согласно описанию данных, датасет **`mobile_soures.csv`** содержит колонки:

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

Все столбцы имеют тип `object`.

Целесообразно оценить на этапе EDA:

- множество каналов;
- количество уникальных пользователей в таблице.

В данных отсутствуют явные дубликаты и пропуски.

Целочисленные и вещественные данные, за счёт которых можно было бы сократить использование памяти, в твблице отсутствуют.

Выводы

1. Предварительно, данные выглядят полными - в таблицах отсутствуют явные дубликаты и пропуски.

1. В то же время, данные выглядят "грязными", мало пригодными для анализа: все события "свалены в кучу", сложно понять как часто и долго пользователи находятся в приложении, какие типовые сценарии используют. В этой связи на этапе EDA следует провести очистку и обогащение данных новыми метриками и признаками.

1. **На этапе предварительной обработки** представляется целесообразным:

- привести названия колонок в обоих датасетах к единому стилю - snake_case;
- привести дату и время в таблице `mobile_dataset` к типу `datetime`.

1. **На этапе EDA** представляется целесообразным:

- сравнить уникальных пользователей в обеих таблицах по количеству и по составу;
- в таблице `mobile_dataset` оценить:
  - временной диапазон данных;
  - множество событий;
- в таблице `mobile_sources` оценить множество каналов.

Предварительная обработка данных

Переименование столбцов

Переименуем столбцы в обеих таблицах в едином стиле snake_case:

In [8]:
```
# переименование столбцов в таблице mobile_dataset
mobile_dataset.rename(
    columns={
        'event.time' : 'event_time',
        'event.name' : 'event_name',
        'user.id' : 'user_id'
    },
    inplace=True
)

# переименование столбцов в таблице mobile_sources
mobile_sources.rename(
    columns={
        'userId' : 'user_id'
    },
    inplace=True
)

# проверка результатов
print(mobile_dataset.columns)
print(mobile_sources.columns)
```

```
Index(['event_time', 'event_name', 'user_id'], dtype='object')
Index(['user_id', 'source'], dtype='object')
```

Столбцы в обеих таблицах поименованы в едином стиле snake_case. Перейдём к изменению типа столбца `'event_time'` таблицы `mobile_dataset`.

Приведение типов данных

Приведём столбец `'event_time'` таблицы `mobile_dataset` к типу данных `datetime`:

In [9]:
```
# приведение типов
mobile_dataset['event_time'] = pd.to_datetime(mobile_dataset['event_time'])

# проверка результатов
print(mobile_dataset.event_time.dtype)
mobile_dataset.head(2)
```

```
datetime64[ns]
```

Out[9]:

	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

Тип столбца `'event_time'` успешно изменён на `datetime`.

Выводы

1. Первичная предварительная обработка проведена в соответствии с выводами обзора данных.

1. Дополнительная, углубленная предобработка может потребоваться на этапе исследовательского анализа.

Исследовательский анализ данных (EDA)

На этапе EDA предполагается:

- исследовать состав значений имеющихся столбцов данных:
  - сравнить уникальных пользователей в обеих таблицах по количеству и составу;
  - поискать и устранить неявные дубликаты;
  - построить распределения разных типов событий (гистограммы частот, распределения по времени);
  - выделить наиболее популярные события;
- выбрать метод и разделить события на сессии для каждого пользователя, построить профили пользователей (как при когортном анализе);
- подумать, какими метриками возможно обогатить данные;
- построить и описать распределения придуманных метрик;
- сделать выводы на основе полученных распределений, возможно, предложить предварительную сегментацию на основе метрик.

Определение функций EDA и визуализации

In [10]:
```
# определение функции отображения распределений
# источников привлечения и событий в виде PieChart
# =====================================================
# на вход подаётся:
#     src_df - датафрейм с источниками
#     evt_df - датафрейм с событиями
# =====================================================
def draw_source_event_pie(src_df, evt_df):
    # задаём размер сетки для графиков
    plt.figure(figsize=(20, 10))

    # в таблице графиков — 2 столбца и 1 строка
    # 1. в первом строим распределение источников привлечения
    ax1 = plt.subplot(1, 2, 1)
    source_groupping = (
        src_df
        .groupby(by='source', as_index=False)
        .agg('count')
        .sort_values(by='user_id', ascending=False)
    )
    plt.pie(
        source_groupping.user_id,
        labels=source_groupping.source,
```

```
        autopct=make_autopct(source_groupping.user_id)
    )
    ax1.set_title('Распределение источников привлечения')

    # 2. во втором строим распределение событий
    ax2 = plt.subplot(1, 2, 2)
    event_groupping = (
        evt_df
        .groupby(by='event_name', as_index=False)
        .agg({'user_id':'count'})
        .sort_values(by='user_id', ascending=False)
    )
    plt.pie(
        event_groupping.user_id,
        labels=event_groupping.event_name,
        autopct=make_autopct(event_groupping.user_id)
    )
    ax2.set_title('Распределение событий')

    plt.tight_layout()
    plt.show()
```

In [11]:
```
# определение функции вывода PieChart для одного параметра
# ========================================================
# на вход подаётся:
#     df - датафрейм
#     group_by - имя столбца категорий
#     sort_by - имя столбца для подсчёта количества
#     title - Название диаграммы
#     figsize - размер фигуры (по умолчанию - (10, 10))
# ========================================================
def draw_pie(df, group_by, sort_by, title, figsize=(10, 10)):
    # задаём размер сетки для графиков
    plt.figure(figsize=figsize)

    # строим график
    ax1 = plt.subplot(1, 1, 1)
    groupping = (
        df
        .groupby(by=group_by, as_index=False)
        .agg('count')
        .sort_values(by=sort_by, ascending=False)
    )
    plt.pie(
        groupping[sort_by],
        labels=groupping[group_by],
        autopct=make_autopct(groupping[sort_by])
    )
    ax1.set_title(title)

    plt.tight_layout()
    plt.show()
```

In [12]:
```
# определение функции вывода boxplot для множества
# категорий
# ========================================================
# на вход подаётся:
#     df - датафрейм
#     x - имя столбца с категориями
#     y - имя столбца со значениями
#     title - название диаграммы
#     x_title - подпись по оси x
#     y_title - подпись по оси y
# ========================================================
def draw_boxplots(df, x, y, title, x_title, y_title):
    # построим диаграммы
    fig = px.box(df, x=x, y=y)

    # зададим названия гистограммы и осей
    fig.update_layout(
        title_text=title,
        xaxis_title_text=x_title,
        yaxis_title_text=y_title
    )

    fig.show()
```

In [13]:
```
# определение функции преобразования pivot_table в
# вертикальную таблицу и вывода boxplot для множества
# категорий
# ========================================================
# на вход подаётся:
#     df - таблица pivot_table
#     title - название диаграммы
#     x_title - подпись по оси x
#     y_title - подпись по оси y
# ========================================================
def melt_and_boxplot(df, title, x_title, y_title):

    new_features = df

    # превратим таблицу в вертикальную
    new_features = (
        new_features
        .melt(
            value_vars=new_features.columns,
            var_name='feature',
            ignore_index=False
        )
    )

    # построим диаграммы
    draw_boxplots(df=new_features,
                  x="feature",
                  y="value",
                  title=title,
                  x_title=x_title,
                  y_title=y_title)
```

Исследование состава значений имеющихся столбцов данных

Рассмотрим диапазоны и множества значений в столбцах таблиц `mobile_dataset` и `mobile_sources`:

In [14]:
```
# столбцы таблицы mobile_dataset

# диапазон времени
print('MIN event datetime:', mobile_dataset.event_time.min())
print('MAX event datetime:', mobile_dataset.event_time.max())
print(
    'Период наблюдений:',
    mobile_dataset.event_time.max() - mobile_dataset.event_time.min()
)
print()

# количество пользователей
print('Количество уникальных пользователей:',
      mobile_dataset.user_id.nunique())
print()

# сохраним отсортированный список уникальных пользователей
# для сверки со второй таблицей
dataset_unique_users = mobile_dataset.user_id.sort_values().unique()

# уникальные названия событий
print('Количество уникальных названий событий:',
      mobile_dataset.event_name.nunique())
print(
    '\nУникальные названия событий:',
    mobile_dataset.event_name.sort_values().unique()
)
```

```
MIN event datetime: 2019-10-07 00:00:00.431357
MAX event datetime: 2019-11-03 23:58:12.532487
Период наблюдений: 27 days 23:58:12.101130

Количество уникальных пользователей: 4293

Количество уникальных названий событий: 16

Уникальные названия событий: ['advert_open' 'contacts_call' 'contacts_show' 'favorites_add' 'map'
 'photos_show' 'search_1' 'search_2' 'search_3' 'search_4' 'search_5'
 'search_6' 'search_7' 'show_contacts' 'tips_click' 'tips_show']
```

Итак:

- мы имеем данные о событиях вновь привлечённых пользователей за 28 дней: с 7 октября по 3 ноября 2019 года;
- всего за этот период привлечены 4293 пользователя;
- всего в логе зарегистрировано 16 уникальных событий.

Согласно описанию данных, события имеют следующую расшифровку:

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

Общее количество уникальных событий в описании и датафрейме совпадает, следовательно, все события встречаются в данных и нет неописанных событий.

В то же время, события `contacts_show` и `show_contacts` выглядят, как неявные дубликаты. Тимлид подтвердил, что данная ошибка, скорее всего, возникла во время сбора данных из различных источников, и данные события следует отождествлять. Исправим неявные дубликаты позже.

События поиска `search_1` — `search_7`, как заявлено, являются различными. Тем не менее, при дальнейших исследованиях может быть интересно общее количество событий поиска в некотором разрезе данных. Не будем пока отождествлять разные события поиска - возможно, они окажутся хорошим набором признаков для кластеризации.

In [15]:
```
# столбцы таблицы mobile_sources

# количество пользователей
print('Количество уникальных пользователей:',
      mobile_sources.user_id.nunique())
print()

# уникальные названия каналов
print('Количество уникальных названий каналов:',
      mobile_sources.source.nunique())
print(
    '\nУникальные названия каналов:',
    mobile_sources.source.sort_values().unique()
)

# посчитаем, сколько пользователей в совпадают в обеих таблицах
print(
    '\nКоличество уникальных пользователей, совпавших с таблицей mobile_dataset:',
    sum(mobile_sources.user_id.sort_values().unique() == dataset_unique_users)
)
```

```
Количество уникальных пользователей: 4293

Количество уникальных названий каналов: 3

Уникальные названия каналов: ['google' 'other' 'yandex']

Количество уникальных пользователей, совпавших с таблицей mobile_dataset: 4293
```

Итак:

- общее количество привлечённых из разных каналов пользователей равно 4293;

- данные о всех них присутствуют в обеих таблицах;
- всего насчитывается 3 канала привлечения - 'google', 'other', 'yandex'.

Таким образом, можно считать, что данные в представленных таблицах непротиворечивы.

Выше мы обнаружили неявные дубликаты - события `contacts_show` и `show_contacts`. Приведём их к единому написанию - `contacts_show`:

In [16]:
```
# замена значений
mobile_dataset.event_name.where(
    mobile_dataset.event_name != 'show_contacts',
    other='contacts_show',
    inplace=True
)

# проверка
print('Количество уникальных названий событий:',
      mobile_dataset.event_name.nunique())
print(
    '\nУникальные названия событий:',
    mobile_dataset.event_name.sort_values().unique()
)
```

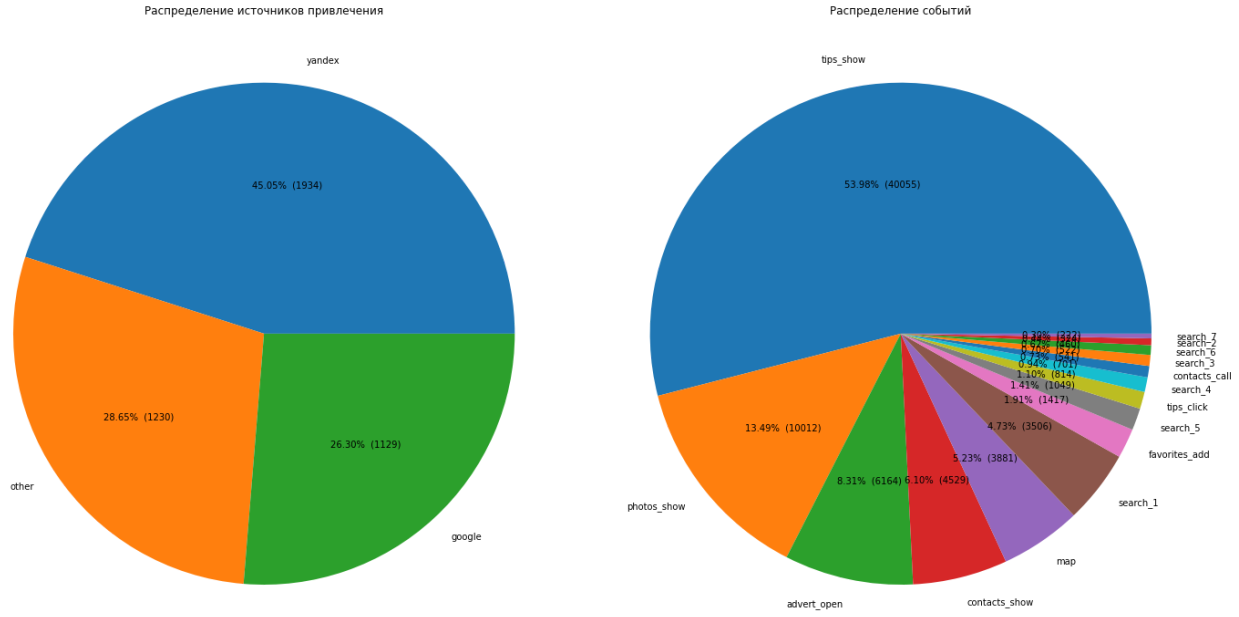
```
Количество уникальных названий событий: 15

Уникальные названия событий: ['advert_open' 'contacts_call' 'contacts_show' 'favorites_add' 'map'
 'photos_show' 'search_1' 'search_2' 'search_3' 'search_4' 'search_5'
 'search_6' 'search_7' 'tips_click' 'tips_show']
```

По результатам обработки неявных дубликатов у нас осталось 15 видов событий.

Рассмотрим распределения имеющихся данных:

- источников привлечения;
- событий;
- времени активности пользователей.

In [17]:
```
# визуализируем источники и события
draw_source_event_pie(mobile_sources, mobile_dataset)
```



Из построенных круговых диаграмм следует, что:

- **Для источников привлечения:**
  1. Наиболее популярным источником является `yandex` (1934 пользователя - свыше 45% от общего количества).
  2. Источник `google` привлёк 1129 пользователей (около 26.3% от общего количества).
  3. В целом, выборки количественно сравнимы для статистической значимости при проверке статистических гипотез.

- **Для событий:**
  1. Наиболее популярными событиями являются:
     - `tips_show` — пользователь увидел рекомендованные объявления (почти 54% от общего количества событий);
     - `photos_show` — просмотр фотографий в объявлении (13.49% от общего количества событий);
     - `advert_open` — открытие карточки объявления (8.31% от общего количества событий);
     - `contacts_show` — пользователь нажал на кнопку "посмотреть номер телефона" на карточке объявления (6.1% от общего количества событий);
     - `map` — пользователь открыл карту размещенных объявлений (5.23% от общего количества событий);
     - `search_1` — самое популярное событие поиска (4.73% от общего количества событий);
     - `favorites_add` — добавление объявления в избранное (1.91% от общего количества событий).
  2. Остальные события, в том числе другие виды поиска, составляют менее 1.5% от общего количества каждое.
  3. Специфические виды поиска, скорее всего, ввиду редкого их использования, не дадут нам хорошей дифференциации при кластеризации, поэтому ***для целей настоящего исследования их можно объединить с событием*** `search_1` ***в единое событие*** `search`.
  4. Пользователи крайне редко кликают на рекомендованные объявления (1.1%) и звонят из приложения (0.73%).

Замечания:

1. Наиболее частое событие `tips_show`, происходит в приложении автоматически, не зависит от действий пользователя. Однако, для того, чтобы оно произошло, пользователь должен, как минимум, войти в приложение. Поэтому данное событие можно считать частью пользовательской сессии и не следует удалять из

датафрейма.

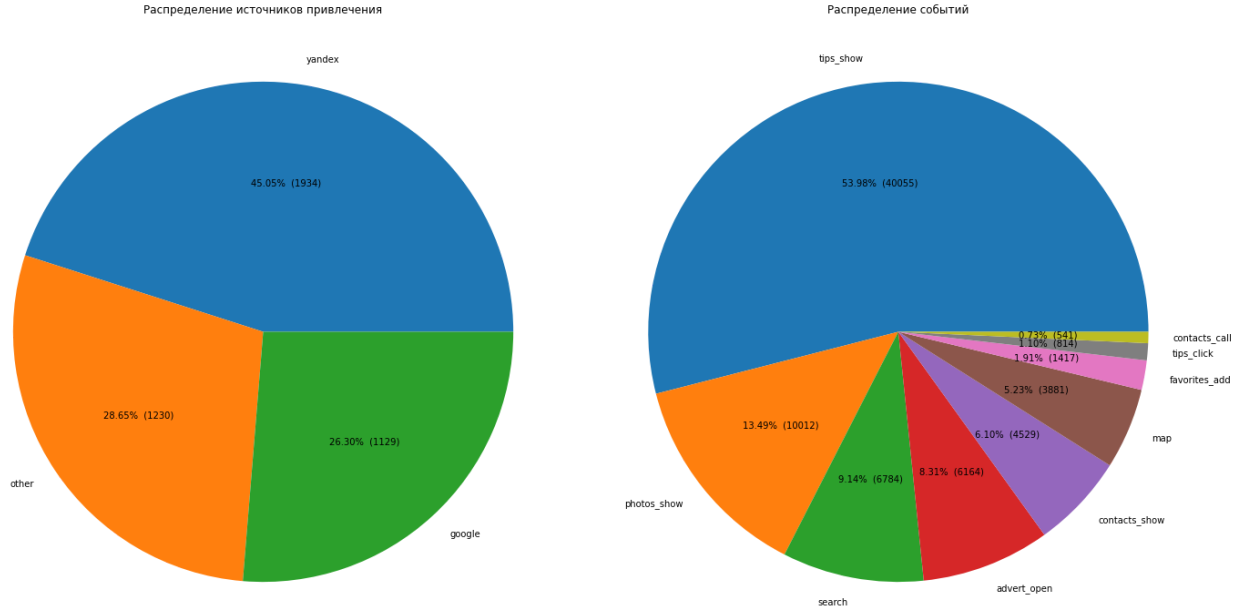
2. Событие `tips_click`, в свою очередь, хотя и является редким, но представляет собой действие, и его следует учитывать при выборе наиболее популярных событий.
3. Событие `contacts_call`, хотя и является действием, не в полной мере характеризует пользователей, поскольку им доступны и звонки вне приложения.

In [18]:
```
# сделаем копию исходного датасета, который будем "чистить"
clean_mobile_dataset = mobile_dataset.copy()

# объединение событий поиска
clean_mobile_dataset.event_name.where(
    ~mobile_dataset.event_name.isin(
        ['search_1', 'search_2', 'search_3',
         'search_4', 'search_5', 'search_6',
         'search_7']
    ),
    other='search',
    inplace=True
)

# проверка
print('Количество уникальных названий событий:',
      clean_mobile_dataset.event_name.nunique())
print(
    '\nУникальные названия событий:',
    clean_mobile_dataset.event_name.sort_values().unique()
)
print('\nОбщее количество событий в логе:', len(clean_mobile_dataset))
print('\nОбщее количество пользователей в логе:',
      clean_mobile_dataset.user_id.nunique())
```

```
Количество уникальных названий событий: 9

Уникальные названия событий: ['advert_open' 'contacts_call' 'contacts_show' 'favorites_add' 'map'
 'photos_show' 'search' 'tips_click' 'tips_show']

Общее количество событий в логе: 74197

Общее количество пользователей в логе: 4293
```

Итак, у нас осталось 9 событий. Снова построим распределения:

In [19]:
```
# визуализируем источники и события
draw_source_event_pie(mobile_sources, clean_mobile_dataset)
```



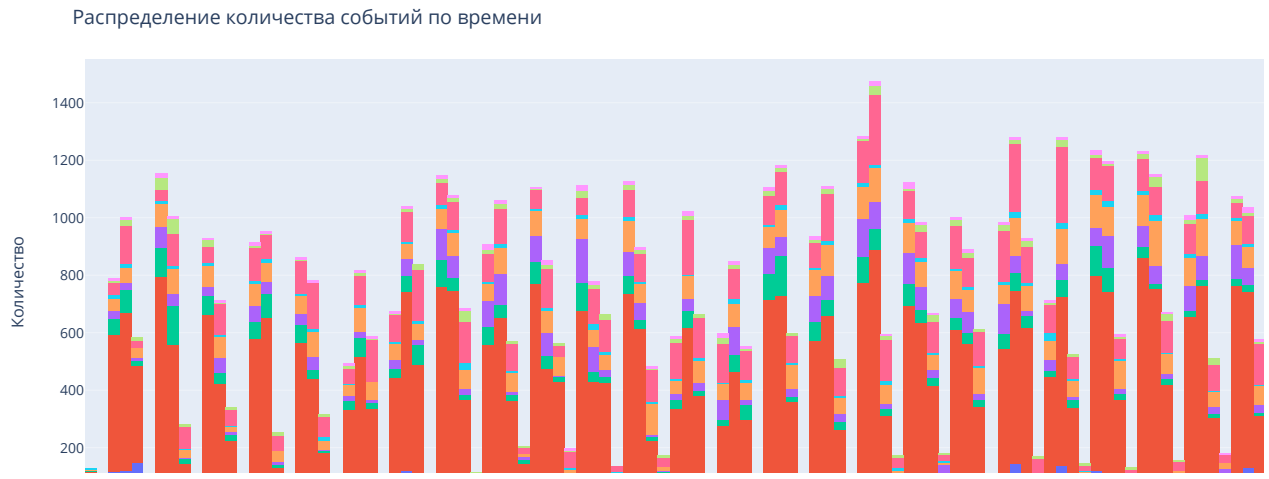
Наиболее значимые события в порядке убывания частоты:

- `tips_show` — пользователь увидел рекомендованные объявления (53.98% от общего количества событий);
- `photos_show` — просмотр фотографий в объявлении (13.49%);
- `search` — объединённое событие поиска (9.14%);
- `advert_open` — открытие карточки объявления (8.31%);
- `contacts_show` — пользователь нажал на кнопку "посмотреть номер телефона" на карточке объявления (6.1%);
- `map` — пользователь открыл карту размещенных объявлений (5.23%);
- `favorites_add` — добавление объявления в избранное (1.91%);
- `tips_click` — пользователь кликнул на рекомендованное объявление (1.1%);
- `contacts_call` — пользователь позвонил по номеру телефона на карточке объявления (0.73%).

Посмотрим на распределение событий во времени:

In [20]:
```
# используем удобную для анализа библиотеку plotly express
fig = px.histogram(
    data_frame=clean_mobile_dataset,
    x='event_time',
    color='event_name',
    labels={
        'event_name':'События',
    },
)

# зададим названия гистограммы и осей
```

```
fig.update_layout(
    title_text='Распределение количества событий по времени',
    xaxis_title_text='Время события',
    yaxis_title_text='Количество'
)

fig.show()
```



Из гистограммы распределения событий по времени следует, что использование приложения за период наблюдений выглядит более-менее равномерно. Тем не менее:

- наблюдаются провалы в использовании с 9 по 12 октября и 2 ноября;
- в остальные дни в среднем количество зарегистрированных событий равномерно неубывает;
- прослеживается четкая временная структура - в ночные часы активность пользователей минимальна, в вечерние - падает.

Общее количество зарегистрированных целевых событий проседает каждую неделю на 6-7 день кратно от начала периода наблюдений. ***Это может свидетельствовать о недельной цикличности покупательской активности пользователей.*** Это подтверждается тем фактом, что количество показанных рекомендаций (tips_show) имеет ту же структуру.

Количество целевых событий, совершённых во вторую неделю и далее, превышает их количество в первую неделю. Вероятно, это может свидетельствовать о неплохом удержании в приложении: к активным пользователям первой недели добавляются пользователи второй и последующих недель, но большого оттока не происходит.

Нечто, похожее на недельную цикличность, можно заметить и для события добавления в избранное, звонков, а также, в меньшей степени, для кликов по рекламным объявлениям.

Пользование картой в приложении снижается к концу месяца.

Посчитаем количество пользователей, совершивших каждое событие:

In [21]:
```
event_users = (
    clean_mobile_dataset
    .groupby(by='event_name'
    )
    .agg({'user_id':'nunique'})
    .sort_values(by='user_id', ascending=False)
)
event_users
```

Out[21]:

event_name	user_id
tips_show	2801
search	1666
map	1456
photos_show	1095
contacts_show	981
advert_open	751
favorites_add	351
tips_click	322
contacts_call	213

Итак, из 4293 пользователей в логе:

- 2801 увидели рекомендации - возможно, рекомендательную систему можно отключить/заблокировать, а может быть, она работает некорректно (***обратить внимание разработчиков***);
- 1666 предпочитают искать объявления поиском, 1456 - на карте;
- 981 посмотрел контакты продавца;
- при этом только 751 открыл само объявление (возможна некорректная работа системы логирования действий - по логике приложения кнопка "контакты" находится на странице объявления);
- редкие пользователи добавляют объявление в избранное (351) и кликают по рекомендациям (322).

Общая конверсия за весь период наблюдений составляет около 23% :

In [22]:
```
cr_total = (
    event_users.loc['contacts_show', 'user_id'] /
    clean_mobile_dataset.user_id.nunique()
)
cr_total
```

Out[22]:
```
0.22851153039832284
```

Следует обратить внимание продакта на малое количество событий `favorites_add` и `tips_click`. Возможно, использование этих подсистем приложения неудобно для пользователей.

Посчитаем количество разных событий для каждого пользователя:

In [23]:
```
user_events = (
    clean_mobile_dataset
    .groupby(by=['user_id', 'event_name'])
    .agg({'event_time':'count'})
    .reset_index()
    .rename(
        columns={'event_time':'event_count'}
    )
)
user_features = (
    user_events
    .pivot(index='user_id', columns='event_name', values='event_count')
    .fillna(0)
)
user_features.sort_values(by='photos_show', ascending=False).head(20)
```

Out[23]:

event_name	advert_open	contacts_call	contacts_show	favorites_add	map	photos_show	search	tips_click	tips_show
user_id									
e13f9f32-7ae3-4204-8d60-898db040bcfc	65.0	0.0	6.0	24.0	23.0	177.0	41.0	0.0	129.0
9c78948d-5850-4916-9d7f-341fec1b7737	0.0	0.0	7.0	0.0	0.0	149.0	0.0	0.0	0.0
97d1107f-1d9c-4086-b2d9-83985afecca3	8.0	0.0	0.0	9.0	0.0	126.0	6.0	0.0	0.0
13140930-df18-4793-a230-7cca5c8813db	0.0	0.0	1.0	1.0	0.0	111.0	1.0	0.0	0.0
62a5375a-eb94-4ed2-90ef-3d79d8e0c359	0.0	8.0	11.0	0.0	0.0	108.0	0.0	0.0	0.0
06216934-8394-482e-a9fd-001f93bbebde	0.0	0.0	1.0	0.0	0.0	104.0	0.0	0.0	0.0
c0097deb-203a-42c7-baba-7d54374fb97f	0.0	4.0	4.0	0.0	0.0	102.0	22.0	0.0	0.0
f6f94ebe-e69a-4ae3-9fb0-312d52d35826	12.0	0.0	1.0	1.0	11.0	90.0	19.0	0.0	10.0
05b35678-bbc6-47f0-b552-ab639249a0d4	0.0	0.0	0.0	0.0	0.0	85.0	12.0	0.0	0.0
06edf71c-b725-47dc-acfe-0c78f079fe8f	0.0	0.0	0.0	1.0	0.0	84.0	2.0	0.0	0.0
9f95c9ee-750c-4dd5-8b2a-275105f9c9e7	0.0	0.0	3.0	0.0	0.0	75.0	7.0	0.0	0.0
9f9034e9-966d-4052-b3ab-5389f9585eb3	0.0	2.0	3.0	0.0	0.0	74.0	7.0	0.0	0.0
f71afbca-b830-4a92-8d1d-03824a8d1b6e	0.0	0.0	1.0	0.0	0.0	74.0	0.0	0.0	0.0
6383ff6a-04b8-4562-a98f-bb4f760d3c39	0.0	0.0	6.0	0.0	0.0	70.0	0.0	0.0	0.0
2b453c9b-3f97-4747-ae1b-107d46767025	0.0	0.0	0.0	0.0	0.0	70.0	3.0	0.0	0.0
1af9ffcd-2c77-4de0-9d35-3ff30604c9bd	0.0	1.0	1.0	0.0	0.0	69.0	5.0	0.0	0.0
b3de93e2-1b08-4b1e-9fc9-44bb0eb05999	1.0	0.0	0.0	1.0	0.0	68.0	0.0	0.0	0.0
d5e14ec3-7ae5-4598-ad36-f626b3ce24e3	20.0	0.0	0.0	8.0	0.0	63.0	10.0	0.0	0.0
bfe95d6c-79e3-4532-a8b7-e2270d7c8a65	0.0	0.0	9.0	0.0	0.0	63.0	49.0	0.0	0.0
25069cad-0d00-48cb-a627-0871a877307e	0.0	2.0	7.0	0.0	0.0	62.0	58.0	0.0	0.0

Из таблицы видно, что существуют пользователи, которые только просматривали фото и контакты. Просматривать контакты продавца по логике приложения должно быть возможно только из объявления, но событие `advert_open` не зарегистрировано.

Возможно, имеет место недоработка в системе логирования событий. Следует обратить на это внимание продакта!

Данная гипотеза подтверждается также наличием пользователей, которые только просматривали объявления, но при этом не пользовались поиском, и им не было показано ни одной рекомендации:

In [24]:
```
user_features.sort_values(by='advert_open', ascending=False).head(1)
```

Out[24]:

event_name	advert_open	contacts_call	contacts_show	favorites_add	map	photos_show	search	tips_click	tips_show
user_id									
0d5c7fc6-7a74-4a7d-a7f6-f19a739365f6	137.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Отсортируем таблицу по убыванию целевого события:

In [25]:
```
user_features.sort_values(by='contacts_show', ascending=False).head(10)
```

Out[25]:

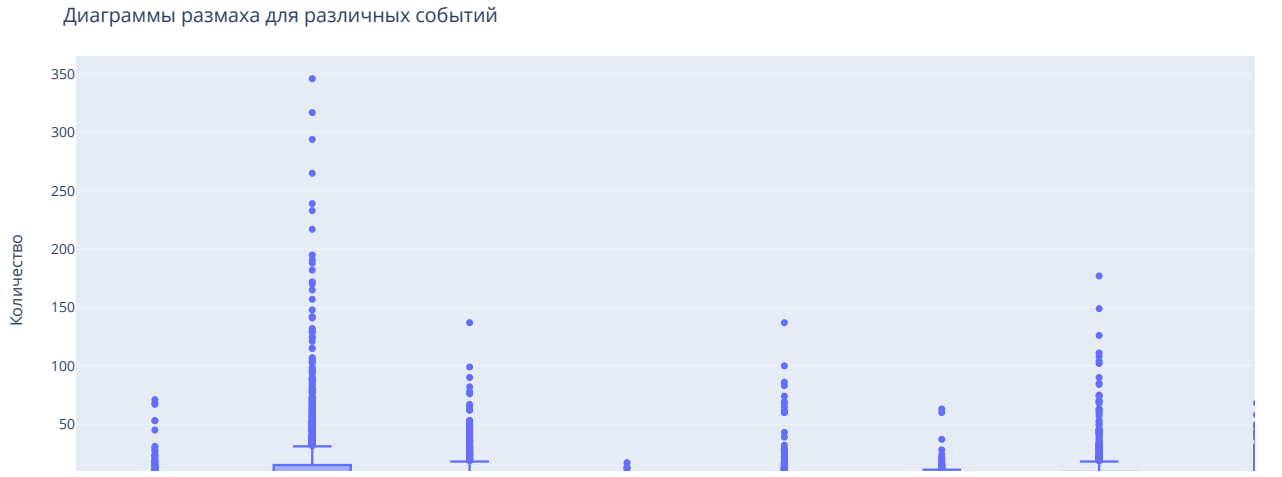
event_name user_id	advert_open	contacts_call	contacts_show	favorites_add	map	photos_show	search	tips_click	tips_show
e38cb669-7335-4d56-9de5-c8d5d2f13fd3	1.0	0.0	137.0	1.0	19.0	0.0	2.0	7.0	195.0
320cab3c-e823-4dff-8c01-c4253764640a	40.0	0.0	100.0	0.0	16.0	0.0	6.0	0.0	191.0
cb36854f-570a-41f4-baa8-36680b396370	0.0	0.0	86.0	0.0	68.0	0.0	0.0	7.0	317.0
be1449f6-ca45-4f94-93a7-ea4b079b8f0f	0.0	0.0	83.0	0.0	67.0	0.0	0.0	30.0	217.0
9b835c74-8ede-4586-9f59-e5473aa48de2	25.0	0.0	74.0	1.0	27.0	0.0	0.0	1.0	121.0
955bd7b0-8da8-49df-adee-546b59347634	4.0	0.0	69.0	4.0	0.0	0.0	0.0	0.0	170.0
fffb9e79-b927-4dbb-9b48-7fd09b23a62b	0.0	0.0	68.0	0.0	2.0	0.0	0.0	0.0	233.0
0a59892f-3578-484b-af84-eb3b2298fb8c	0.0	0.0	65.0	0.0	0.0	0.0	0.0	0.0	0.0
786b9f36-e41c-4a17-870e-68b329695647	0.0	0.0	62.0	0.0	0.0	0.0	0.0	0.0	3.0
a83c2011-d536-4c57-8841-d1b6aefd6311	0.0	0.0	61.0	0.0	12.0	0.0	0.0	0.0	28.0

Из данных следует, что некоторые пользователи кликали на рекомендации. По логике приложения, в этом случае должно было открыться объявление (`advert_open`), однако количество таких событий для этих пользователей равно 0.

Кроме того, есть также, по крайней мере, один пользователь (user_id = '0a59892f-3578-484b-af84-eb3b2298fb8c'), 65 раз совершивший целевое событие, но не совершивший ни одного другого события.

Построим диаграммы размаха для различных событий:

In [26]:
```
# построим диаграммы
draw_boxplots(df=user_events,
              x="event_name",
              y="event_count",
              title='Диаграммы размаха для различных событий',
              x_title='Событие',
              y_title='Количество')
```



Диаграммы размаха показывают, что верхний ус самого частого события (`tips_show`) отсекает частоты выше 31 показа, как выбросы.

Кроме того, видно, что:

- меньше всего разброс значений для событий `tips_click` и `contacts_call` ;
- достаточно компактно распределены события `search` , `map` и `favorites_add` ;
- разброс значений событий `advert_open` и `contacts_show` примерно одинаков;
- более всего подвержены выбросам события `tips_show` и `photos_show` .

Масштабировав графики к диапазону от 0 до 40 событий, а также приняв за среднее медиану в условиях выбросов, можно отметить, что:

- в среднем пользователи:
  - видят 8 рекомендованных объявлений, 5 фотографий;
  - по 1 разу кликают на рекомендации и пользуются картой;
  - 2 раза просматривают контакт, звонят продавцу в приложении и добавляют объявление в избранное;
  - 4 раза открывали объявления;
  - 3 раза пользовались поиском;
- 75% пользователей:
  - не более 3 раз ищут объявления на карте, кликают на рекомендации, или звонят продавцам;
  - не более 4 раз пользуются поиском, или просматривают контакты;
  - не более 9 раз открывают объявления и просматривают фото;
  - не более 5 раз добавляют объявления в избранное;
  - не более 15 раз просматривают рекомендации.

На этом окончим исследование исходных данных. Поскольку мы не можем детально проработать ситуацию для каждого из 4293 пользователей, оставим данные, как есть, и сосредоточимся на их обогащении, а именно, определении сессий использования приложения и связанных с ними метрик и признаков.

Отметим, что выше мы начали формировать матрицу признаков пользователей (`user_event_features`), которую в дальнейшем можно будет использовать для сегментации пользователей.

Выделение сессий использования приложения

Выше на гистограмме распределения количества событий по времени мы отмечали, что использование приложения обладает не только недельной цикличностью, но и варьируется в течение дня. В этой связи представляется целесообразным использовать не суточную модель сессии, а на основе таймаута сессии.

Будем считать, что для каждого пользователя события с разницей по времени более 30 минут принадлежат разным сессиям.

Будем использовать сквозную нумерацию сессий (`'session_id'`) для каждого всех пользователей (если предварительно отсортировать датасет по пользователю и времени события, то в результате сквозной нумерации мы получим для каждого пользователя последовательно-инкрементные целочисленные последовательности `'session_id'` , которые при необходимости легко трансформируются в последовательности идентификаторов сессий виде 1, 2, 3,...).

In [27]:
```
# отсортируем датасет по пользователям и времени
clean_mobile_dataset.sort_values(by=['user_id', 'event_time'], inplace=True)

# определение идентификаторов сессий
# 1) пометим с помощью группировки и cumsum события, отстоящие
# более чем на 30 минут
grp = (
    clean_mobile_dataset.groupby('user_id')['event_time'].diff() >
    pd.Timedelta('30Min')
).cumsum()

# 2) сгруппируем датасет по пользователям и grp, вычислим session_id
# с помощью функции ngroup()
clean_mobile_dataset['session_id'] = (
    clean_mobile_dataset.groupby(['user_id', grp], sort=False).ngroup() + 1
)

clean_mobile_dataset.head(20)
```

Out[27]:

	event_time	event_name	user_id	session_id
805	2019-10-07 13:39:45.989359	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	1
806	2019-10-07 13:40:31.052909	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	1
809	2019-10-07 13:41:05.722489	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	1
820	2019-10-07 13:43:20.735461	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	1
830	2019-10-07 13:45:30.917502	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	1
831	2019-10-07 13:45:43.212340	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	1
832	2019-10-07 13:46:31.033718	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	1
836	2019-10-07 13:47:32.860234	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	1
839	2019-10-07 13:49:41.716617	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	1
6541	2019-10-09 18:33:55.577963	map	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	2
6546	2019-10-09 18:35:28.260975	map	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	2
6565	2019-10-09 18:40:28.738785	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	2
6566	2019-10-09 18:42:22.963948	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	2
36412	2019-10-21 19:52:30.778932	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	3
36416	2019-10-21 19:53:17.165009	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	3
36419	2019-10-21 19:53:38.767230	map	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	3
36421	2019-10-21 19:54:45.009859	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	3
36423	2019-10-21 19:54:56.854811	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	3
36430	2019-10-21 19:56:49.417415	map	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	3
36435	2019-10-21 19:57:21.124551	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	3

Итак сессии длиной не более 30 минут выделены. Посчитаем их количество:

In [28]:
```
print(
    'Общее количество сессий в датасете:',
    clean_mobile_dataset.session_id.nunique()
)
print(
    'Среднее количество сессий на пользователя:',
    clean_mobile_dataset.session_id.nunique() /
    clean_mobile_dataset.user_id.nunique()
)
```

```
Общее количество сессий в датасете: 10368
Среднее количество сессий на пользователя: 2.4150943396226414
```

Итак, у нас имеется 10368 сессий для 4293 пользователей (в среднем по 2.4 сессии на человека).

Обогащение данных метриками и признаками

Начнём с метрик и признаков, связанных с сессиями.

1. Посчитаем среднее количество сессий в день для каждого пользователя.

In [29]:
```
# зафиксируем день, в который произошло каждое событие
clean_mobile_dataset['event_date'] = clean_mobile_dataset['event_time'].dt.date

# проверим, сколько сессий лежит на стыке соседних суток (в %)
len(
    clean_mobile_dataset
    .groupby(by='session_id', as_index=False)
    .agg({'event_date':'nunique'})
    .rename(columns={'event_date':'event_count'})
    .sort_values(by='event_count', ascending=False)
    .query('event_count > 1')
) / clean_mobile_dataset.session_id.nunique()
```

Out[29]:
```
0.005208333333333333
```

Итак, менее 1% ночных сессий могут быть посчитаны дважды при вычислении метрик. Это допустимая погрешность.

Посчитаем среднее количество сессий в день для каждого пользователя:

In [30]:
```
# попределим минимальный и максимальный номер сессии в течение суток
user_sessions = (
    clean_mobile_dataset
    .groupby(by=['user_id', 'event_date'])
    .agg({'session_id':['min', 'max']})
)

# переименуем столбцы
user_sessions.columns = ['min_session_id', 'max_session_id']

# посчитаем количество сессий
user_sessions['session_count'] = (
    user_sessions['max_session_id'] - user_sessions['min_session_id'] + 1
)

# восстановим индекс, удалим лишние столбцы
user_sessions = user_sessions.reset_index().drop(
    columns=['min_session_id', 'max_session_id']
)

# определим среднее количество сессий в день для каждого пользователя
user_sessions = (
    user_sessions
    .groupby(by='user_id')
    .agg({'session_count':'mean'})
    .rename(columns={'session_count':'sessions_per_day'})
)

# добавим среднее количество сессий к таблице признаков
user_features = user_features.join(user_sessions)

user_features.head()
```

Out[30]:

user_id	advert_open	contacts_call	contacts_show	favorites_add	map	photos_show	search	tips_click	tips_show	sessions_per_day
0001b1d5-b74a-4cbf-aeb0-7df5947bf349	0.0	0.0	0.0	0.0	6.0	0.0	0.0	0.0	29.0	1.000000
00157779-810c-4498-9e05-a1e9e3cedf93	2.0	5.0	11.0	2.0	0.0	33.0	18.0	0.0	0.0	1.000000
00463033-5717-4bf1-91b4-09183923b9df	0.0	0.0	0.0	0.0	0.0	10.0	0.0	0.0	0.0	1.000000
004690c3-5a84-4bb7-a8af-e0c8f8fca64e	5.0	0.0	0.0	0.0	6.0	0.0	17.0	0.0	4.0	1.166667
00551e79-152e-4441-9cf7-565d7eb04090	0.0	3.0	3.0	0.0	0.0	1.0	1.0	0.0	0.0	1.000000

Мы обогатили таблицу признаков средним количеством сессий в день. Таким образом, мы учли активность пользователя в течение дня.

2. Посчитаем общее количество сессий для пользователей.

In [31]:
```
# определим минимальный и максимальный номер сессии для пользователя
# это правомочно, т.к. мы нумеровали сессии, упорядочив по пользователям
# используем ту же переменную user_sessions
user_sessions = (
    clean_mobile_dataset
    .groupby(by='user_id')
    .agg({'session_id':['min', 'max']})
)

# переименуем столбцы
user_sessions.columns = ['min_session_id', 'max_session_id']

# посчитаем количество сессий
user_sessions['session_count'] = (
    user_sessions['max_session_id'] - user_sessions['min_session_id'] + 1
)

# удалим лишние столбцы
user_sessions = user_sessions.drop(
    columns=['min_session_id', 'max_session_id']
)

# добавим общее количество сессий к таблице признаков
user_features = user_features.join(user_sessions)

user_features.head()
```

Out[31]:

user_id	advert_open	contacts_call	contacts_show	favorites_add	map	photos_show	search	tips_click	tips_show	sessions_per_day	session_count
0001b1d5-b74a-4cbf-aeb0-7df5947bf349	0.0	0.0	0.0	0.0	6.0	0.0	0.0	0.0	29.0	1.000000	4
00157779-810c-4498-9e05-a1e9e3cedf93	2.0	5.0	11.0	2.0	0.0	33.0	18.0	0.0	0.0	1.000000	6
00463033-5717-4bf1-91b4-09183923b9df	0.0	0.0	0.0	0.0	0.0	10.0	0.0	0.0	0.0	1.000000	1
004690c3-5a84-4bb7-a8af-e0c8f8fca64e	5.0	0.0	0.0	0.0	6.0	0.0	17.0	0.0	4.0	1.166667	6
00551e79-152e-4441-9cf7-565d7eb04090	0.0	3.0	3.0	0.0	0.0	1.0	1.0	0.0	0.0	1.000000	3

Общее количество сессий добавлено к таблице признаков.

3. Определим среднюю длину сессии для каждого пользователя (в секундах).

In [32]:
```
# определим начало и конец сессии
user_sessions = (
    clean_mobile_dataset
    .groupby(by=['user_id', 'session_id'])
    .agg({'event_time':['min', 'max']})
```

```
)

# переименуем столбцы
user_sessions.columns = ['session_start', 'session_end']

# вычислим длительность сессии в timedelta
user_sessions['session_len'] = (
    (user_sessions['session_end'] - user_sessions['session_start'])
)

# переведём длительность в секунды
user_sessions['session_len'] = user_sessions['session_len'].astype('timedelta64[s]')

# восстановим индекс, удалим лишние столбцы
user_sessions = user_sessions.reset_index().drop(
    columns=['session_start', 'session_end']
)

# определим среднюю длину сессии для каждого пользователя
user_sessions = (
    user_sessions
    .groupby(by='user_id')
    .agg({'session_len':'mean'})
)

# добавим общее количество сессий к таблице признаков
user_features = user_features.join(user_sessions)

user_features.head()
```

Out[32]:

user_id	advert_open	contacts_call	contacts_show	favorites_add	map	photos_show	search	tips_click	tips_show	sessions_per_day	session_count	session_len
0001b1d5-b74a-4cbf-aeb0-7df5947bf349	0.0	0.0	0.0	0.0	6.0	0.0	0.0	0.0	29.0	1.000000	4	689.750000
00157779-810c-4498-9e05-a1e9e3cedf93	2.0	5.0	11.0	2.0	0.0	33.0	18.0	0.0	0.0	1.000000	6	1961.833333
00463033-5717-4bf1-91b4-09183923b9df	0.0	0.0	0.0	0.0	0.0	10.0	0.0	0.0	0.0	1.000000	1	1482.000000
004690c3-5a84-4bb7-a8af-e0c8f8fca64e	5.0	0.0	0.0	0.0	6.0	0.0	17.0	0.0	4.0	1.166667	6	1107.000000
00551e79-152e-4441-9cf7-565d7eb04090	0.0	3.0	3.0	0.0	0.0	1.0	1.0	0.0	0.0	1.000000	3	186.333333

Средняя длина сессий добавлена к таблице признаков. Ещё одним признаком, связанным с пользовательскими сессиями, является среднее количество событий в сессии.

4. Определение среднего количества событий в сессии для каждого пользователя.

In [33]:

```
# определим количество событий в каждой сессии каждого пользователя
user_sessions = (
    clean_mobile_dataset
    .groupby(by=['user_id', 'session_id'], as_index=False)
    .agg({'event_name':'count'})
    .rename(columns={'event_name':'session_events'})
)

# определим среднее количество событий в сессии для каждого пользователя
user_sessions = (
    user_sessions
    .groupby(by='user_id')
    .agg({'session_events':'mean'})
)

# добавим среднее количество событий в сессии к таблице признаков
user_features = user_features.join(user_sessions)

user_features.head()
```

Out[33]:

user_id	advert_open	contacts_call	contacts_show	favorites_add	map	photos_show	search	tips_click	tips_show	sessions_per_day	session_count	session_len	session_events
0001b1d5-b74a-4cbf-aeb0-7df5947bf349	0.0	0.0	0.0	0.0	6.0	0.0	0.0	0.0	29.0	1.000000	4	689.750000	8.750000
00157779-810c-4498-9e05-a1e9e3cedf93	2.0	5.0	11.0	2.0	0.0	33.0	18.0	0.0	0.0	1.000000	6	1961.833333	11.833333
00463033-5717-4bf1-91b4-09183923b9df	0.0	0.0	0.0	0.0	0.0	10.0	0.0	0.0	0.0	1.000000	1	1482.000000	10.000000
004690c3-5a84-4bb7-a8af-e0c8f8fca64e	5.0	0.0	0.0	0.0	6.0	0.0	17.0	0.0	4.0	1.166667	6	1107.000000	5.333333
00551e79-152e-4441-9cf7-565d7eb04090	0.0	3.0	3.0	0.0	0.0	1.0	1.0	0.0	0.0	1.000000	3	186.333333	2.666667

Среднее количество событий в сессий добавлено к таблице признаков. С признаками и метриками, связанными с сессиями пользовательской активности, разобрались.

Выше мы отметили, что пользовательская активность имеет недельную цикличность. Следовательно, можно предположить, что пользователи, пришедшие в разные дни недели, ведут себя по-разному.

5. Определим день недели первого события для каждого пользователя.

In [34]:
```
# определим дату первого события для каждого пользователя
user_acquisition = (
    clean_mobile_dataset
    .sort_values(by=['user_id', 'event_time'])
    .groupby(by='user_id')
    .agg({'event_date':'first'})
    .rename(columns={'event_date':'acq_date'})
)

# приведём тип столбца
user_acquisition['acq_date'] = pd.to_datetime(user_acquisition.acq_date)

# определим день недели
user_acquisition['acq_week_day'] = user_acquisition['acq_date'].dt.weekday

user_acquisition.head()
```

Out[34]:

user_id	acq_date	acq_week_day
0001b1d5-b74a-4cbf-aeb0-7df5947bf349	2019-10-07	0
00157779-810c-4498-9e05-a1e9e3cedf93	2019-10-19	5
00463033-5717-4bf1-91b4-09183923b9df	2019-11-01	4
004690c3-5a84-4bb7-a8af-e0c8f8fca64e	2019-10-18	4
00551e79-152e-4441-9cf7-565d7eb04090	2019-10-25	4

Дни недели привлечения пользователей определены. Добавим их в таблицу признаков:

In [35]:
```
# добавим день недели первого события к таблице признаков
user_features = user_features.join(user_acquisition[['acq_week_day']])

user_features.head()
```

Out[35]:

user_id	advert_open	contacts_call	contacts_show	favorites_add	map	photos_show	search	tips_click	tips_show	sessions_per_day	session_count	session_len	session_events
0001b1d5-b74a-4cbf-aeb0-7df5947bf349	0.0	0.0	0.0	0.0	6.0	0.0	0.0	0.0	29.0	1.000000	4	689.750000	8.750000
00157779-810c-4498-9e05-a1e9e3cedf93	2.0	5.0	11.0	2.0	0.0	33.0	18.0	0.0	0.0	1.000000	6	1961.833333	11.833333
00463033-5717-4bf1-91b4-09183923b9df	0.0	0.0	0.0	0.0	0.0	10.0	0.0	0.0	0.0	1.000000	1	1482.000000	10.000000
004690c3-5a84-4bb7-a8af-e0c8f8fca64e	5.0	0.0	0.0	0.0	6.0	0.0	17.0	0.0	4.0	1.166667	6	1107.000000	5.333333
00551e79-152e-4441-9cf7-565d7eb04090	0.0	3.0	3.0	0.0	0.0	1.0	1.0	0.0	0.0	1.000000	3	186.333333	2.666667

День недели первого события добавлен к таблице признаков. Всего мы добавили 5 признаков и метрик, характеризующих поведение пользователей в приложении. Изучим добавленные метрики.

Исследование обогащённых данных

Охарактеризуем вновь добавленные признаки:

In [36]:
```
(
    user_features[['sessions_per_day', 'session_count',
                   'session_len', 'session_events']]
    .describe()
)
```

Out[36]:

	sessions_per_day	session_count	session_len	session_events
count	4293.000000	4293.000000	4293.000000	4293.000000
mean	1.246549	2.415094	855.089452	8.291200
std	0.506747	3.536466	943.535030	8.393259
min	1.000000	1.000000	0.000000	1.000000
25%	1.000000	1.000000	238.333333	3.666667
50%	1.000000	1.000000	562.000000	6.000000
75%	1.333333	3.000000	1149.000000	10.000000
max	6.000000	99.000000	9660.500000	104.000000

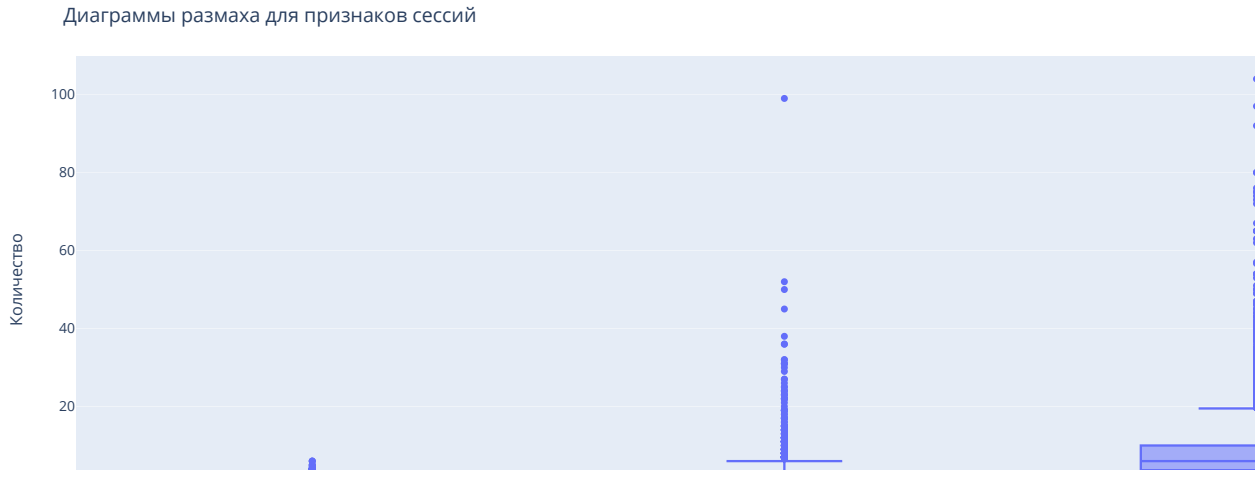
Из полученных характеристик следует:

- пользователи приложения заходят в него в среднем от 1 до 6 раз в день (среднее - 1.25), причём половина из них бывает в нём в среднем не чаще 1 раза, а 75% - не чаще 2 раз;
- общее количество сессий для пользователей варьируется от 1 до 99 (среднее - 2.42), причём половина из них была в приложении не более 1 раза, а 75% - не более 3 раз;

- средняя длина сессии варьируется от 0 до 9660.5 секунды, при этом половина пользователей непрерывно проводила в приложении в среднем не более 562 секунд, а 75% - не более 1149 секунд;
- в среднем на каждого пользователя в рамках одной сессии приходится от 1 до 104 событий, для 50% пользователей - не более 6 событий, для 75% - не более 10%.

Проиллюстрируем выводы диаграммами размаха для этих признаков:

In [37]:
```
# ввиду разного масштаба отдельно рассмотрим признаки ['sessions_per_day',
# 'session_count', 'session_events'] и 'session_len'

# выделим признаки ['sessions_per_day', 'session_count', 'session_events']
new_features = user_features[['sessions_per_day', 'session_count',
                              'session_events']]

melt_and_boxplot(new_features,
                 title='Диаграммы размаха для признаков сессий',
                 x_title='Признак',
                 y_title='Количество')

# выделим признак ['session_len']
new_features = user_features[['session_len']]

melt_and_boxplot(new_features,
                 title='Диаграммы размаха для признаков сессий (продолжение)',
                 x_title='Признак',
                 y_title='Количество')
```



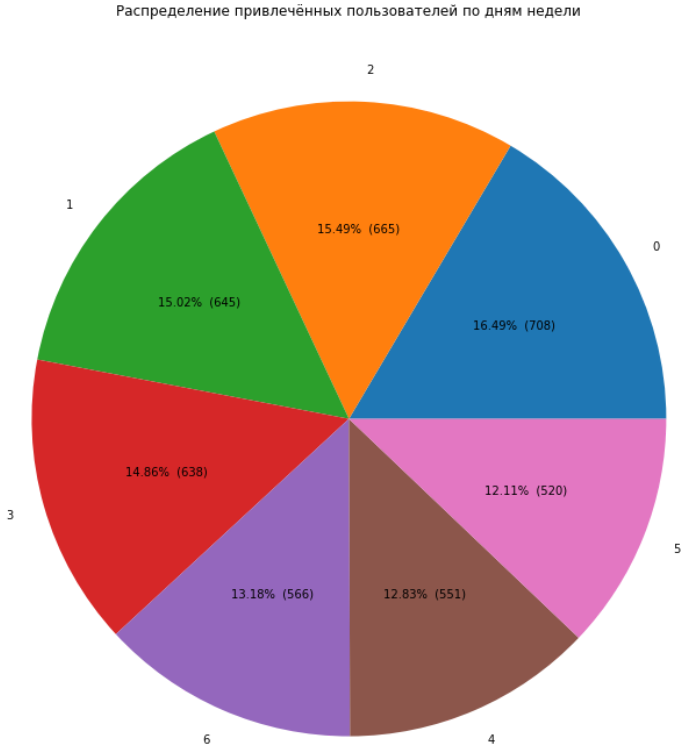
Из диаграмм видно, что:

- для средней длины сессии выбросами можно считать сессии длиной свыше 2514 секунд;
- для среднего количества событий в сессии - значения более 19.5;
- для количества сессий - больше 6;
- для среднего количества сессий в течении дня - более 2.

Отобразим распределение признака `acq_week_day` (0 - понедельник, ... , 6 - воскресенье):

In [38]:
```
new_features = user_features[['acq_week_day']]
new_features = (
    new_features
    .melt(
        value_vars=new_features.columns,
```

```
        var_name='feature',
        ignore_index=False
    )
)

draw_pie(
    df=new_features,
    group_by='value',
    sort_by='feature',
    title='Распределение привлечённых пользователей по дням недели'
)
```



Из диаграммы видно, что в целом пользователи равномерно приходили в приложение, однако в каждый из будних дней (пн-чт) приходило несколько большее количество новых кользователей по сравнению с концом недели (пт-вс).

Выводы

В ходе исследовательского анализа данных проведены:

- исследование значений столбцов данных в исходных датасетах;
- выделение пользовательских сессий;
- обогащение данных новыми метриками и признаками, а также их исследование.

1. В ходе исследования значений столбцов данных установлено:
   - мы имеем данные о событиях вновь привлечённых пользователей за 28 дней: с 7 октября по 3 ноября 2019 года;
   - всего за этот период привлечены 4293 пользователя;
   - исходные датасеты непротиворечивы;
   - **для источников привлечения:**
     - наиболее популярным источником является `yandex` (1934 пользователя - свыше 45% от общего количества);
     - источник `google` привлёк 1129 пользователей (около 26.3% от общего количества);
     - в целом, выборки количественно сравнимы для статистической значимости при проверке статистических гипотез;
   - **для событий:**
     - наиболее популярными событиями являются: `tips_show` (почти 54% от общего количества событий), `photos_show` (13.49%), `advert_open` (8.31%), `contacts_show` (6.1%), `map` (5.23%), `search_1` (4.73%), `favorites_add` (1.91%);
     - остальные события, в том числе другие виды поиска, составляют менее 1.5% от общего количества каждое;
     - специфические виды поиска, скорее всего, ввиду редкого их использования, не дадут нам хорошей дифференциации при кластеризации, поэтому ***для целей настоящего исследования они были объединены с событием*** `search_1` ***в единое событие*** `search`;
     - пользователи крайне редко кликают на рекомендованные объявления (1.1%) и звонят из приложения (0.73%).

1. Из гистограммы распределения событий по времени следует, что использование приложения за период наблюдений выглядит более-менее равномерным. Тем не менее:
   - наблюдаются провалы в использовании с 9 по 12 октября и 2 ноября;
   - в остальные дни в среднем количество зарегистрированных событий равномерно неубывает;
   - прослеживается четкая временная структура - в ночные часы активность пользователей минимальна, в вечерние - падает.

1. Общее количество зарегистрированных целевых событий проседает каждую неделю на 6-7 день кратно от начала периода наблюдений. Это может свидетельствовать о недельной цикличности покупательской активности пользователей.

1. Количество целевых событий, совершённых во вторую неделю и далее, превышает их количество в первую неделю. Вероятно, это может свидетельствовать о неплохом удержании в приложении: к активным пользователям первой недели добавляются пользователи второй и последующих недель, но большого оттока не происходит.

1. Нечто, похожее на недельную цикличность, можно заметить и для события добавления в избранное, звонков, а также, в меньшей степени, для кликов по рекламным объявлениям.

1. Пользование картой в приложении снижается к концу месяца.

1. Из 4293 пользователей в логе:

   - 2801 увидели рекомендации - ***возможно, рекомендательную систему можно отключить/заблокировать, а может быть, она работает некорректно (обратить внимание продактов и разработчиков)***;
   - 1666 предпочитают искать объявления поиском, 1456 - на карте;
   - 981 посмотрел контакты продавца;
   - при этом только 751 открыл само объявление (***возможна некорректная работа системы логирования действий - по логике приложения кнопка "контакты" находится на странице объявления***);
   - редкие пользователи добавляют объявление в избранное (351) и кликают по рекомендациям (322).
2. ***Общая конверсия пользователей в целевое событие за весь период наблюдений составляет около 23% :***

Вниманию продакта: следует обратить внимание на малое количество событий `favorites_add` и `tips_click`. Возможно, использование этих подсистем приложения неудобно для пользователей.

1. На возможно некорректную работу системы логирования (***вниманию продакта***) событий указывают также следующие факты:
   - существуют пользователи, которые просматривали только фото и контакты (просматривать контакты продавца по логике приложения должно быть возможно только из объявления, но событий `advert_open` для этих пользователей не зарегистрировано);
   - существуют пользователи, которые только просматривали объявления, но при этом не пользовались поиском, и им не было показано ни одной рекомендации;
   - некоторые пользователи кликали на рекомендации - по логике приложения, в этом случае должно было открыться объявление (`advert_open`), однако количество таких событий для этих пользователей равно 0;
   - кроме того, есть по крайней мере один пользователь, 65 раз совершивший целевое событие, но не совершивший ни одного другого события.

1. Диаграммы размаха для распределений событий показывают, что:
   - верхний ус самого частого события (`tips_show`) отсекает частоты выше 31 показа, как выбросы;
   - меньше всего разброс значений для событий `tips_click` и `contacts_call` ;
   - достаточно компактно распределены события `search` , `map` и `favorites_add` ;
   - разброс значений событий `advert_open` и `contacts_show` примерно одинаков;
   - более всего подвержены выбросам события `tips_show` и `photos_show` .
   - в среднем пользователи:
     - видят 8 рекомендованных объявлений, 5 фотографий;
     - по 1 разу кликают на рекомендации и пользуются картой;
     - 2 раза просматривают контакт, звонят продавцу в приложении и добавляют объявление в избранное;
     - 4 раза открывали объявления;
     - 3 раза пользовались поиском;
   - 75% пользователей:
     - не более 3 раз ищут объявления на карте, кликают на рекомендации, или звонят продавцам;
     - не более 4 раз пользуются поиском, или просматривают контакты;
     - не более 9 раз открывают объявления и просматривают фото;
     - не более 5 раз добавляют объявления в избранное;
     - не более 15 раз просматривают рекомендации.

1. Все зарегистрированные события поделены на пользовательские сессии длиной не более 30 минут. Применена сквозная нумерация сессий для всех пользователей. Всего выделено 10368 сессий для 4293 пользователей (в среднем по 2.4 сессии на человека).

1. В процессе обогащения данных добавлены следующие признаки и метрики:
   - среднее количество сессий в день для каждого пользователя;
   - общее количество сессий для пользователей;
   - средняя длина сессии для каждого пользователя (в секундах);
   - среднее количество событий в сессии для каждого пользователя;
   - день недели первого события для каждого пользователя.

1. В ходе анализа вновь добавленных признаков установлено:
   - пользователи приложения заходят в него в среднем от 1 до 6 раз в день (среднее - 1.25), причём половина из них бывает в нём в среднем не чаще 1 раза, а 75% - не чаще 2 раз;
   - общее количество сессий для пользователей варьируется от 1 до 99 (среднее - 2.42), причём половина из них была в приложении не более 1 раза, а 75% - не более 3 раз;
   - средняя длина сессии варьируется от 0 до 9660.5 секунды, при этом половина пользователей непрерывно проводила в приложении в среднем не более 562 секунд, а 75% - не более 1149 секунд;
   - в среднем на каждого пользователя в рамках одной сессии приходится от 1 до 104 событий, для 50% пользователей - не более 6 событий, для 75% - не более 10%;
   - для средней длины сессии выбросами можно считать сессии длиной свыше 2514 секунд;
   - для среднего количества событий в сессии - значения более 19.5;
   - для количества сессий - больше 6;
   - для среднего количества сессий в течении дня - более 2;
   - в целом пользователи равномерно приходили в приложение, однако в каждый из будних дней (пн-чт) приходило несколько большее количество новых кользователей по сравнению с концом недели (пт-вс).

1. В процессе EDA составлена таблица `user_features` , характеризующая поведение пользователей с использованием 14 признаков разного масштаба, которая может быть использована для кластеризации пользователей.

Сегментация пользователей приложения

Рассмотрим и сравним 2 подхода к сегментации пользователей:

- на основе эвристик;
- на основе ML-кластеризации.

Определение функций

In [39]:
```
# определение функции строки для сегментирования
# ======================================================
# на вход подаётся: строка , содержащая поля
#     source - источник привлечения
#     acq_week_day - день привлечения
# на выходе - метка сегмента
# ======================================================
def get_segment(row):
    row_src = row['source']
    row_day = row['acq_week_day']

    if (row_src == 'yandex' and row_day in [0, 1, 2, 3]):
        return 'A'
    elif (row_src == 'yandex' and row_day in [4, 5, 6]):
        return 'B'
    elif (row_src == 'google' and row_day in [0, 1, 2, 3]):
        return 'C'
    elif (row_src == 'google' and row_day in [4, 5, 6]):
        return 'D'
    elif (row_src == 'other' and row_day in [0, 1, 2, 3]):
        return 'E'
    elif (row_src == 'other' and row_day in [4, 5, 6]):
        return 'F'

    raise ValueError(
        "ERROR: Unknown source or weekday ..."
    )
```

In [40]:
```
# определение функции описания сегментов
# ======================================================
# на вход подаются:
#     segments - таблица с количеством человек в сегменте
#     segment_set - обогащённый сегментами лог событий
# на выходе - обогащённая таблица segments
# ======================================================
def describe_segments(segments, segment_set):
    # 1) добавляем количество целевых событий
    segments = (
        segments
        .join(
            (
                segment_set
                .groupby(by=['seg_id', 'event_name'])
                .agg({'user_id':'nunique'})
                .reset_index()
                .query('event_name == "contacts_show"')
                .drop(columns=['event_name'])
                .rename(columns={'user_id':'contacts_show_count'})
                .set_index('seg_id')
            ),
            how='left'
        )
    )

    # 2) считаем конверсию в целевое событие
    segments['cr_pct'] = (
        round(
            segments['contacts_show_count'] * 100 /
            segments['users_total'],
            2
        )
    )

    # 3) добавляем общее количество сессий
    segments = (
        segments
        .join(
            (
                segment_set
                .groupby(by=['seg_id'])
                .agg({'session_id':'nunique'})
                .rename(columns={'session_id':'sessions_total'})
            ),
            how='left'
        )
    )

    # 4) добавим среднюю длительность сессии в секундах
    tmp = ( # определим начало и конец сессий
        segment_set
        .groupby(by=['seg_id', 'session_id'])
        .agg({'event_time':['min', 'max']})
    )
    tmp.columns = ['session_start', 'session_end']
    # вычислим длину в секундах
    tmp['session_len'] = (tmp.session_end - tmp.session_start).astype('timedelta64[s]')
    segments = ( # добавим результат в общую таблицу
        segments
        .join(
            (
                tmp
                .drop(columns=['session_start', 'session_end'])
                .reset_index()
                .groupby(by='seg_id')
                .agg({'session_len':'mean'})
                .rename(columns={'session_len':'mean_session_len'})
                .round(2)
            ),
            how='left'
        )
    )

    # 5) добавим среднее количество сессий в день
    segments = (
        segments
        .join(
            (
```

```
                (
                    segment_set
                    .groupby(by=['seg_id', 'event_date'])
                    .agg({'session_id':'nunique'})
                    .rename(columns={'session_id':'session_count'})
                    .reset_index()
                )
                .groupby(by='seg_id')
                .agg({'session_count':'mean'})
                .round(2)
                .rename(columns={'session_count':'mean_sessions_per_day'})
            ),
            how='left'
        )
    )

    # 6) добавляем среднее количество событий в день (частота действий)
    segments = (
        segments
        .join(
            (
                (
                    segment_set
                    .groupby(by=['seg_id', 'event_date'], as_index=False)
                    .agg({"event_name":'count'})
                    .rename(columns={'event_name':'event_count'})
                )
                .groupby(by='seg_id')
                .agg({'event_count':'mean'})
                .round(2)
                .rename(columns={'event_count':'mean_events_per_day'})
            ),
            how='left'
        )
    )

    # 7) добавим среднее количество событий в сессии
    segments = (
        segments
        .join(
            (
                (
                    segment_set
                    .groupby(by=['seg_id', 'session_id'])
                    .agg({'event_name':'count'})
                    .rename(columns={'event_name':'event_count'})
                    .reset_index()
                )
                .groupby(by='seg_id')
                .agg({'event_count':'mean'})
                .round(2)
                .rename(columns={'event_count':'mean_events_per_session'})
            ),
            how='left'
        )
    )

    return segments
```

In [41]:

```
# определение функции подсчёта удержания Retention Rate
# ==========================================================
# на вход подаются:
#     profiles,              профили пользователей
#     sessions,              данные о сессиях
#     observation_date,      момент анализа
#     horizon_days,          горизонт анализа
#     dimensions=[],         признаки когорт
#     ignore_horizon=False   игнорировать горизонт
# на выходе:
#         result_raw,        "сырые данные"
#         result_grouped,    таблица удержания
#         result_in_time     динамика удержания
# ==========================================================
def get_retention(
    profiles,             # профили пользователей
    sessions,             # данные о сессиях
    observation_date,     # момент анализа
    horizon_days,         # горизонт анализа
    dimensions=[],        # признаки когорт
    ignore_horizon=False  # игнорировать горизонт
):

    dimensions = dimensions

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

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

    # вычисляем лайфтайм для каждой сессии активности в днях
    result_raw['lifetime'] = (
        result_raw['session_start'] - result_raw['first_ts']
    ).dt.days

    # функция для группировки таблицы по желаемым признакам
    def group_by_dimensions(df, dims, horizon_days):
        # строим «треугольную» таблицу удержания (количества активных в лайфтайм)
        result = df.pivot_table(
            index=dims, columns='lifetime', values='user_id', aggfunc='nunique'
        )

        # вычисляем размеры когорт
```

```
        cohort_sizes = (
            df.groupby(dims)
            .agg({'user_id': 'nunique'})
            .rename(columns={'user_id': 'cohort_size'})
        )

        # добавляем размеры когорт в таблицу конверсии
        result = cohort_sizes.merge(result, on=dims, how='left').fillna(0)

        # делим каждую «ячейку» в строке на размер когорты
        # и получаем RetentionRate
        result = result.div(result['cohort_size'], axis=0)

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

        # восстанавливаем размеры когорт
        result['cohort_size'] = cohort_sizes
        return result

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

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

    # возвращаем обе таблицы и сырые данные
    return (
        result_raw,     # "сырые данные"
        result_grouped, # таблица удержания
        result_in_time  # динамика удержания
    )
```

In [42]:
```
# функция для визуализации удержания (Retention Rate)
# ===================================================
# на вход подаются:
#     retention,             таблица удержания
#     horizon,               горизонт анализа
#     figsize                общий размер сетки графиков
# ===================================================
def plot_retention(
    retention,           # таблица удержания
    horizon,
    figsize=(15, 10)     # общий размер сетки графиков
):

    # задаём размер сетки для графиков
    plt.figure(figsize=figsize)

    # исключаем размеры когорт и удержание первого дня
    retention = retention.drop(columns=['cohort_size', 0])

    # строим кривые удержания платящих пользователей
    ax1 = plt.subplot(1, 1, 1)
    retention.T.plot(
        grid=True, ax=ax1
    )
    plt.legend()
    plt.xlabel('Лайфтайм')
    plt.title('Удержание пользователей за {} дней'.format(horizon))

    plt.tight_layout()
    plt.show()
```

In [43]:
```
# функция для визуализации barplot
# ===================================================
# на вход подаётся:
#     df - датафрейм
#     x - имя столбца с категориями
#     y - имя столбца со значениями
#     title - название диаграммы
#     x_title - подпись по оси x
#     y_title - подпись по оси y
# ===================================================
def plot_bar(df, x_col, y_col, title, x_title, y_title):
    fig = px.bar(
        df,
        x=x_col,
        y=y_col,
        text=y_col,
        title=title
    )
    fig.update_layout(
        xaxis_title_text=x_title,
        yaxis_title_text=y_title
    )
    fig.show()
```

Сегментация на основе эвристик

Естественным образом пользователи делятся на 3 примерно равномощные группы по источнику привлечения. Кроме того, в ходе EDA мы установили, что поведение пользователей имеет недельную цикличность, из чего предположили, что, возможно, пользователи, пришедшие в начале (пн-чт) и в конце (пт-вс) недели, отличаются по поведению.

Данные предположения позволяют разделить пользователей на 6 групп:

- пришедшие в начале недели из `yandex` - группа A;
- пришедшие в конце недели из `yandex` - группа B;
- пришедшие в начале недели из `google` - группа C;
- пришедшие в конце недели из `google` - группа D;
- пришедшие в начале недели из `other` - группа E;
- пришедшие в конце недели из `other` - группа F.

Сформируем данные группы и посчитаем их численность.

In [44]:
```
# дополним clean_mobile_dataset данными об источниках и днях привлечения
euristic_segment_set = (
    clean_mobile_dataset
    # источники
    .merge(
        mobile_sources,
        how='left',
        on='user_id'
    )
    # дни привлечения
    .merge(
        (
            user_acquisition.reset_index()
            [['user_id', 'acq_week_day']]
        ),
        how='left',
        on='user_id'
    )
)

euristic_segment_set.sample(10)
```

Out[44]:

	event_time	event_name	user_id	session_id	event_date	source	acq_week_day
15500	2019-10-29 16:07:30.746731	contacts_show	320cab3c-e823-4dff-8c01-c4253764640a	2143	2019-10-29	google	2
71440	2019-10-19 18:29:41.412001	photos_show	f6f94ebe-e69a-4ae3-9fb0-312d52d35826	9974	2019-10-19	other	2
72397	2019-10-22 11:24:56.273510	tips_show	fa7994c2-8e08-486d-b36a-c894bcd4d57b	10138	2019-10-22	yandex	1
23503	2019-10-16 15:58:35.079168	tips_show	52df3193-2eaa-4aaf-a3ce-cee25e2520e6	3335	2019-10-16	other	2
18804	2019-10-13 17:42:04.679054	tips_show	3f172d82-041f-4a58-8e12-4c5e7481d121	2625	2019-10-13	yandex	6
39192	2019-10-29 23:25:54.375546	tips_show	8c356c42-3ba9-4cb6-80b8-3f868d0192c3	5647	2019-10-29	yandex	0
56184	2019-10-23 16:51:06.519159	advert_open	c4667906-ed8a-48a8-adfd-602b25d22244	7909	2019-10-23	other	2
69133	2019-11-01 14:01:55.142756	tips_show	ed58f77d-c951-47eb-a325-ccced526759d	9636	2019-11-01	google	4
21220	2019-10-30 06:38:52.254661	search	490d382a-225c-4240-a7ea-d311fd091f34	3010	2019-10-30	other	6
11368	2019-10-23 13:47:17.896731	tips_show	230b1f7a-17c1-4ee6-b8f3-a996f1a3b7e3	1611	2019-10-23	yandex	2

In [45]:
```
# пропишем каждому пользователю сегмент
try:
    euristic_segment_set['seg_id'] = euristic_segment_set.apply(get_segment, axis=1)
    display(euristic_segment_set.sample(10))
except ValueError as e:
    printnt(e)
```

	event_time	event_name	user_id	session_id	event_date	source	acq_week_day	seg_id
31361	2019-10-21 11:55:02.950558	search	70b57b3c-01b5-4635-b0c5-ed14b94e1359	4503	2019-10-21	google	3	C
13967	2019-10-28 22:35:51.179854	advert_open	2c05bb20-adcf-4cc8-a0d5-48e97235c272	1966	2019-10-28	google	6	D
16862	2019-10-22 21:34:10.790312	tips_show	36191cd7-3f63-4f61-b9e0-3131facc5bc3	2310	2019-10-22	other	1	E
43199	2019-10-29 11:57:10.756635	tips_show	9b835c74-8ede-4586-9f59-e5473aa48de2	6179	2019-10-29	other	0	E
1779	2019-11-01 16:13:36.036453	photos_show	06216934-8394-482e-a9fd-001f93bbebde	220	2019-11-01	google	3	C
16822	2019-10-19 17:14:24.459104	tips_show	3615463b-be22-4167-819c-324affd368a1	2308	2019-10-19	other	0	E
41800	2019-10-27 09:17:42.215811	tips_show	97896555-2af6-4b6b-b23a-2616f8173914	6006	2019-10-27	yandex	6	B
35325	2019-10-13 17:57:00.284298	photos_show	7faaa8f2-b704-45c0-a766-d0ae7df23c43	5116	2019-10-13	google	6	D
1822	2019-10-07 20:16:15.937173	photos_show	063c0b06-6d9d-4580-8d03-f1c8f19ebfa1	225	2019-10-07	other	0	E
69424	2019-10-28 19:08:06.262737	photos_show	eeae6890-380a-44a2-aae6-f7b1fc3adbc8	9679	2019-10-28	other	6	F

In [46]:
```
# посчитаем мощности получившихся сегментов
euristic_segments = (
    euristic_segment_set
    .groupby(by='seg_id')
    .agg({'user_id':'nunique'})
    .rename(columns={'user_id':'users_total'})
)
euristic_segments
```

Out[46]:

seg_id	users_total
A	1158
B	776
C	696
D	433
E	802
F	428

Группы близки по численности. Опишем их, вычислив различные, в том числе, целевые метрики:

In [47]:
```
euristic_segments = describe_segments(euristic_segments, euristic_segment_set)
euristic_segments.T
```

Out[47]:

seg_id	A	B	C	D	E	F
users_total	1158.00	776.00	696.00	433.00	802.00	428.00
contacts_show_count	283.00	195.00	175.00	100.00	147.00	81.00
cr_pct	24.44	25.13	25.14	23.09	18.33	18.93
sessions_total	2778.00	1745.00	1812.00	1016.00	2075.00	942.00
mean_session_len	816.04	886.87	779.41	798.01	628.24	706.77
mean_sessions_per_day	99.64	73.29	65.43	42.42	74.29	39.29
mean_events_per_day	733.39	572.96	474.43	298.38	465.14	268.42
mean_events_per_session	7.39	7.88	7.33	7.05	6.28	6.84

Заказчик просил посчитать для каждого класса следующие метрики:

- retention rate,
- время, проведённое в приложении,
- частота действий,
- конверсия в целевое действие — просмотр контактов.

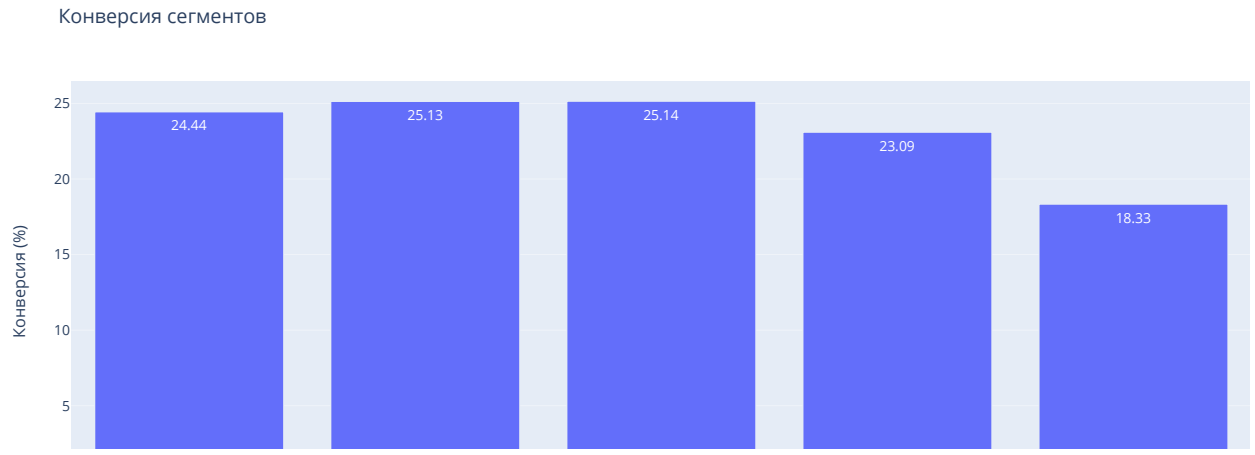
Время, которое пользователи проводят в приложении характеризуется средней длиной сессии (`mean_session_len` , в секундах), частота действий - средним количеством сессий в день (`mean_sessions_per_day`), конверсия в целевое действие в процентах (`cr_pct`) посчитана как отношение количества уникальных пользователей сегмента, совершивших целевое действие, к общему количеству уникальных пользователей сегмента.

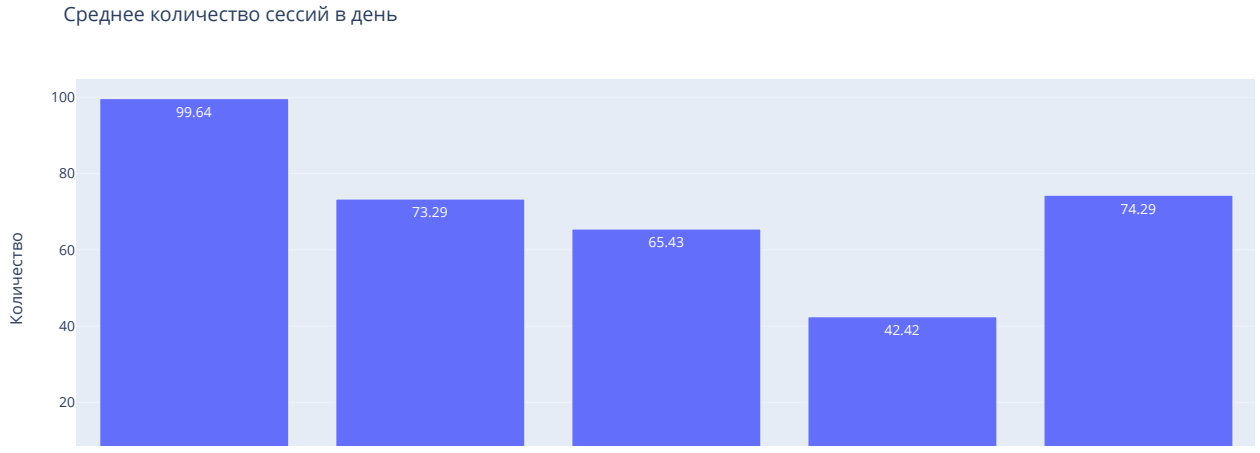
На основе посчитанных метрик можно сделать следующие выводы:

- размеры сегментов сравнимы;
- наилучшую конверсию демонстрируют пользователи групп B и C (аутсайдеры - пользователи групп E и F);
- больше всего заходов в приложение у пользователей групп A и B (причём в разрезе обеих половин недели привлечения), меньше всего - у пользователей из группы F;
- наибольшая средняя длина сессии у пользователей групп A и B, наименьшая - у групп E и F;
- при этом у всех пользователей, пришедших в пятницу и выходные, сессии в среднем длиннее по сравнению с пришедшими с понедельника по четверг;
- по среднему количеству сессий в день лидируют сегменты A и B, а сегменты C и D - в аутсайдерах;
- впрочем, по среднему количеству событий за день сегменты C и D на соответствующих вторых местах после A и B, при этом для всех источников пользователи, пришедшие с понедельника по четверг, совершают больше событий в день;
- по количеству событий в сессии лидируют пользователи из группы A, а вутсайдерах - из группы У.

Визуализируем посчитанные метрики:

In [48]:
```
#  визуализируем конверсию
plot_bar(
    euristic_segments.reset_index(),
    x_col="seg_id", y_col="cr_pct",
    title='Конверсия сегментов',
    x_title='Сегменты',
    y_title='Конверсия (%)'
)

#  визуализируем время, проведённое в приложении
plot_bar(
    euristic_segments.reset_index(),
    x_col="seg_id", y_col="mean_session_len",
    title='Время, проводимое в приложении пользователями сегментов',
    x_title='Сегменты',
    y_title='Средняя длина сессии (с)'
)

#  визуализируем среднее количество сессий в день
plot_bar(
    euristic_segments.reset_index(),
    x_col="seg_id", y_col="mean_sessions_per_day",
    title='Среднее количество сессий в день',
    x_title='Сегменты',
    y_title='Количество'
)
```





Посчитаем последнюю заказанную метрику - retention rate - в разрезе полученных сегментов:

In [49]:
```
# создадим профили пользователей:
#     - seg_id - идентификатор сегмента
#     - user_id - идентификатор пользователя
#     - first_ts - дата и время 1 посещения
user_profiles = (
    euristic_segment_set
    .sort_values(by=['user_id', 'event_time'])
    .groupby(by='user_id', as_index=False)
    .agg({'event_time':'first'})
    .rename(columns={'event_time':'first_ts'})
    .merge(
        (
            euristic_segment_set[['user_id', 'seg_id']]
            .drop_duplicates()
        ),
        on='user_id',
        how='left'
    )
)
user_profiles['dt'] = user_profiles['first_ts'].dt.date
user_profiles.head()
```

Out[49]:

	user_id	first_ts	seg_id	dt
0	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	2019-10-07 13:39:45.989359	E	2019-10-07
1	00157779-810c-4498-9e05-a1e9e3cedf93	2019-10-19 21:34:33.849769	B	2019-10-19
2	00463033-5717-4bf1-91b4-09183923b9df	2019-11-01 13:54:35.385028	B	2019-11-01
3	004690c3-5a84-4bb7-a8af-e0c8f8fca64e	2019-10-18 22:14:05.555052	D	2019-10-18
4	00551e79-152e-4441-9cf7-565d7eb04090	2019-10-25 16:44:41.263364	B	2019-10-25

In [50]:
```
# создадим журнал сессий:
#     - user_id - идентификатор пользователя
#     - session_start - дата и время начала сессии
sessions = (
    euristic_segment_set
    .sort_values(by=['user_id', 'session_id', 'event_time'])
    .groupby(by=['user_id', 'session_id'], as_index=False)
    .agg({'event_time':'first'})
```

```
    .rename(columns={'event_time':'session_start'})
    .drop(columns=['session_id'])
)
sessions.head()
```

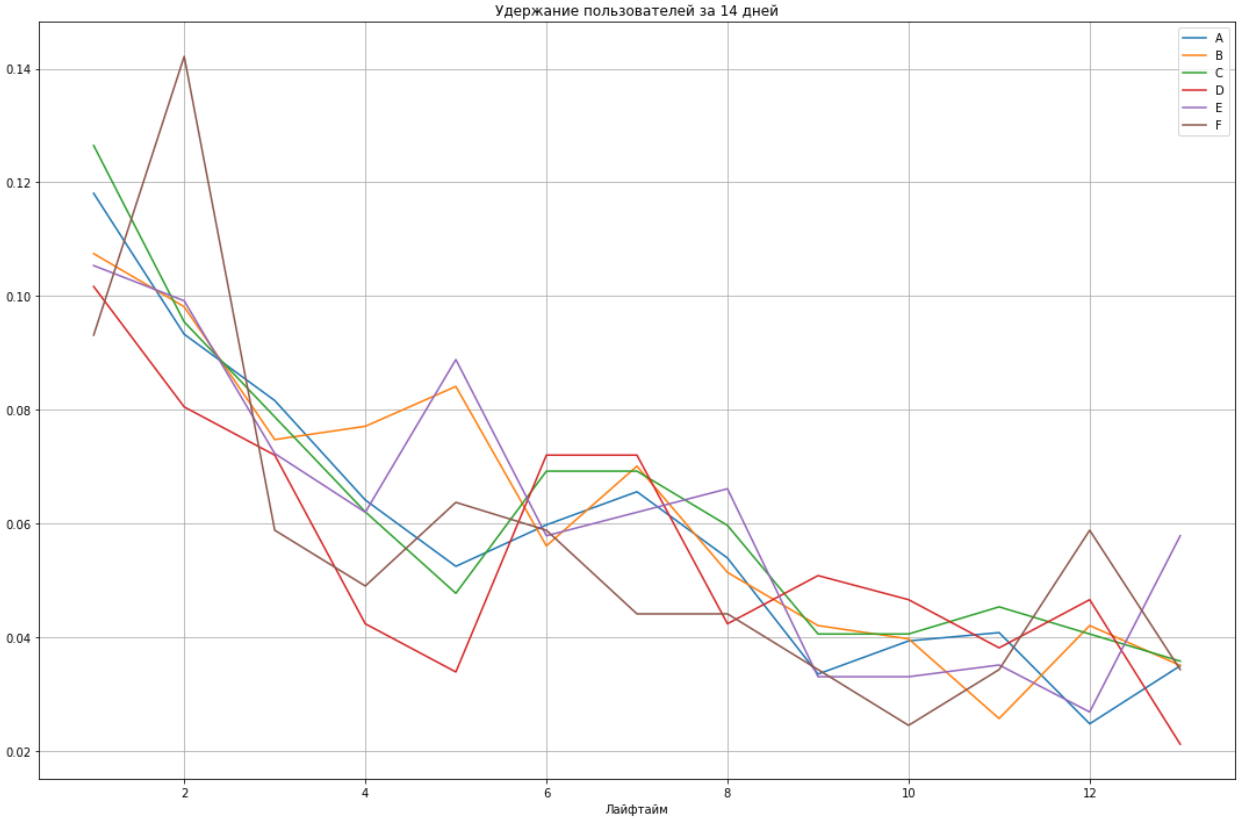
Out[50]:

	user_id	session_start
0	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	2019-10-07 13:39:45.989359
1	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	2019-10-09 18:33:55.577963
2	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	2019-10-21 19:52:30.778932
3	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	2019-10-22 11:18:14.635436
4	00157779-810c-4498-9e05-a1e9e3cedf93	2019-10-19 21:34:33.849769

In [51]:

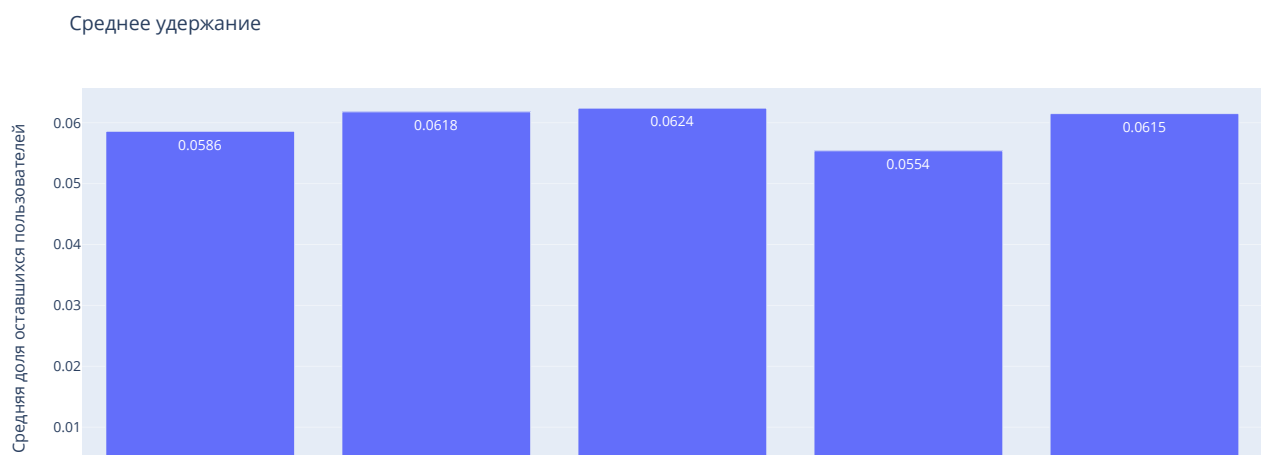
```
# зададим момент и горизонт анализа
observation = euristic_segment_set.event_date.max()
horizon = 14

# посчитаем retention
ret_raw, ret, ret_history = get_retention(
    user_profiles,                          # профили пользователей
    sessions,                               # данные о сессиях
    observation,                            # момент анализа
    horizon,                                # горизонт анализа
    dimensions=['seg_id']                   # признаки когорт
)
display(ret.drop(columns=['cohort_size', 0]))

# визуализируем полученное удержание
plot_retention(
    ret,                # таблица удержания
    horizon,            # горизонт анализа
    figsize=(15, 10)    # общий размер сетки графиков
)

print('Среднее удержание по сегментам:')
display(ret.drop(columns=['cohort_size', 0]).T.mean().round(4))

# визуализируем среднее удержание по сегментам
plot_bar(
    ret.drop(columns=['cohort_size', 0]).T.mean().round(4).reset_index().rename(columns={0:'mean_rr'}),
    x_col="seg_id", y_col="mean_rr",
    title='Среднее удержание',
    x_title='Сегменты',
    y_title='Средняя доля оставшихся пользователей'
)
```

seg_id	1	2	3	4	5	6	7	8	9	10	11	12	13
A	0.118076	0.093294	0.081633	0.064140	0.052478	0.059767	0.065598	0.053936	0.033528	0.039359	0.040816	0.024781	0.034985
B	0.107477	0.098131	0.074766	0.077103	0.084112	0.056075	0.070093	0.051402	0.042056	0.039720	0.025701	0.042056	0.035047
C	0.126492	0.095465	0.078759	0.062053	0.047733	0.069212	0.069212	0.059666	0.040573	0.040573	0.045346	0.040573	0.035800
D	0.101695	0.080508	0.072034	0.042373	0.033898	0.072034	0.072034	0.042373	0.050847	0.046610	0.038136	0.046610	0.021186
E	0.105372	0.099174	0.072314	0.061983	0.088843	0.057851	0.061983	0.066116	0.033058	0.033058	0.035124	0.026860	0.057851
F	0.093137	0.142157	0.058824	0.049020	0.063725	0.058824	0.044118	0.044118	0.034314	0.024510	0.034314	0.058824	0.034314




```
Среднее удержание по сегментам:
```

```
seg_id
A    0.0586
B    0.0618
C    0.0624
D    0.0554
E    0.0615
F    0.0569
dtype: float64
```



Из полученных графиков удержания можно видеть, что:

- удержание всех групп изменяется неравномерно;
- пользователи из группы A равномерно оттекают в течение недели, но возвращаются к началу следующей недели пользования приложением;
- в среднем лучшее удержание на границу недели демонстрирует группа E - к концу первой (на 6 день) и второй (на 14 день) недели больше всего пользователей из этой группы возвращается в приложение;
- в целом сделать вывод о лидерах и аутсайдерах по данной метрике на горизонте 14 дней затруднительно - все группы ведут себя крайне неравномерно, в течение недели падение удержания сменяется ростом и наоборот;
- в среднем для пользователей, пришедших из `yandex` удержание выше для пришедших в выходные, для остальных каналов - в будни.

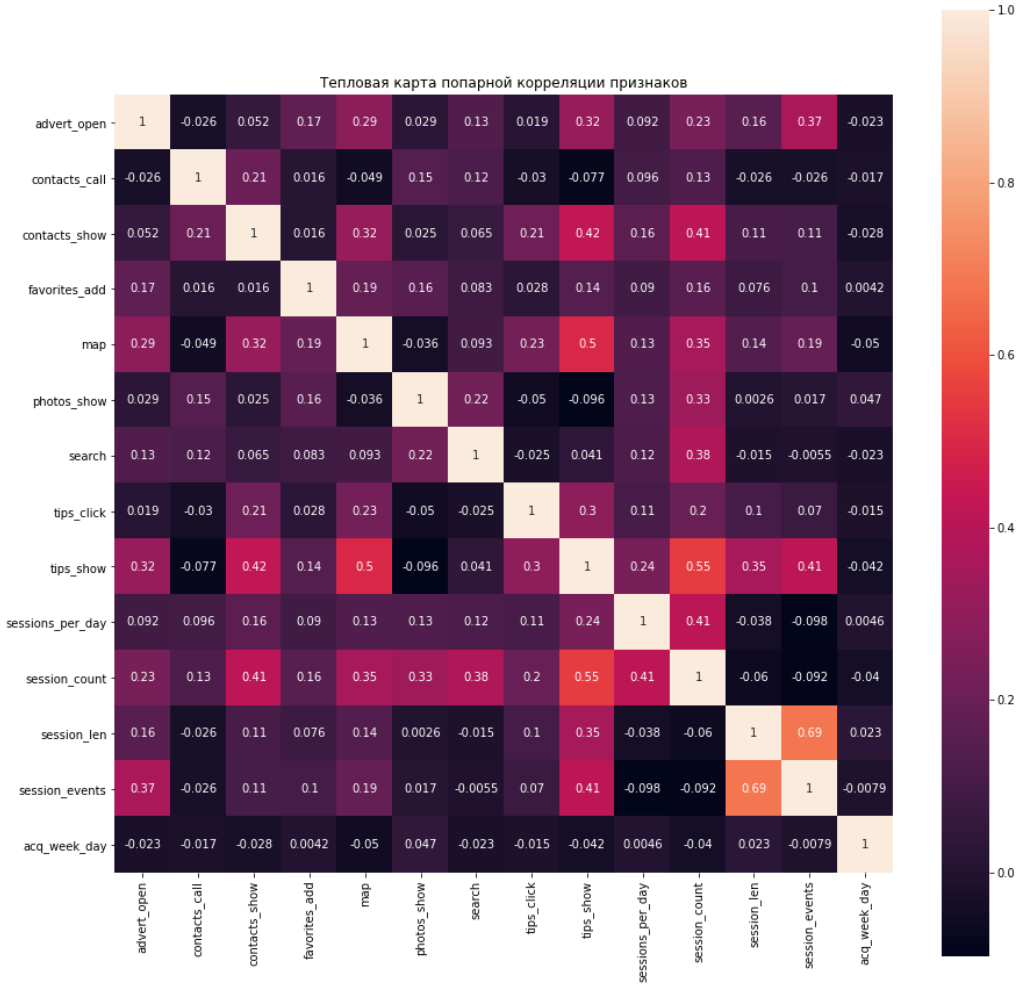
Сегментация на основе кластеризации

В предыдущем разделе мы сегментировали пользователей на основе эвристик. Ключевым условием качества сегментации являлось примерно одинаковый количественный состав классов.

Проверим, сможет ли ML-кластеризация добиться такого результата?

В ходе EDA мы создали матрицу признаков `user_features`, которые характеризуют поведение пользователей и могут быть использованы для кластеризации. Проверим, коррелируют ли они между собой:

In [52]:
```
# рассчитаем  матрицу корреляций
user_features_corr = user_features.corr()

# построим тепловую карту
fig, ax = plt.subplots(figsize=(15, 15))
sns.heatmap(
    user_features_corr, annot=True, square=True
)

ax.set_title('Тепловая карта попарной корреляции признаков')
plt.show()
```



На тепловой карте попарной корреляции мультиколлинеарные признаки не визуализируются. Однако определённая прямая зависимость существует между признаками:

- `session_len` - `session_events`;
- `tips_show` - `session_count`;
- `tips_show` - `map`.

Будем иметь это ввиду при проведении кластеризации.

Стандартизируем данные, построим матрицу расстояний и отрисуем дендрограмму в попытке определить оптимальное количество кластеров:

In [68]:
```
user_features
```

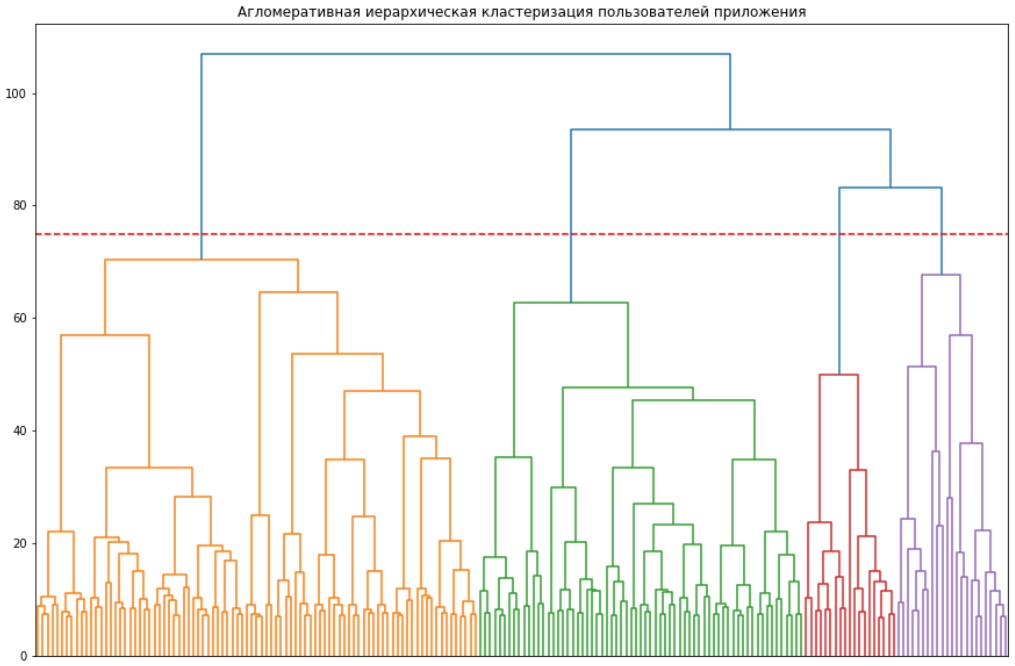
Out[68]:

user_id	advert_open	contacts_call	contacts_show	favorites_add	map	photos_show	search	tips_click	tips_show	sessions_per_day	session_count	session_len	session_events
0001b1d5-b74a-4cbf-aeb0-7df5947bf349	0.0	0.0	0.0	0.0	6.0	0.0	0.0	0.0	29.0	1.000000	4	689.750000	8.750000
00157779-810c-4498-9e05-a1e9e3cedf93	2.0	5.0	11.0	2.0	0.0	33.0	18.0	0.0	0.0	1.000000	6	1961.833333	11.833333
00463033-5717-4bf1-91b4-09183923b9df	0.0	0.0	0.0	0.0	0.0	10.0	0.0	0.0	0.0	1.000000	1	1482.000000	10.000000
004690c3-5a84-4bb7-a8af-e0c8f8fca64e	5.0	0.0	0.0	0.0	6.0	0.0	17.0	0.0	4.0	1.166667	6	1107.000000	5.333333
00551e79-152e-4441-9cf7-565d7eb04090	0.0	3.0	3.0	0.0	0.0	1.0	1.0	0.0	0.0	1.000000	3	186.333333	2.666667
...	...	...	...	...	...	...	...	...	...	...	...	...	...
ffab8d8a-30bb-424a-a3ab-0b63ebbf7b07	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0	15.0	1.000000	2	1482.500000	8.500000
ffc01466-fdb1-4460-ae94-e800f52eb136	0.0	0.0	1.0	0.0	0.0	6.0	0.0	0.0	0.0	1.000000	1	52.000000	7.000000
ffcf50d9-293c-4254-8243-4890b030b238	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.000000	1	80.000000	2.000000
ffe68f10-e48e-470e-be9b-eeb93128ff1a	0.0	0.0	1.0	0.0	0.0	7.0	5.0	0.0	0.0	1.000000	3	777.000000	4.333333
fffb9e79-b927-4dbb-9b48-7fd09b23a62b	0.0	0.0	68.0	0.0	2.0	0.0	0.0	0.0	233.0	1.875000	30	1098.066667	10.100000

4293 rows × 15 columns

In [53]:
```
# стандартизируем данные
X = (
    user_features
)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# построим матрицу расстояний
linked_matrix = linkage(X_scaled, method='ward')
```

In [54]:
```
# отрисуем дендрограмму
fig, ax1 = plt.subplots(figsize=(15, 10))
dendrogram(
    linked_matrix,
    p=200,
    truncate_mode='lastp',
    ax=ax1,
    no_labels=True,
    orientation='top'
)
ax1.hlines(75, 0, 2000, color = 'r', linestyles='--')
plt.title('Агломеративная иерархическая кластеризация пользователей приложения')
plt.show()
```



Метод агломеративной иерархической кластеризации показал, что оптимальное количество классов для заданной матрицы признаков равно 4 (по количеству цветов на дендрограмме). При этом размеры классов, на первый взгляд, соотносятся неплохо.

Кластеризуем пользователей на 4 сегмента с применением метода "k-средних":

In [55]:
```
# разделим пользователей на 4 кластера
km_model = KMeans(n_clusters=4, random_state=0)

# спрогнозируем кластеры
cl_labels = km_model.fit_predict(X_scaled)

# посчитаем метрику силуэта для нашей кластеризации
print('Silhouette_score: {:.2f}'.format(silhouette_score(X_scaled, cl_labels)))
```

```
Silhouette_score: 0.45
```

Значение метрики силуэта 0.45 свидетельствует о достаточно хорошем качестве кластеризации. Добавим полученные метки кластеров в таблицу признаков:

In [56]:
```
# сохраним разбиение на кластеры в исходный датафрейм
user_features['claster_id'] = cl_labels
user_features.sample(5).T
```

Out[56]:

user_id	98761a02-7b98-4ab5-8b08-23fe031fbfa3	8c6c5b2d-826c-4a36-a6e9-ea82a63a2fc6	a07be23d-cfa0-4f30-9772-ad908bdd9b22	8192183e-3d08-48e4-b975-f9f0ede5e919	772cce95-50f6-4409-b8a1-77e8e24d792a
advert_open	0.000000	0.000000	0.00	1.00	0.0
contacts_call	0.000000	0.000000	0.00	0.00	0.0
contacts_show	0.000000	1.000000	0.00	0.00	0.0
favorites_add	0.000000	0.000000	0.00	3.00	0.0
map	0.000000	1.000000	1.00	11.00	0.0
photos_show	0.000000	0.000000	0.00	0.00	0.0
search	0.000000	4.000000	0.00	4.00	4.0
tips_click	0.000000	3.000000	0.00	0.00	0.0
tips_show	12.000000	14.000000	35.00	27.00	5.0
sessions_per_day	1.500000	1.500000	1.00	2.00	1.0
session_count	3.000000	3.000000	4.00	4.00	1.0
session_len	245.666667	958.333333	1105.25	991.25	389.0
session_events	4.000000	7.666667	9.00	11.50	9.0
acq_week_day	1.000000	3.000000	2.00	1.00	2.0
claster_id	0.000000	0.000000	0.00	3.00	0.0

Оценим мощности полученных сегментов:

In [57]:
```
(
    user_features[['claster_id']]
    .reset_index()
    .groupby(by='claster_id')
    .agg('count')
)
```

Out[57]:

claster_id	user_id
0	3643
1	21
2	98
3	531

Мы видим, что численность двух самых маленьких классов составляет менее 10% от численности самого большого. Такие маленькие классы не могут обеспечить статистическую значимость.

Попробуем укрупнить размеры классов, уменьшив их количество:

In [58]:
```
# заново стандартизируем данные
X = (
    user_features
    .drop(columns=['claster_id'])
)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# разделим пользователей на 3 кластера
km_model = KMeans(n_clusters=3, random_state=0)

# спрогнозируем кластеры
cl_labels = km_model.fit_predict(X_scaled)

# посчитаем метрику силуэта для нашей кластеризации
print('Silhouette_score: {:.2f}'.format(silhouette_score(X_scaled, cl_labels)))
```

```
Silhouette_score: 0.43
```

Мы разделили пользователей на 3 сегмента, метрика силуэта несколько уменьшилась. Оценим мощности полученных сегментов:

In [59]:
```
# сохраним разбиение на кластеры в исходный датафрейм
user_features['claster_id'] = cl_labels

(
    user_features[['claster_id']]
    .reset_index()
    .groupby(by='claster_id')
    .agg('count')
)
```

Out[59]:

claster_id	user_id
0	537
1	3735
2	21

И снова самый маленький сегмент составляет по численности менее 1% от численности самого большого.

В ходе данного исследования было проведено несколько десятков экспериментов: удалялись отдельные и группы признаков из матрицы `user_features`, варьировалось количество кластеров и т.д. Однако добиться приближенного к равномерному распределения численности сегментов не удалось.

Причина этого, вероятно, кроется в том, что выбросы и смещённые распределения первичных признаков (количество событий разных типов) повлияли на производные признаки.

Поэтому ML-подход к сегментированию пользователей в рамках данной работы следует признать нецелесообразным.

Выводы

1. Для сегментирования пользователей приложения были использованы 2 подхода:
   - на основе эвристик;
   - на основе ML-кластеризации.

1. ML-подход, по итогу, признан нецелесообразным, поскольку, предположительно, выбросы и смещённые распределения первичных признаков (количество событий разных типов) повлияли на производные признаки, что привело к невозможности обеспечить приближенное к равномерному распределение численности сегментов.

1. На основе эвристического подхода пользователи были разделены на 6 групп сравнимой численности:
   - пришедшие в начале недели из `yandex` - группа A (1158 пользователей);
   - пришедшие в конце недели из `yandex` - группа B (776 пользователей);
   - пришедшие в начале недели из `google` - группа C (696 пользователей);
   - пришедшие в конце недели из `google` - группа D (433 пользователей);
   - пришедшие в начале недели из `other` - группа E (802 пользователей);
   - пришедшие в конце недели из `other` - группа F (428 пользователей).

1. В соответствии с требованиями заказчика для каждой из групп были посчитаны следующие метрики:
   - retention rate - изменение удержания на горизонте 14 суточных лайфтаймов;
   - время, проведённое в приложении - средняя длина сессии (`mean_session_len`, в секундах);
   - частота действий - среднее количество сессий в день (`mean_sessions_per_day`);
   - конверсия в целевое действие (просмотр контактов) - `cr_pct`, посчитана в процентах как отношение количества уникальных пользователей сегмента, совершивших целевое действие, к общему количеству уникальных пользователей сегмента.

1. На основе посчитанных для всех классов метрик сделаны следующие выводы:
   - размеры сегментов сравнимы;
   - наилучшую конверсию демонстрируют пользователи групп B и C (аутсайдеры - пользователи групп E и F);
   - больше всего заходов в приложение у пользователей групп A и B (причём в разрезе обеих половин недели привлечения), меньше всего - у пользователей из группы F;
   - наибольшая средняя длина сессии у пользователей групп A и B, наименьшая - у групп E и F;
   - при этом у всех пользователей, пришедших в пятницу и выходные, сессии в среднем длиннее по сравнению с пришедшими с понедельника по четверг;
   - по среднему количеству сессий в день лидируют сегменты A и B, а сегменты C и D - в аутсайдерах;
   - впрочем, по среднему количеству событий за день сегменты C и D на соответствующих вторых местах после A и B, при этом для всех источников пользователи, пришедшие с понедельника по четверг, совершают больше событий в день;
   - по количеству событий в сессии лидируют пользователи из группы A, а вутсайдерах - из группы У;
   - удержание всех групп изменяется неравномерно;
   - пользователи из группы A равномерно оттекают в течение недели, но возвращаются к началу следующей недели пользования приложением;

- в среднем лучшее удержание на границу недели демонстрирует группа E - к концу первой (на 6 день) и второй (на 14 день) недели больше всего пользователей из этой группы возвращается в приложение;
- в целом сделать вывод о лидерах и аутсайдерах по данной метрике на горизонте 14 дней затруднительно - все группы ведут себя крайне неравномерно, в течение недели падение удержания сменяется ростом и наоборот;
- в среднем для пользователей, пришедших из `yandex` удержание выше для пришедших в выходные, для остальных каналов - в будни.

Проверка гипотез

На заключительном этапе исследования требуется проверить 2 статистических гипотезы:

- гипотеза 1: пользователи, установившие приложение по ссылке из yandex и из google демонстрируют разную конверсию в просмотры контактов;
- гипотеза 2: среднее время сессии для пользователей, пришедших в начале недели (пн-чт) и в конце недели (пт-вс) отличается.

Определение функций

In [60]:
```
# определение функции статистического критерия на равенство долей
# ==========================================================================
def prop_difference_criteria(
    df,          # датафрейм с данными пропорций
    part_col,    # числитель пропорции
    full_col,    # знаменатель пропорции
    alpha=.05    # критический уровень статистической значимости

):
    alpha = alpha

    successes = np.array(df[part_col])
    trials = np.array(df[full_col])

    # пропорция успехов в первой группе:
    p1 = successes[0]/trials[0]
    # пропорция успехов во второй группе:
    p2 = successes[1]/trials[1]
    # пропорция успехов в комбинированном датасете:
    p_combined = (successes[0] + successes[1]) / (trials[0] + trials[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/trials[0] + 1/trials[1]))

    # задаем стандартное нормальное распределение (среднее 0, ст.отклонение 1)
    distr = st.norm(0, 1)

    # считаем вероятность того, что статистика "уехала" от 0 на заданную величину
    # или больше, с использованием кумулятивной функции распределения (для
    # нормального распределения)
    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'
        )
```

In [61]:
```
# определение функции статистического критерия на равенство средних двух
# генеральных совокупностей
# ==========================================================================
def means_equality_criteria(
    sample_1,    # выборка 1 (pd.Series, [])
    sample_2,    # выборка 2 (pd.Series, [])
    equal_var=True, # признак равенства дисперсий
    alpha=.05    # критический уровень статистической значимости
):
    # зададим пороговое значение
    alpha = alpha # если p-value окажется меньше него - отвергнем гипотезу

    # выполним статистический тест на равенство средних
    # двух генеральных совокупностей
    results = st.ttest_ind(
        sample_1,
        sample_2,
        equal_var=equal_var
    )

    print('статистика разности:', results.statistic)
    print('p-значение:', results.pvalue)

    if results.pvalue < alpha:
        print("Отвергаем H0")
    else:
        print("Не получилось отвергнуть H0")
```

Проверка гипотезы 1

Гипотеза 1 (пользователи, установившие приложение по ссылке из yandex и из google демонстрируют разную конверсию в просмотры контактов) представляет собой *статистическую гипотезу о равенстве долей*. Сформулируем основную гипотезу и альтернативу:

- **Основная гипотеза H0**: конверсия в просмотры контактов для пользователей из каналов yandex и google одинакова;
- **Альтернативная гипотеза H1**: конверсия в просмотры контактов для пользователей из каналов yandex и google различается.

Для проверки гипотезы нам надо посчитать общее количество пользователей, которые пришли из выбранных каналов и количество тех из них, которые совершили целевое действие:

In [62]:
```
test_data = (
    # посчитаем пользователей, которые пришли из каналов yandex и google
    mobile_sources
    .groupby(by='source', as_index=False)
    .agg({'user_id':'nunique'})
    .query('source != "other"')
    .rename(columns={'user_id':'user_total'})
    # добавим количество пользователей , совершивших целевое событие
    .merge(
        clean_mobile_dataset
        .merge(
            mobile_sources,
            on='user_id',
            how='left'
        )
        .groupby(by=['source', 'event_name'], as_index=False)
        .agg({'user_id':'nunique'})
        .query('event_name == "contacts_show"')
        .drop(columns=['event_name'])
        .rename(columns={'user_id':'contacts_show_count'}),
        how='left',
        on='source'
    )
)
test_data
```

Out[62]:

	source	user_total	contacts_show_count
0	google	1129	275
1	yandex	1934	478

Проверим гипотезу о равенстве конверсий для выбранных источников:

In [63]:
```
# проверим гипотезу о равенстве конверсий для выбранных источников
prop_difference_criteria(test_data, 'contacts_show_count', 'user_total')
```

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

На имеющихся данных при заданном критическом уровне статистической значимости 0.05 нет оснований считать конверсии конверсии источников `yandex` и `google` различными.

Проверка гипотезы 2

Гипотеза 2 (среднее время сессии для пользователей, пришедших в начале недели - пн-чт - и в конце недели - пт-вс - отличается) представляет собой *статистическую гипотезу о равенстве выборочных средних*. Сформулируем основную гипотезу и альтернативу:

- **Основная гипотеза H0**: среднее время сессии для пользователей, пришедших в начале недели (пн-чт) и в конце недели (пт-вс) одинаково;
- **Альтернативная гипотеза H1**: среднее время сессии для пользователей, пришедших в начале недели (пн-чт) и в конце недели (пт-вс) различается.

Для проверки нам необходимо подготовить две выборки, каждая из которых содержит время, проведённое в приложении сообветствующей группой пользователей. Вся необходимая для расчётов информация (день недели привлечения `acq_week_day`, номер `сессии session_id` и время событий в сессии `event_time`) содержится в сформированной нами ранее таблице `euristic_segment_set`:

In [64]:
```
euristic_segment_set.head(1)
```

Out[64]:

	event_time	event_name	user_id	session_id	event_date	source	acq_week_day	seg_id
0	2019-10-07 13:39:45.989359	tips_show	0001b1d5-b74a-4cbf-aeb0-7df5947bf349	1	2019-10-07	other	0	E

Посчитаем длительности сессий для каждого дня недели привлечения:

In [65]:
```
# определим начало и конец сессий
equality_test_data = (
    euristic_segment_set
    .groupby(by=['acq_week_day', 'session_id'])
    .agg({'event_time':['min', 'max']})
)
equality_test_data.columns = ['session_start', 'session_end']

# вычислим длину в секундах
equality_test_data['session_len'] = (equality_test_data.session_end -
                                     equality_test_data.session_start).astype('timedelta64[s]')
equality_test_data.reset_index(inplace=True)
equality_test_data.sample(5)
```

Out[65]:

	acq_week_day	session_id	session_start	session_end	session_len
7613	4	7567	2019-10-30 13:27:49.941795	2019-10-30 13:27:49.941795	0.0
10258	6	9361	2019-10-20 22:23:03.653329	2019-10-20 22:50:30.057550	1646.0
1888	1	474	2019-10-08 10:23:22.534514	2019-10-08 10:29:01.014798	338.0
9397	6	1949	2019-10-20 21:21:07.817091	2019-10-20 21:21:07.817091	0.0
5873	3	4635	2019-10-16 21:22:33.682548	2019-10-16 21:32:38.933187	605.0

Сравним размеры выборок для каждой из целевых групп пользователей:

In [66]:
```
print(
    'Длина выборки для пользователей, привлечённых в пн-чт:',
    len(equality_test_data.query('acq_week_day in [0, 1, 2, 3]'))
)
print(
    'Длина выборки для пользователей, привлечённых в пт-вс:',
    len(equality_test_data.query('acq_week_day in [4, 5, 6]'))
)
```

```
Длина выборки для пользователей, привлечённых в пн-чт: 6665
Длина выборки для пользователей, привлечённых в пт-вс: 3703
```

Длины выборок достаточны для статистической значимости оценок среднего. Однако мы не можем быть уверены в одинаковости дисперсий выборок.

Для проверки гипотезы о равенстве средней длины сессии для выделенных групп пользователей воспользуемся статистическим тестом `ttest_ind` с параметрами критического уровня статистической значимости `alpha=.05` и признака тавенства дисперсий выборок `equal_var=False`:

In [67]:
```
means_equality_criteria(
    equality_test_data.query('acq_week_day in [0, 1, 2, 3]')['session_len'],
    equality_test_data.query('acq_week_day in [4, 5, 6]')['session_len'],
    equal_var=False
)
```

```
статистика разности: -2.807065013757788
p-значение: 0.005012463203117644
Отвергаем H0
```

На имеющихся данных на 5% уровне значимости имеются основания отвергнуть гипотезу H0 в пользу альтернативы H1. Есть основания считать, что среднее время сессии для пользователей, привлечённых в пн-чт и в пт-вс, различается.

Выводы

Мы проверили две статистических гипотезы:

- гипотеза 1: пользователи, установившие приложение по ссылке из yandex и из google демонстрируют разную конверсию в просмотры контактов;
- гипотеза 2: среднее время сессии для пользователей, пришедших в начале недели (пн-чт) и в конце недели (пт-вс) отличается.

1. Для проверки гипотезы 1 использован критерий равенства долей (`z-test`), для проверки гипотезы 2 - критерий равенства средних двух генеральных совокупностей (`ttest_ind`).

1. На имеющихся данных при заданном критическом уровне статистической значимости 0.05 нет оснований считать конверсии источников `yandex` и `google` различными.

1. На имеющихся данных при заданном критическом уровне статистической значимости 0.05 есть основания считать, что среднее время сессии для пользователей, привлечённых в пн-чт и в пт-вс, различается.

Итоговые выводы исследования и базовые рекомендации для продакт-менеджера

Целями настоящего исследования, заказанного продакт-менеджером приложения "Ненужные вещи" являлись:

- разделение пользователей на несколько различающихся по поведению в приложении групп, на которые можно влиять для управления вовлечённостью;
- получение на основе сегментации гипотез о путях улучшения пользовательского опыта (UX) в приложении.

Для достижения целей требовалось ответить на следующие вопросы продакта:

1. Пользователи какой группы склонны часто возвращаться в мобильное приложение (retention rate)?
2. Пользователи какой группы часто совершают целевое событие (конверсия в целевое событие)?
3. Как различается время, проводимое в приложении, для пользователей разных групп?

В ходе исследования мы провели:

- исследовательский анализ (EDA);
- сегментирование пользователей и подсчёт целевых метрик;
- проверку статистических гипотез о поведении пользователей приложения.

Подробные выводы по каждому этапу приведены в соответствующем разделе отчёта.

Ответим на основные вопросы исследования:

1. Наиболее часто склонны возвращаться в прилодение пользователи, пришедшие из канала `other` в будни с понедельника по четверг, ниже всего в среднем удержание для пользователей, привлечённых из канала `google` с пятницы по воскресенье. Канал `yandex` в среднем выглядит крепким середняком вне зависимости от дня привлечения пользователей. Кроме того, в среднем для пользователей, пришедших из `yandex` удержание выше для пришедших в выходные, для остальных каналов - в будни.

В целом, разница в удержании между группами незначительна - десятые и сотые доли процента.

1. Наилучшую конверсию в целевое событие демонстрируют пользователи, привлечённые из `yandex` в выходные, а также из `google` в будни (аутсайдеры - пользователи из канала `other`).

1. Наибольшее время в приложении проводят пользователи из источника `yandex` , наименьшее - из `other` , пользователи, пришедшие из `google` , занимают среднюю позицию по данной метрике. При этом данные результаты справедливы в разрезе обеих подгрупп по дням недели привлечения.

В то же время, **результаты проверки статистических критериев** показали, что на имеющихся данных при заданном критическом уровне статистической значимости 0.05:

- нет оснований считать конверсии конверсии источников `yandex` и `google` различными.
- есть основания считать, что среднее время сессии для пользователей, привлечённых в пн-чт и в пт-вс, различается.

Базовые рекомендации для продакт-менеджера:

1. Каналы `yandex` и `google` в среднем опережают `other` по конверсии, но отстают по удержанию, при этом имеет место своего рода фактор сезонности в разрезе дней недели привлечения. Возможно, пользователи с более низким удержанием, но более высокой конверсией быстрее находят нужный товар и совершают покупку. Вероятно, поиск известных поисковых гигантов выдаёт более релевантные результаты, кликая на которые пользователи приходят в приложение. В этой связи целесообразно проанализировать поисковые запросы пользователей из разных каналов, пришедших в разное время, и сопоставить их с поисковыми выдачами источников - это, вероятно, поможет выяснить, почему пользователи по-разному конвертируются в приложении.

1. Вместе с тем, в ходе EDA обнаружены недостатки системы логирования событий, или системы выгрузки данных, которые не позволяют провести полноценный анализ воронки пользовательского поведения, а именно:

- из 4293 пользователей в логе:
  - всего 2801 увидели рекомендации (`tips_show`) - возможно, рекомендательную систему можно отключить/заблокировать, а может быть, она работает некорректно;

- 981 посмотрел контакты продавца, при этом только 751 открыл само объявление (по логике приложения кнопка "контакты" должна находиться на странице объявления);
- существуют пользователи, которые просматривали только фото и контакты (просматривать контакты продавца по логике приложения должно быть возможно только из объявления, но событий `advert_open` для этих пользователей не зарегистрировано);
- существуют пользователи, которые только просматривали объявления, но при этом не пользовались поиском, и им не было показано ни одной рекомендации;
- некоторые пользователи кликали на рекомендации - по логике приложения, в этом случае должно было открыться объявление (`advert_open`), однако количество таких событий для этих пользователей равно 0;
- кроме того, есть по крайней мере один пользователь, 65 раз совершивший целевое событие, но не совершивший ни одного другого события.

1. После выяснения причин столь "грязного" логирования событий целесообразно повторить данное исследование на новых данных. Возможно, на основе более чистых логов удастся построить иную, более качественную сегментацию.

Замечание: Также следует отметить, что редкие пользователи добавляют объявление в избранное и кликают по рекомендациям - возможно, использование этих подсистем приложения неудобно для пользователей.

Дополнительные материалы

Подготовка презентации

Презентацию к отчёту можно скачать тут.

Подготовка дашбордов

Дашборд распределения состава событий

Техническое задание на построение дашборда:

1. Постройте диаграмму распределения количества событий по типу события.
2. Добавьте индикатор количества пользователей.
3. Добавьте фильтр дашборда по дате совершения события.

Описание выбранных решений:

Дашборд составлялся по сырым данным - таблица `mobile_dataset.csv` .

1. Для распределения количества событий по их типу выбрана столбчатая гистограмма, поскольку количество событий велико и круговая диаграмма была бы не информативна. Для каждого столбика, соответствующего событию, выведена информация об общем количестве событий данного типа, а также о проценте от общего количества событий.
2. Для индикатора количества пользователей посчитаны уникальные `user.id` в исходной таблице.
3. Для дашборда добавлен единый фильтр по дате и времени события. Фильтр выполнен в виде диапазона значений. При изменении состояния фильтра меняются как гистограмма, так и значение индикатора.

Дашборд распределения состава событий опубликован на сайте Tableau.Public.

Дашборд распределения событий по дням для пользователей из разных источников

Техническое задание на построение дашборда:

1. Постройте диаграмму, отображающую количество событий по дням.
2. Постройте гистограмму, отображающую количество пользователей, пришедших из разных источников.
3. Добавьте фильтр дашборда по типу события.

Описание выбранных решений:

Дашборд составлялся по сырым данным - связке таблиц `mobile_dataset.csv` - `mobile_sources.csv` по принципу "многие-к-одному".

1. Для распределения количества событий по дням выбрана столбчатая состыкованная гистограмма, позволяющая не только оценить общее количество событий на конкретный день, но и доли в нём каждого типа событий.
2. Для количества пользователей из разных источников выбрана круговая диаграмма. Диаграмма наглядна, поскольку источников всего 3.
3. Для дашборда добавлен единый фильтр по типу события. Фильтр выполнен в виде check-list. При изменении состава событий в фильтре меняются как гистограмма, так и значение круговая диаграмма.

Дашборд распределения событий по дням для пользователей из разных источников опубликован на сайте Tableau.Public.