

## Описание проекта

Несмотря на огромные вложения в рекламу, последние несколько месяцев развлекательное приложение Procrastinate Pro+ терпит убытки. Необходимо разобраться в причинах и помочь компании выйти в плюс.

Есть данные о пользователях, привлечённых с 1 мая по 27 октября 2019 года:

- лог сервера с данными об их посещениях,
- выгрузка их покупок за этот период,
- рекламные расходы.

## Описание данных

Структура *visits\_info\_short.csv*:

- User Id — уникальный идентификатор пользователя,
- Region — страна пользователя,
- Device — тип устройства пользователя,
- Channel — идентификатор источника перехода,
- Session Start — дата и время начала сессии,
- Session End — дата и время окончания сессии.

Структура *orders\_info\_short.csv*:

- User Id — уникальный идентификатор пользователя,
- Event Dt — дата и время покупки,
- Revenue — сумма заказа.

Структура *costs\_info\_short.csv*:

- dt — дата проведения рекламной кампании,
- Channel — идентификатор рекламного источника,
- costs — расходы на эту кампанию.

## Цель исследования

Необходимо изучить:

- откуда приходят пользователи и какими устройствами они пользуются,
- сколько стоит привлечение пользователей из различных рекламных каналов;
- сколько денег приносит каждый клиент,
- когда расходы на привлечение клиента окупаются,
- какие факторы мешают привлечению клиентов.

## План работы

### Шаг 1. Загрузить данные и подготовить их к анализу

- Загрузите данные о визитах, заказах и рекламных расходах из CSV-файлов в переменные.
- Изучить данные и выполнить предобработку. Найти пропуски и дубликаты
- Убедиться, что типы данных во всех колонках соответствуют сохранённым в них значениям. Обратить внимание на столбцы с датой и временем.

### Шаг 2. Задать функции для расчёта и анализа LTV, ROI, удержания и конверсии.

Функции для вычисления значений метрик:

- `get_profiles()` — для создания профилей пользователей,
- `get_retention()` — для подсчёта Retention Rate,
- `get_conversion()` — для подсчёта конверсии,
- `get_ltv()` — для подсчёта LTV.

Функции для построения графиков:

- `filter_data()` — для сглаживания данных,
- `plot_retention()` — для построения графика Retention Rate,
- `plot_conversion()` — для построения графика конверсии,
- `plot_ltv_roi` — для визуализации LTV и ROI.

### Шаг 3. Исследовательский анализ данных

- Составить профили пользователей. Определить минимальную и максимальную даты привлечения пользователей.
- Выяснить, из каких стран пользователи приходят в приложение и на какую страну приходится больше всего платящих пользователей. - Построить таблицу, отражающую количество пользователей и долю платящих из каждой страны.
- Узнать, какими устройствами пользуются клиенты и какие устройства предпочитают платящие пользователи. Построить таблицу, отражающую количество пользователей и долю платящих для каждого устройства.
- Изучить рекламные источники привлечения и определить каналы, из которых пришло больше всего платящих пользователей. Построить таблицу, отражающую количество пользователей и долю платящих для каждого канала привлечения.
- После каждого пункта сформулировать выводы.

#### **Шаг 4. Маркетинг**

- Посчитать общую сумму расходов на маркетинг.
- Выяснить, как траты распределены по рекламным источникам, то есть сколько денег потратили на каждый источник.
- Построить визуализацию динамики изменения расходов во времени (по неделям и месяцам) по каждому источнику. Отразить это на одном графике.
- Узнать, сколько в среднем стоило привлечение одного пользователя (CAC) из каждого источника. Использовать профили пользователей.
- Написать промежуточные выводы.

#### **Шаг 5. Оценить окупаемость рекламы**

- Используя графики LTV, ROI и CAC, проанализировать окупаемость рекламы. Считать, что на календаре 1 ноября 2019 года, а в бизнес-плане заложено, что пользователи должны окупаться не позднее чем через две недели после привлечения. Необходимость включения в анализ органических пользователей определить самостоятельно.
- Проанализировать окупаемость рекламы с помощью графиков LTV и ROI, а также графики динамики LTV, CAC и ROI.
- Проверить конверсию пользователей и динамику её изменения. То же самое сделать с удержанием пользователей. Построить и изучить графики конверсии и удержания.
- Проанализировать окупаемость рекламы с разбивкой по устройствам. Построить графики LTV и ROI, а также графики динамики LTV, CAC и ROI.
- Проанализировать окупаемость рекламы с разбивкой по странам. Построить графики LTV и ROI, а также графики динамики LTV, CAC и ROI.
- Проанализировать окупаемость рекламы с разбивкой по рекламным каналам. Построить графики LTV и ROI, а также графики динамики LTV, CAC и ROI.
- Ответить на такие вопросы:
  1. Окупается ли реклама, направленная на привлечение пользователей в целом?
  2. Какие устройства, страны и рекламные каналы могут оказывать негативное влияние на окупаемость рекламы?
  3. Чем могут быть вызваны проблемы окупаемости?
- Написать вывод, описать возможные причины обнаруженных проблем и промежуточные рекомендации для рекламного отдела.

#### **Шаг 6. Написать выводы**

- Выделить причины неэффективности привлечения пользователей.
- Сформулировать рекомендации для отдела маркетинга.

### **1. Загрузка данных и их обработка**

**Импортируем необходимые библиотеки и загружаем данные из файлов в датафреймы:**

Возможно, для отображения графика нужно обновить библиотеку `plotly` (так как используемый в графиках параметр `text_auto=True` появился только в поздних версиях)

```
In [1]: # установка необходимых библиотек  
#!/usr/bin/env python3
```

```
In [2]: import pandas as pd  
from datetime import datetime, timedelta  
  
import numpy as np  
from scipy import stats as st  
  
import seaborn as sns  
import matplotlib.pyplot as plt  
import plotly.express as px  
from tabulate import tabulate  
import warnings  
  
warnings.filterwarnings("ignore")
```

```
In [3]: try:  
    visits = pd.read_csv('C:/Users/Acer/Desktop/анализ бизнес-показателей/датасеты/visits_info_short.csv')  
    orders = pd.read_csv('C:/Users/Acer/Desktop/анализ бизнес-показателей/датасеты/orders_info_short.csv')  
    costs = pd.read_csv('C:/Users/Acer/Desktop/анализ бизнес-показателей/датасеты/costs_info_short.csv')  
except:  
    visits = pd.read_csv('https://code.s3.yandex.net/datasets/visits.csv')  
    orders = pd.read_csv('https://code.s3.yandex.net/datasets/orders_info_short.csv')  
    costs = pd.read_csv('https://code.s3.yandex.net/datasets/costs_info_short.csv')
```

In [4]: # Выведем первые 5 строк датафреймов

```
display(visits.head().style.set_caption('Visits'),  
        orders.head().style.set_caption('Orders'),  
        costs.head().style.set_caption('Costs'))
```

Visits

	User Id	Region	Device	Channel	Session Start	Session End
0	981449118918	United States	iPhone	organic	2019-05-01 02:36:01	2019-05-01 02:45:01
1	278965908054	United States	iPhone	organic	2019-05-01 04:46:31	2019-05-01 04:47:35
2	590706206550	United States	Mac	organic	2019-05-01 14:09:25	2019-05-01 15:32:08
3	326433527971	United States	Android	TipTop	2019-05-01 00:29:59	2019-05-01 00:54:25
4	349773784594	United States	Mac	organic	2019-05-01 03:33:35	2019-05-01 03:57:40

Orders

	User Id	Event Dt	Revenue
0	188246423999	2019-05-01 23:09:52	4.990000
1	174361394180	2019-05-01 12:24:04	4.990000
2	529610067795	2019-05-01 11:34:04	4.990000
3	319939546352	2019-05-01 15:34:40	4.990000
4	366000285810	2019-05-01 13:59:51	4.990000

Costs

	dt	Channel	costs
0	2019-05-01	FaceBoom	113.300000
1	2019-05-02	FaceBoom	78.100000
2	2019-05-03	FaceBoom	85.800000
3	2019-05-04	FaceBoom	136.400000
4	2019-05-05	FaceBoom	122.100000

```
In [5]: # Выведем основные характеристики датафреймов (типы столбцов, пропущенные значения)
print('Посещения')
visits.info()
print()
print('Заказы')
orders.info()
print()
print('Реклама')
costs.info()
```

### Посещения

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 309901 entries, 0 to 309900
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   User Id     309901 non-null   int64  
 1   Region      309901 non-null   object  
 2   Device       309901 non-null   object  
 3   Channel      309901 non-null   object  
 4   Session Start 309901 non-null   object  
 5   Session End   309901 non-null   object  
dtypes: int64(1), object(5)
memory usage: 14.2+ MB
```

### Заказы

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40212 entries, 0 to 40211
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   User Id     40212 non-null   int64  
 1   Event Dt    40212 non-null   object  
 2   Revenue      40212 non-null   float64 
dtypes: float64(1), int64(1), object(1)
memory usage: 942.6+ KB
```

### Реклама

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1800 entries, 0 to 1799
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   dt          1800 non-null   object  
 1   Channel     1800 non-null   object  
 2   costs        1800 non-null   float64 
dtypes: float64(1), object(2)
memory usage: 42.3+ KB
```

Всего в датафрейме `visits` с данными посещений пользователей 309 901 строк, в столбцах пропусков нет. Отметим, что тип данных столбцов `Session Start` и `Session End` принадлежат типу `Object`. В датафрейме `orders` покупок пользователей 40 212 строк, пропусков нет. Столбец `Event Dt` следует привести к типу данных `datetime`. Датафрейм `costs` о рекламных расходах содержит 1 800 строк, пропуски отсутствуют. Столбец `dt` следует привести к типу данных `datetime`.

```
In [6]: # приведем наименования столбцов к нижнему регистру и устраним пробелы в их названии  
visits.columns = visits.columns.str.lower().str.replace(' ', '_')  
orders.columns = orders.columns.str.lower().str.replace(' ', '_')  
costs.columns = costs.columns.str.lower().str.replace(' ', '_')
```

Проверим, присутствуют ли дубликаты в датафрейме:

```
In [7]: print(visits.duplicated().sum(), orders.duplicated().sum(), costs.duplicated().sum())
```

0 0 0

Явных дубликатов нет.

```
In [8]: print('Уникальные значения в столбце стран -', ", ".join(map(str, visits['region'].unique())))
print()
print('Уникальные значения в столбце устройств -', ", ".join(map(str, visits['device'].unique())))
print()
print('Уникальные значения в столбце источников перехода -', ", ".join(map(str, visits['channel'].unique())))
print()
print('Уникальные значения в столбце источников перехода -', ", ".join(map(str, costs['channel'].unique())))
```

Уникальные значения в столбце стран - United States, UK, France, Germany

Уникальные значения в столбце устройств - iPhone, Mac, Android, PC

Уникальные значения в столбце источников перехода - organic, TipTop, RocketSuperAds, YRabbit, FaceBoom, MediaTornado, AdNonSense, LeapBob, WahooNetBanner, OppleCreativeMedia, lambdaMediaAds

Уникальные значения в столбце источников перехода - FaceBoom, MediaTornado, RocketSuperAds, TipTop, YRabbit, AdNonSense, LeapBob, OppleCreativeMedia, WahooNetBanner, lambdaMediaAds

Среди уникальных значений неявных дупликатов не найдено.

```
In [9]: # Приведем столбцы, относящиеся к времени, к типу datetime
visits['session_start'] = pd.to_datetime(visits['session_start'])
visits['session_end'] = pd.to_datetime(visits['session_end'])
orders['event_dt'] = pd.to_datetime(orders['event_dt'])
costs['dt'] = pd.to_datetime(costs['dt']).dt.date
```

**Вывод:**

- пропущенных значений не обнаружено
- столбцы `session_start`, `session_end`, `event_dt` и `dt` приведены к типу `datetime`
- явных и неявных дупликатов не обнаружено
- названия столбцов приведены к нижнему регистру

## 2. Функции для расчёта и анализа LTV, ROI, удержания и конверсии.

Зададим функции для вычисления значений метрик:

`get_profiles()` — для создания профилей пользователей



In [10]: `def get_profiles(sessions, orders, ad_costs):`

```
# находим параметры первых посещений
profiles = (
    sessions.sort_values(by=['user_id', 'session_start'])
    .groupby('user_id')
    .agg(
        {
            'session_start': 'first',
            'channel': 'first',
            'device': 'first',
            'region': 'first',
        }
    )
    .rename(columns={'session_start': 'first_ts'})
    .reset_index()
)

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

# добавляем признак платящих пользователей
profiles['payer'] = profiles['user_id'].isin(orders['user_id'].unique())

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

# объединяем траты на рекламу и число привлечённых пользователей
ad_costs = ad_costs.merge(new_users, on=['dt', 'channel'], how='left')

# делим рекламные расходы на число привлечённых пользователей
ad_costs['acquisition_cost'] = ad_costs['costs'] / ad_costs['unique_users']

# добавляем стоимость привлечения в профили
profiles = profiles.merge(
    ad_costs[['dt', 'channel', 'acquisition_cost']],
    on=['dt', 'channel'],
    how='left',
```

```
)  
  
# стоимость привлечения органических пользователей равна нулю  
profiles['acquisition_cost'] = profiles['acquisition_cost'].fillna(0)  
  
return profiles
```

**get\_retention()** — для подсчёта Retention Rate



In [11]: # функция для расчёта удержания

```
def get_retention(
    profiles,
    sessions,
    observation_date,
    horizon_days,
    dimensions=[],
    ignore_horizon=False,
):
    # добавляем столбец payer в передаваемый dimensions список
    dimensions = ['payer'] + dimensions

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

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

    # функция для группировки таблицы по желаемым признакам
    def group_by_dimensions(df, dims, horizon_days):
        result = df.pivot_table(
            index=dims, columns='lifetime', values='user_id', aggfunc='nunique'
        )
        cohort_sizes = (
            df.groupby(dims)
            .agg({'user_id': 'nunique'})
            .rename(columns={'user_id': 'cohort_size'})
        )
        result = cohort_sizes.merge(result, on=dims, how='left').fillna(0)
        result = result.div(result['cohort_size'], axis=0)
        result = result[['cohort_size'] + list(range(horizon_days))]
        result['cohort_size'] = cohort_sizes
        return result
```

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

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

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

`get_conversion()` — для подсчёта конверсии



In [12]: # функция для расчёта конверсии

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

    # определяем дату и время первой покупки для каждого пользователя
    first_purchases = (
        purchases.sort_values(by=['user_id', 'event_dt'])
        .groupby('user_id')
        .agg({'event_dt': 'first'})
        .reset_index()
    )

    # добавляем данные о покупках в профили
    result_raw = result_raw.merge(
        first_purchases[['user_id', 'event_dt']], on='user_id', how='left'
    )

    # рассчитываем лайфтайм для каждой покупки
    result_raw['lifetime'] = (
        result_raw['event_dt'] - result_raw['first_ts']
    ).dt.days

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

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

```
)  
result = result.fillna(0).cumsum(axis = 1)  
cohort_sizes = (  
    df.groupby(dims)  
    .agg({'user_id': 'nunique'})  
    .rename(columns={'user_id': 'cohort_size'})  
)  
result = cohort_sizes.merge(result, on=dims, how='left').fillna(0)  
# делим каждую «ячейку» в строке на размер когорты  
# и получаем conversion rate  
result = result.div(result['cohort_size'], axis=0)  
result = result[['cohort_size'] + list(range(horizon_days))]  
result['cohort_size'] = cohort_sizes  
return result  
  
# получаем таблицу конверсии  
result_grouped = group_by_dimensions(result_raw, dimensions, horizon_days)  
  
# для таблицы динамики конверсии убираем 'cohort' из dimensions  
if 'cohort' in dimensions:  
    dimensions = []  
  
# получаем таблицу динамики конверсии  
result_in_time = group_by_dimensions(  
    result_raw, dimensions + ['dt'], horizon_days  
)  
  
# возвращаем обе таблицы и сырье данные  
return result_raw, result_grouped, result_in_time
```

`get_ltv()` — для подсчёта LTV



In [13]:

```
def get_ltv(
    profiles,
    purchases,
    observation_date,
    horizon_days,
    dimensions=[],
    ignore_horizon=False,
):
    # исключаем пользователей, не «доживших» до горизонта анализа
    last_suitable_acquisition_date = observation_date
    if not ignore_horizon:
        last_suitable_acquisition_date = observation_date - timedelta(
            days=horizon_days - 1
        )
    result_raw = profiles.query('dt <= @last_suitable_acquisition_date')
    # добавляем данные о покупках в профили
    result_raw = result_raw.merge(
        purchases[['user_id', 'event_dt', 'revenue']], on='user_id', how='left'
    )
    # рассчитываем лайфтайм пользователя для каждой покупки
    result_raw['lifetime'] = (
        result_raw['event_dt'] - result_raw['first_ts']
    ).dt.days
    # группируем по cohort, если в dimensions ничего нет
    if len(dimensions) == 0:
        result_raw['cohort'] = 'All users'
        dimensions = dimensions + ['cohort']

    # функция группировки по желаемым признакам
    def group_by_dimensions(df, dims, horizon_days):
        # строим «треугольную» таблицу выручки
        result = df.pivot_table(
            index=dims, columns='lifetime', values='revenue', aggfunc='sum'
        )
        # находим сумму выручки с накоплением
        result = result.fillna(0).cumsum(axis=1)
        # вычисляем размеры когорт
        cohort_sizes = (
            df.groupby(dims)
            .agg({'user_id': 'nunique'})
            .rename(columns={'user_id': 'cohort_size'})
        )
        # объединяем размеры когорт и таблицу выручки
        result = cohort_sizes.merge(result, on=dims, how='left').fillna(0)
        # считаем LTV: делим каждую «ячейку» в строке на размер когорты
        result['ltv'] = result['revenue'].div(result['cohort_size'])
        return result
    result = group_by_dimensions(result_raw, dimensions, horizon_days)
    # считаем LTV: делим каждую «ячейку» в строке на размер когорты
    result['ltv'] = result['revenue'].div(result['cohort_size'])
```

```
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

# собираем датафрейм с данными пользователей и значениями САС,
# добавляя параметры из dimensions
cac = df[['user_id', 'acquisition_cost'] + dims].drop_duplicates()

# считаем средний САС по параметрам из dimensions
cac = (
    cac.groupby(dims)
    .agg({'acquisition_cost': 'mean'})
    .rename(columns={'acquisition_cost': 'cac'})
)

# считаем ROI: делим LTV на САС
roi = result.div(cac['cac'], axis=0)

# удаляем строки с бесконечным ROI
roi = roi[~roi['cohort_size'].isin([np.inf])]

# восстанавливаем размеры когорт в таблице ROI
roi['cohort_size'] = cohort_sizes

# добавляем САС в таблицу ROI
roi['cac'] = cac['cac']

# в финальной таблице оставляем размеры когорт, САС
# и ROI в лайфтаймы, не превышающие горизонт анализа
roi = roi[['cohort_size', 'cac'] + list(range(horizon_days))]

# возвращаем таблицы LTV и ROI
return result, roi

# получаем таблицы LTV и ROI
result_grouped, roi_grouped = group_by_dimensions(
    result_raw, dimensions, horizon_days
)

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

# получаем таблицы динамики LTV и ROI
```

```
result_in_time, roi_in_time = group_by_dimensions(  
    result_raw, dimensions + ['dt'], horizon_days  
)  
  
return (  
    result_raw, # сырые данные  
    result_grouped, # таблица LTV  
    result_in_time, # таблица динамики LTV  
    roi_grouped, # таблица ROI  
    roi_in_time, # таблица динамики ROI  
)
```

Зададим функции для построения графиков:

#### **filter\_data() — для сглаживания данных**

```
In [14]: # функция для сглаживания фрейма
```

```
def filter_data(df, window):  
    # для каждого столбца применяем скользящее среднее  
    for column in df.columns.values:  
        df[column] = df[column].rolling(window).mean()  
    return df
```

#### **plot\_retention() — для построения графика Retention Rate**



```
In [15]: def plot_retention(retention, retention_history, horizon, window=7):
```

```
    # задаём размер сетки для графиков
    plt.figure(figsize=(15, 10))

    # исключаем размеры когорт и удержание первого дня
    retention = retention.drop(columns=['cohort_size', 0])
    # в таблице динамики оставляем только нужный лайфтайм
    retention_history = retention_history.drop(columns=['cohort_size'])[
        [horizon - 1]
    ]

    # если в индексах таблицы удержания только payer,
    # добавляем второй признак – cohort
    if retention.index.nlevels == 1:
        retention['cohort'] = 'All users'
        retention = retention.reset_index().set_index(['cohort', 'payer'])

    # в таблице графиков – два столбца и две строки, четыре ячейки
    # в первой строим кривые удержания платящих пользователей
    ax1 = plt.subplot(2, 2, 1)
    retention.query('payer == True').droplevel('payer').T.plot(
        grid=True, ax=ax1
    )
    plt.legend()
    plt.xlabel('Лайфтайм')
    plt.title('Удержание платящих пользователей')

    # во второй ячейке строим кривые удержания неплатящих
    # вертикальная ось – от графика из первой ячейки
    ax2 = plt.subplot(2, 2, 2, sharey=ax1)
    retention.query('payer == False').droplevel('payer').T.plot(
        grid=True, ax=ax2
    )
    plt.legend()
    plt.xlabel('Лайфтайм')
    plt.title('Удержание неплатящих пользователей')

    # в третьей ячейке – динамика удержания платящих
    ax3 = plt.subplot(2, 2, 3)
    # получаем названия столбцов для сводной таблицы
    columns = [
        name
        for name in retention_history.index.names
        if name not in ['dt', 'payer']
    ]

```

```
# фильтруем данные и строим график
filtered_data = retention_history.query('payer == True').pivot_table(
    index='dt', columns=columns, values=horizon - 1, aggfunc='mean'
)
filter_data(filtered_data, window).plot(grid=True, ax=ax3)
plt.xlabel('Дата привлечения')
plt.title(
    'Динамика удержания платящих пользователей на {}-й день'.format(
        horizon
    )
)

# в четвертой ячейке – динамика удержания неплатящих
ax4 = plt.subplot(2, 2, 4, sharey=ax3)
# фильтруем данные и строим график
filtered_data = retention_history.query('payer == False').pivot_table(
    index='dt', columns=columns, values=horizon - 1, aggfunc='mean'
)
filter_data(filtered_data, window).plot(grid=True, ax=ax4)
plt.xlabel('Дата привлечения')
plt.title(
    'Динамика удержания неплатящих пользователей на {}-й день'.format(
        horizon
    )
)

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

**plot\_conversion()** — для построения графика конверсии

```
In [16]: def plot_conversion(conversion, conversion_history, horizon, window=7):
```

```
    # задаём размер сетки для графиков
    plt.figure(figsize=(15, 5))

    # исключаем размеры когорт
    conversion = conversion.drop(columns=['cohort_size'])
    # в таблице динамики оставляем только нужный лайфтайм
    conversion_history = conversion_history.drop(columns=['cohort_size'])[horizon - 1]

    # первый график – кривые конверсии
    ax1 = plt.subplot(1, 2, 1)
    conversion.T.plot(grid=True, ax=ax1)
    plt.legend()
    plt.xlabel('Лайфтайм')
    plt.title('Конверсия пользователей')

    # второй график – динамика конверсии
    ax2 = plt.subplot(1, 2, 2, sharey=ax1)
    columns = [
        # столбцами сводной таблицы станут все столбцы индекса, кроме даты
        name for name in conversion_history.index.names if name not in ['dt']]
    filtered_data = conversion_history.pivot_table(index='dt', columns=columns, values=horizon - 1, aggfunc='mean')
    filter_data(filtered_data, window).plot(grid=True, ax=ax2)
    plt.xlabel('Дата привлечения')
    plt.title('Динамика конверсии пользователей на {}-й день'.format(horizon))

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

**plot\_ltv\_roi** — для визуализации LTV и ROI



In [17]: # функция для визуализации LTV и ROI

```
def plot_ltv_roi(ltv, ltv_history, roi, roi_history, horizon, window=7):

    # задаём сетку отрисовки графиков
    plt.figure(figsize=(20, 10))

    # из таблицы ltv исключаем размеры когорт
    ltv = ltv.drop(columns=['cohort_size'])
    # в таблице динамики ltv оставляем только нужный лайфтайм
    ltv_history = ltv_history.drop(columns=['cohort_size'])[[horizon - 1]]

    # стоимость привлечения запишем в отдельный фрейм
    cac_history = roi_history[['cac']]

    # из таблицы roi исключаем размеры когорт и cac
    roi = roi.drop(columns=['cohort_size', 'cac'])
    # в таблице динамики roi оставляем только нужный лайфтайм
    roi_history = roi_history.drop(columns=['cohort_size', 'cac'])[
        [horizon - 1]
    ]

    # первый график – кривые ltv
    ax1 = plt.subplot(2, 3, 1)
    ltv.T.plot(grid=True, ax=ax1)
    plt.legend()
    plt.xlabel('Лайфтайм')
    plt.title('LTV')

    # второй график – динамика ltv
    ax2 = plt.subplot(2, 3, 2, sharey=ax1)
    # столбцами сводной таблицы станут все столбцы индекса, кроме даты
    columns = [name for name in ltv_history.index.names if name not in ['dt']]
    filtered_data = ltv_history.pivot_table(
        index='dt', columns=columns, values=horizon - 1, aggfunc='mean'
    )
    filter_data(filtered_data, window).plot(grid=True, ax=ax2)
    plt.xlabel('Дата привлечения')
    plt.title('Динамика LTV пользователей на {}-й день'.format(horizon))

    # третий график – динамика cac
    ax3 = plt.subplot(2, 3, 3, sharey=ax1)
    # столбцами сводной таблицы станут все столбцы индекса, кроме даты
    columns = [name for name in cac_history.index.names if name not in ['dt']]
    filtered_data = cac_history.pivot_table(
        index='dt', columns=columns, values='cac', aggfunc='mean'
```

```

)
filter_data(filtered_data, window).plot(grid=True, ax=ax3)
plt.xlabel('Дата привлечения')
plt.title('Динамика стоимости привлечения пользователей')

# четвёртый график – кривые roi
ax4 = plt.subplot(2, 3, 4)
roi.T.plot(grid=True, ax=ax4)
plt.axhline(y=1, color='red', linestyle='--', label='Уровень окупаемости')
plt.legend()
plt.xlabel('Лайфтайм')
plt.title('ROI')

# пятый график – динамика roi
ax5 = plt.subplot(2, 3, 5, sharey=ax4)
# столбцами сводной таблицы станут все столбцы индекса, кроме даты
columns = [name for name in roi_history.index.names if name not in ['dt']]
filtered_data = roi_history.pivot_table(
    index='dt', columns=columns, values=horizon - 1, aggfunc='mean'
)
filter_data(filtered_data, window).plot(grid=True, ax=ax5)
plt.axhline(y=1, color='red', linestyle='--', label='Уровень окупаемости')
plt.xlabel('Дата привлечения')
plt.title('Динамика ROI пользователей на {}-й день'.format(horizon))

plt.tight_layout()
plt.show()

```

### 3. Исследовательский анализ данных

- Составим профили пользователей

In [18]: `profiles = get_profiles(visits, orders, costs)`

```
In [19]: # Выведем первые 5 строк получившегося датафрейма  
profiles.head()
```

```
Out[19]:
```

	user_id	first_ts	channel	device	region	dt	month	payer	acquisition_cost
0	599326	2019-05-07 20:58:57	FaceBoom	Mac	United States	2019-05-07	2019-05-07 20:58:57	True	1.088172
1	4919697	2019-07-09 12:46:07	FaceBoom	iPhone	United States	2019-07-09	2019-07-09 12:46:07	False	1.107237
2	6085896	2019-10-01 09:58:33	organic	iPhone	France	2019-10-01	2019-10-01 09:58:33	False	0.000000
3	22593348	2019-08-22 21:35:48	AdNonSense	PC	Germany	2019-08-22	2019-08-22 21:35:48	False	0.988235
4	31989216	2019-10-02 00:07:44	YRabbit	iPhone	United States	2019-10-02	2019-10-02 00:07:44	False	0.230769

```
In [20]: profiles.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
Int64Index: 150008 entries, 0 to 150007  
Data columns (total 9 columns):  
 #   Column           Non-Null Count  Dtype     
---  --  
 0   user_id          150008 non-null   int64    
 1   first_ts         150008 non-null   datetime64[ns]  
 2   channel          150008 non-null   object   
 3   device           150008 non-null   object   
 4   region           150008 non-null   object   
 5   dt               150008 non-null   object   
 6   month            150008 non-null   datetime64[ns]  
 7   payer            150008 non-null   bool     
 8   acquisition_cost 150008 non-null   float64  
dtypes: bool(1), datetime64[ns](2), float64(1), int64(1), object(4)  
memory usage: 10.4+ MB
```

В датафрейме 150 008 уникальных пользователей. Теперь нам известны дата первого посещения (столбец `first_ts` ), рекламный источник привлечения пользователя (столбец `channel` ), тип устройства пользователя (столбец `device` ), страна пользователя (столбец `region` ), является ли пользователь платящим или нет (столбец `payer` ) и стоимость привлечения (столбец `acquisition_cost` )

- Определим минимальную и максимальную даты привлечения пользователей

```
In [21]: # переведем данный столбец в тип date
```

```
profiles['dt'] = pd.to_datetime(profiles['dt']).dt.date
```

В условии задания оговорено, что у нас есть данные о пользователях, привлечённых с 1 мая по 27 октября 2019 года . Проверим, совпадут ли эти даты с найденными минимальной и максимальной датой привлечения в полученном датафрейме *profiles*

```
In [22]: print('Минимальная дата привлечения - ', profiles['dt'].min())
print('Максимальная дата привлечения - ', profiles['dt'].max())
```

Минимальная дата привлечения - 2019-05-01

Максимальная дата привлечения - 2019-10-27

Минимальная и максимальная даты привлечения совпадают с условиями.

**3.2 Выясним, из каких стран пользователи приходят в приложение и на какую страну приходится больше всего платящих пользователей. Построим таблицу, отражающую количество пользователей и долю платящих из каждой страны.**



In [23]: # создадим функцию, отражающую количество пользователей и долю платящих из каждой страны

```
def users(parameter):
    # построим таблицу о количестве пользователей, сгруппированных по признаку
    region = profiles.pivot_table(index=parameter, values='user_id', aggfunc='count') \
        .sort_values(by='user_id') \
        .reset_index()

    # отберем платящих пользователей
    payers = profiles.query('payer == True') \
        .pivot_table(index=parameter, values='user_id', aggfunc='count') \
        .sort_values(by='user_id') \
        .reset_index()

    # объединим две таблицы по общему полю
    overall = region.merge(payers, on=parameter)
    overall.columns = [parameter, 'all_users', 'payers']
    overall['share'] = round(overall['payers'] / overall['all_users'] * 100, 2)
    print(tabulate(overall, headers='keys', tablefmt='psql'))
    print()
    # строим барплот
    px.bar(overall,
           x=['all_users', 'payers'],
           y=parameter,
           text_auto=True,
           barmode='group',
           color_discrete_map={'all_users': '#fb9f3a', 'payers': 'royalblue'}) \
        .update_layout(plot_bgcolor='rgba(0,0,0,0)',
                      autosize=False,
                      title={
                          'text': f"Распределение пользователей по параметру {parameter}",
                          'y':0.98,
                          'x':0.5,
                          'xanchor': 'center',
                          'yanchor': 'top'
                      },
                      xaxis_title="Количество пользователей",
                      yaxis_title="",
                      font=dict(
                          size=12
                      ),
                      width=970,
                      height=700,
                      legend=dict(
                          title='Типы пользователей',
                          yanchor="bottom",
                          y=0.01,
```

```
xanchor="left",
x=0.8
)) \
.update_xaxes(showline=True,
 linewidth=2,
 linecolor='black',
 gridcolor='LightGrey').show()
```

```
In [24]: users('region')
```

	region	all_users	payers	share
0	Germany	14981	616	4.11
1	France	17450	663	3.8
2	UK	17575	700	3.98
3	United States	100002	6902	6.9

Всего представлено четыре страны - Соединенные Штаты, Великобритания, Франция и Германия. Больше всего пользователей наблюдается в Соединенных Штатах - около 100 тысяч, соответственно, и платящих пользоваталей больше именно в этой стране - 6,9% (6 902 пользователя). В Германии, Франции и Великобритании примерно одинаковое количество платящих пользователей - 4.11% (616), 3.8% (663) и 3.98% (700)

соответственно, что значительно меньше, чем в Соединенных Штатах.

**3.3 Узнаем, какими устройствами пользуются клиенты и какие устройства предпочитают платящие пользователи. Построим таблицу, отражающую количество пользователей и долю платящих для каждого устройства.**

In [25]: `users('device')`

	device	all_users	payers	share
0	Mac	30042	1912	6.36
1	PC	30455	1537	5.05
2	Android	35032	2050	5.85
3	iPhone	54479	3382	6.21

Пользователи пользуются такими устройствами, как iPhone, Android, PC и Mac. Наиболее распространенным устройством является iPhone - всего 54 479 пользователей, из них 6.21 % (3 382 пользователя) являются платящими. Самая высокая доля платящих пользователей у устройства Mac - 6.36% (1912 пользователей из 30 042). Чуть меньше доля платящих пользователей у устройств Android - 5.85%(2 050 пользователей из 35 032).

Наименьшая доля у PC - 5.05% - 1 537 пользователей из 30 455

**3.4 Изучим рекламные источники привлечения и определить каналы, из которых пришло больше всего платящих пользователей. Построим таблицу, отражающую количество пользователей и долю платящих для каждого канала привлечения.**

In [26]: `users('channel')`

	channel	all_users	payers	share
0	lambdaMediaAds	2149	225	10.47
1	AdNonSense	3880	440	11.34
2	YRabbit	4312	165	3.83
3	MediaTornado	4364	156	3.57
4	RocketSuperAds	4448	352	7.91
5	LeapBob	8553	262	3.06
6	WahooNetBanner	8553	453	5.3
7	OppleCreativeMedia	8605	233	2.71
8	TipTop	19561	1878	9.6
9	FaceBoom	29144	3557	12.2
10	organic	56439	1160	2.06



```
In [27]: print('Среди каналов привлечения присутствуют следующие:', ', '.join(map(str, profiles['channel'].unique())))
```

Среди каналов привлечения присутствуют следующие: FaceBoom, organic, AdNonSense, YRabbit, MediaTornado, RocketSuperAds, LeapBob, TipTop, WahooNetBanner, OppleCreativeMedia, lambdaMediaAds

Больше всего пользователей привлекли такие каналы, как organic, FaceBoom и TipTop - 56 439, 29 144 и 19 561 пользователей соответственно. Наименьшее количество пользователей перешло через каналы lambdaMediaAds и AdNonSense - 2149 и 3880 пользователей.

Самые высокие доли платящих пользователей наблюдаются у каналов FaceBoom (12.2%), AdNonSense (11.34%) и lambdaMediaAds (10.47%)

### Вывод:

- Получены 150 008 профилей пользователей. Определены минимальная (1 мая 2019) и максимальная (27 октября 2019) даты привлечения, что совпадают с заданными условиями.
- Больше всего платящих пользователей наблюдается в Соединенных Штатах - 6.9% (6 902 пользователя из 100 002). В Германии, Франции и Великобритании примерно одинаковое количество платящих пользователей - 4.11% (616), 3.8% (663) и 3.98% (700) соответственно, что значительно меньше, чем в Соединенных Штатах.
- Наиболее распространенным устройством является iPhone - всего 54 479 пользователя, из них 6.21 % (3 382 пользователя) являются платящими. Самая высокая доля платящих пользователей у устройства Mac - 6.36% (1912 пользователей из 30 042). Чуть меньше доля платящих пользователей у устройств Android - 5.85% (2 050 пользователей из 35 032). Наименьшая доля у PC - 5.05% - 1 537 пользователей из 30 455
- Больше всего пользователей привлекли такие каналы, как organic, FaceBoom и TipTop - 56 439, 29 144 и 19 561 пользователей соответственно. Наименьшее количество пользователей перешло через каналы lambdaMediaAds и AdNonSense - 2149 и 3880 пользователей. Самые высокие доли платящих пользователей наблюдаются у каналов FaceBoom (12.2%), AdNonSense (11.34%) и lambdaMediaAds (10.47%). Отметим, что несмотря на то, что каналы lambdaMediaAds и AdNonSense привлекли наименьшее количество пользователей, именно данные каналы лидируют по доле платящих пользователей.

## 4. Маркетинг

### 4.1 Посчитаем общую сумму расходов на маркетинг.

```
In [28]: # # Выведем первые 5 строк датафрейма с рекламными расходами  
costs.head()
```

Out[28]:

	dt	channel	costs
0	2019-05-01	FaceBoom	113.3
1	2019-05-02	FaceBoom	78.1
2	2019-05-03	FaceBoom	85.8
3	2019-05-04	FaceBoom	136.4
4	2019-05-05	FaceBoom	122.1

```
In [29]: print('Общая сумма расходов на рекламу составляет', round(costs['costs'].sum(), 2))
```

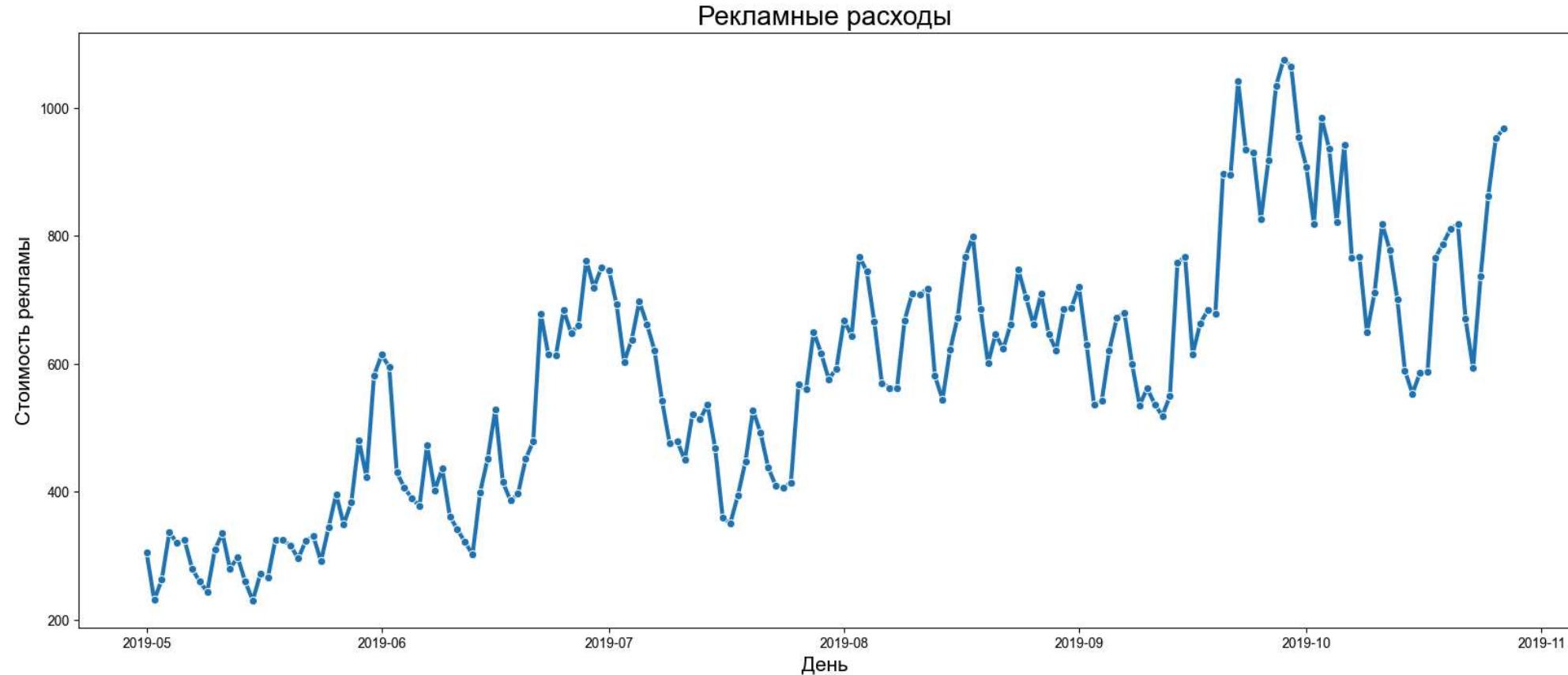
Общая сумма расходов на рекламу составляет 105497.3

```
In [30]: # добавим новые столбцы в датафрейм costs  
costs['day'] = pd.to_datetime(costs['dt']).dt.day  
costs['month'] = pd.to_datetime(costs['dt']).dt.month  
costs['week'] = pd.to_datetime(costs['dt']).dt.week
```

Рассмотрим, как менялась сумма расходов по дням:

```
In [31]: plt.figure(figsize=(20, 8))
sns.lineplot(data=costs.pivot_table(index='dt', values='costs', aggfunc='sum'),
             marker="o",
             linewidth=3,
             legend=False)
sns.set_style("whitegrid")
sns.set_palette('muted')
plt.xlabel('День', fontsize=15)
plt.ylabel("Стоимость рекламы", fontsize=15)
plt.title('Рекламные расходы', fontsize=20)
```

```
Out[31]: Text(0.5, 1.0, 'Рекламные расходы')
```



Полученный график можно назвать циклическим, с мая отмечается рост расходов, к концу мая они достигают максимума, затем расходы уменьшаются. К концу июня, сентября и октября наблюдается та же картина. В середине июня, июля и сентября траты на рекламу заметно снизились.

Рассмотрим, как менялась сумма расходов по месяцам:

```
In [32]: plt.figure(figsize=(20, 8))
px.bar(costs.pivot_table(index='month', values='costs', aggfunc='sum'),
       text_auto=True,
       color_discrete_map={'costs': 'royalblue'}) \
.update_layout(plot_bgcolor='rgba(0,0,0,0)',
               showlegend=False,
               autosize=False,
               title={
                   'text': "Рекламные расходы",
                   'y':0.98,
                   'x':0.5,
                   'xanchor': 'center',
                   'yanchor': 'top'
               },
               xaxis_title="Месяц",
               yaxis_title="Стоимость рекламы",
               font=dict(
                   size=12
               ),
               width=970,
               height=700) \
.update_xaxes(showline=True,
              linewidth=2,
              linecolor='black',
              gridcolor='LightGrey')
```

<Figure size 2000x800 with 0 Axes>

Затраты на рекламу непрерывно растут с мая (9,98 тысяч) по сентябрь (22,44 тысячи), в октябре (20,88 тысяч) траты снизились по сравнению с сентябрем.

Рассмотрим, как менялась сумма расходов по неделям:

```
In [33]: px.bar(costs.pivot_table(index='week', values='costs', aggfunc='sum'),
    text_auto=True,
    x='costs') \
.update_layout(plot_bgcolor='rgba(0,0,0,0)',
    showlegend=False,
    autosize=False,
    title={
        'text': "Рекламные расходы",
        'y':0.98,
        'x':0.5,
        'xanchor': 'center',
        'yanchor': 'top'
    },
    xaxis_title="Стоимость рекламы",
    yaxis_title="Неделя",
    yaxis = dict(
        tickmode = 'linear',
        dtick = 0),

    font=dict(
        size=12
    ),
    width=970,
    height=900) \
.update_xaxes(showline=True,
    linewidth=2,
    linecolor='black',
    gridcolor='LightGrey') \
.update_traces(marker_color='royalblue')
```



Наибольшие вложения в рекламу произошли в 39 и 40 неделю - 6784.58 и 6365.37 соответственно. Графику также присуща цикличность: несколько пиков наблюдаются на 22, 26-27, 39-40 неделях.

#### 4.2 Выясним, как траты распределены по рекламным источникам, то есть сколько денег потратили на каждый источник.

```
In [34]: costs_channel = costs.pivot_table(index='channel', values='costs', aggfunc='sum')
costs_channel['share'] = round(costs_channel['costs'] / sum(costs_channel['costs']) * 100, 2)
costs_channel
```

Out[34]:

channel	costs	share
AdNonSense	3911.25	3.71
FaceBoom	32445.60	30.75
LeapBob	1797.60	1.70
MediaTornado	954.48	0.90
OppleCreativeMedia	2151.25	2.04
RocketSuperAds	1833.00	1.74
TipTop	54751.30	51.90
WahooNetBanner	5151.00	4.88
YRabbit	944.22	0.90
lambdaMediaAds	1557.60	1.48

```
In [35]: px.bar(costs.pivot_table(index='channel', values='costs', aggfunc='sum').sort_values(by='costs'),
    text_auto=True,
    x='costs',
    color_discrete_map={'costs': 'royalblue'}) \
.update_layout(plot_bgcolor='rgba(0,0,0,0)',
    showlegend=False,
    autosize=False,
    title={
        'text': "Рекламные расходы",
        'y':0.98,
        'x':0.5,
        'xanchor': 'center',
        'yanchor': 'top'
    },
    xaxis_title="Стоимость рекламы",
    yaxis_title="Источники привлечения",
    font=dict(
        size=12
    ),
    width=970,
    height=700) \
.update_xaxes(showline=True,
    linewidth=2,
    linecolor='black',
    gridcolor='LightGrey') \
.update_traces(marker_color='royalblue')
```

Более половины расходов на рекламу (51.9%) было потрачено на канал TipTop - 54 751. Около трети рекламных расходов (30.75%) приходится на канал FaceBoom - 32 457. Расходы на остальные каналы менее 5% от общих расходов.

Выясним, как распределяются крупные каналы TipTop, FaceBoom, AdNonSense и WahooNetBanner по странам:

```
In [36]: profiles.query('channel == "TipTop"').value_counts('region')
```

```
Out[36]: region  
United States    19561  
dtype: int64
```

```
In [37]: profiles.query('channel == "FaceBoom"').value_counts('region')
```

```
Out[37]: region  
United States    29144  
dtype: int64
```

```
In [38]: profiles.query('channel == "AdNonSense"').value_counts('region')
```

```
Out[38]: region  
France      1366  
UK          1295  
Germany    1219  
dtype: int64
```

```
In [39]: profiles.query('channel == "WahooNetBanner"').value_counts('region')
```

```
Out[39]: region  
UK          3003  
France     2971  
Germany    2579  
dtype: int64
```

Получается, что каналы TipTop и FaceBoom направлены на привлечение только пользователей из Соединенных Штатов.

\*\*4.3 Построим визуализацию динамики изменения расходов во времени (по неделям и месяцам) по каждому источнику.

Рассмотрим, как изменились расходы по месяцам по каждому источнику:

```
In [40]: costs.pivot_table(index=['dt', 'channel'], values='costs', aggfunc='sum').reset_index()
```

Out[40]:

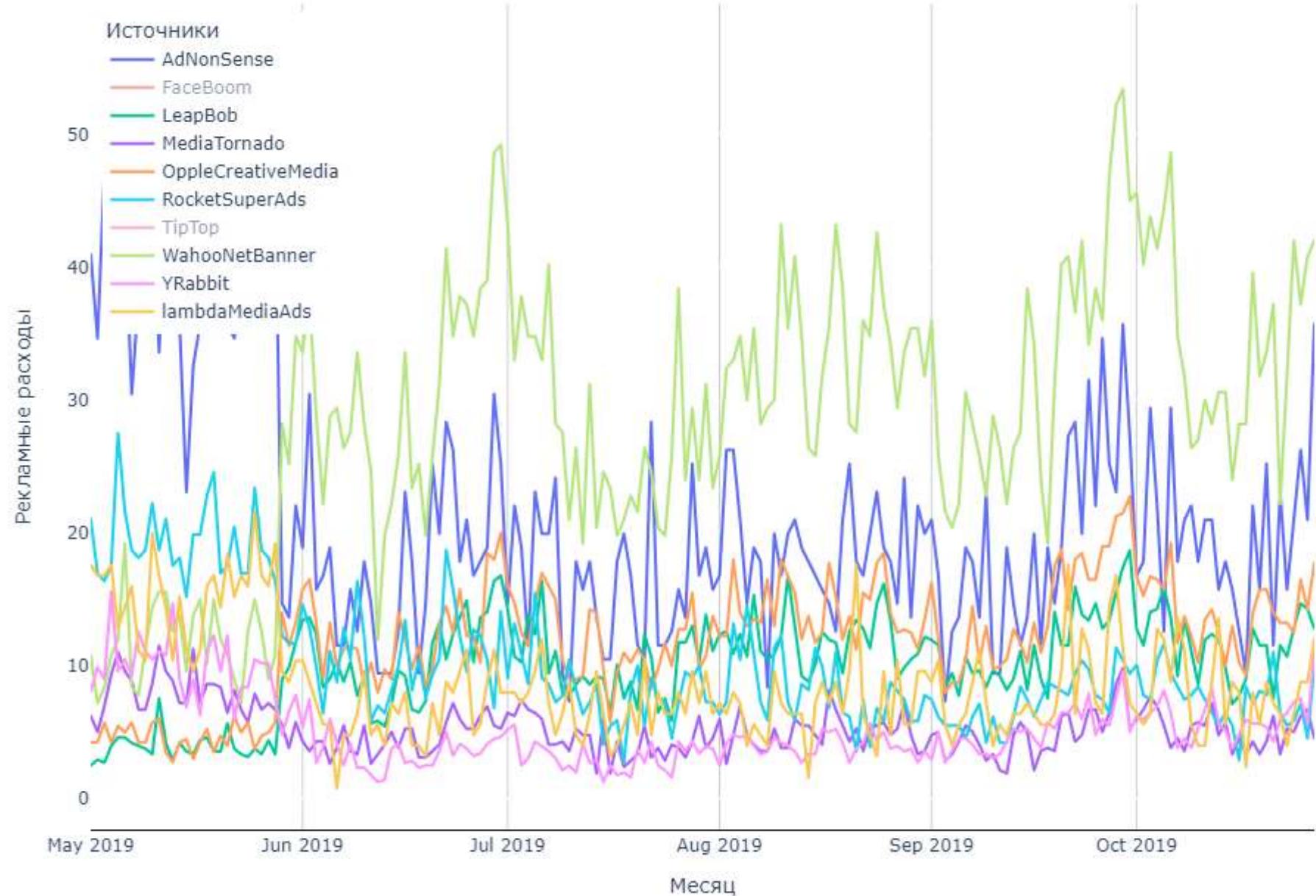
	dt	channel	costs
0	2019-05-01	AdNonSense	40.950
1	2019-05-01	FaceBoom	113.300
2	2019-05-01	LeapBob	2.520
3	2019-05-01	MediaTornado	6.240
4	2019-05-01	OppleCreativeMedia	4.250
...	...	...	...
1795	2019-10-27	RocketSuperAds	10.075
1796	2019-10-27	TipTop	588.000
1797	2019-10-27	WahooNetBanner	42.000
1798	2019-10-27	YRabbit	11.070
1799	2019-10-27	lambdaMediaAds	12.000

1800 rows × 3 columns

```
In [41]: px.line(costs.pivot_table(index=['dt', 'channel'], values='costs', aggfunc='sum').reset_index(),
              x="dt",
              y="costs",
              color="channel") \
    .update_layout(plot_bgcolor='rgba(0,0,0,0)',
                  legend=dict(
                      title='Источники',
                      yanchor="bottom",
                      y=0.7,
                      xanchor="left",
                      x=0.01
                  ),
                  title = {
                      'text': "Динамика изменения рекламных расходов по месяцам",
                      'y':0.95, # new
                      'x':0.5,
                      'xanchor': 'center',
                      'yanchor': 'top' # new
                  },
                  width=990,
                  height=700,
                  xaxis_title="Месяц",
                  yaxis_title="Рекламные расходы",
                  font=dict(
                      size=12
                  )) \
    .update_xaxes(showline=True,
                  linewidth=1,
                  linecolor='black',
                  gridcolor='LightGrey')
```

Сильно выделяются каналы FaceBoom и TipTop, причем на канал TipTop расходы увеличиваются и достигли пика 3 октября 2019 года. Оставим на данном графике все каналы, кроме FaceBoom и TipTop (на графике их можно убрать, нажав один раз на легенду конкретного канала)

## Динамика изменения рекламных расходов по месяцам



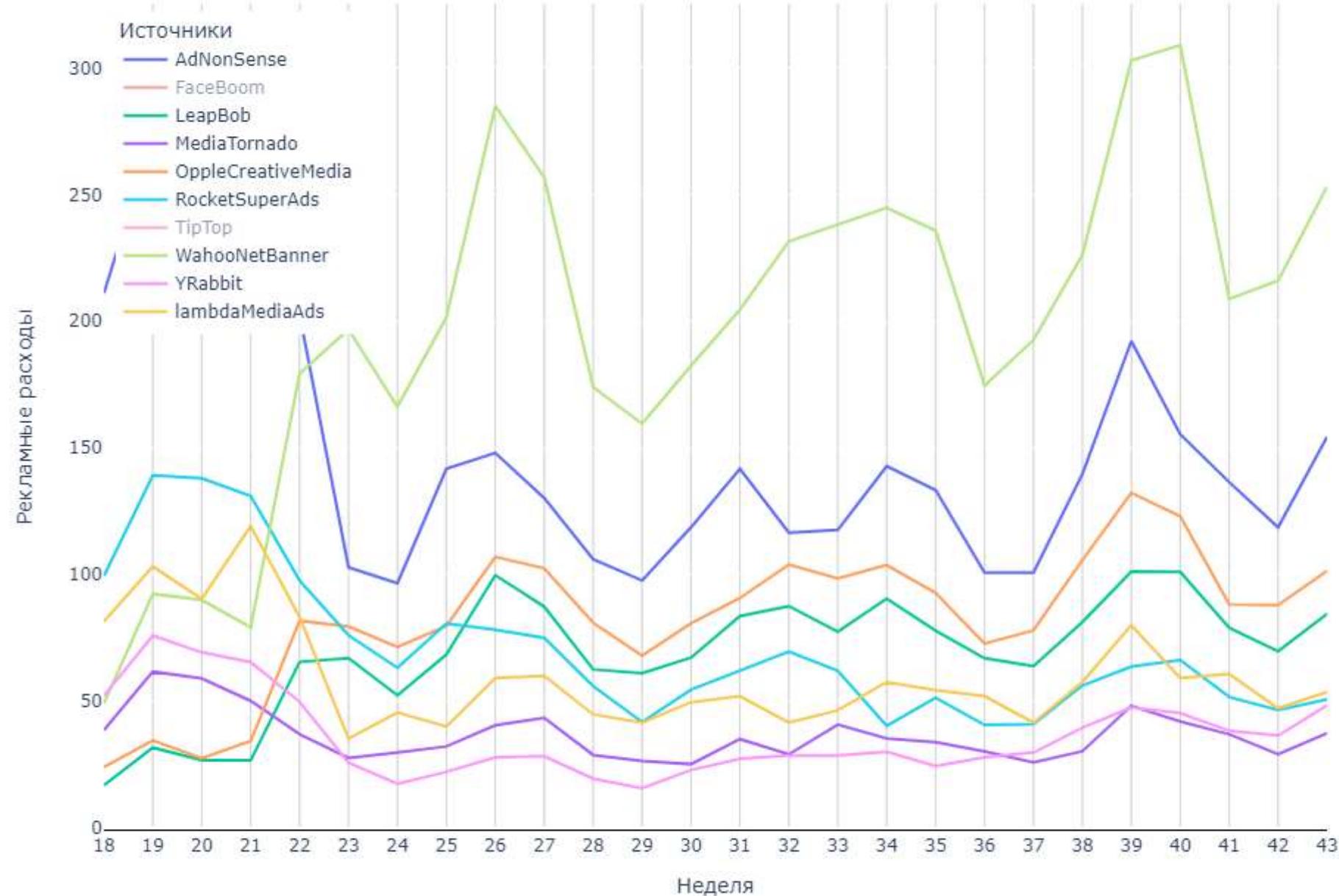
Рост трат можно отметить у канала WahooNetBanner, максимальное значение трат - 29 сентября 2019 (53.4). Затраты на канал AdNonSense с мая упали, затем увеличились к концу сентября.

Рассмотрим, как изменились расходы по неделям по каждому источнику:

```
In [42]: px.line(costs.pivot_table(index=['week', 'channel'], values='costs', aggfunc='sum').reset_index(),
              x="week",
              y="costs",
              color="channel") \
    .update_layout(plot_bgcolor='rgba(0,0,0,0)',
                  legend=dict(
                      title='Источники',
                      yanchor="bottom",
                      y=0.6,
                      xanchor="left",
                      x=0.01
                  ),
                  title = {
                      'text': "Динамика изменения рекламных расходов по неделям",
                      'y':0.95,
                      'x':0.5,
                      'xanchor': 'center',
                      'yanchor': 'top'
                  },
                  width=990,
                  height=700,
                  xaxis_title="Неделя",
                  yaxis_title="Рекламные расходы",
                  xaxis = dict(
                      tickmode = 'linear',
                      dtick = 0),
                  font=dict(
                      size=12
                  )) \
    .update_xaxes(showline=True,
                  linewidth=1,
                  linecolor='black',
                  gridcolor='LightGrey')
```

У каналов FaceBoom и TipTop характер графиков схож, только на канал TipTop выделялось больше средств. Оставим на данном графике все каналы, кроме FaceBoom и TipTop (на графике их можно убрать, нажав один раз на легенду конкретного канала)

## Динамика изменения рекламных расходов по неделям



На остальных каналах наблюдается циклический рост и спад трат на рекламу. С 24 недели графики начинают иметь схожий характер.

\*\*4.4 Узнаем, сколько в среднем стоило привлечение одного пользователя (CAC) из каждого источника.

In [43]: `# рассмотрим численные характеристики датафрейма:  
profiles.describe()`

Out[43]:

	user_id	acquisition_cost
<b>count</b>	1.500080e+05	150008.000000
<b>mean</b>	4.993238e+11	0.703278
<b>std</b>	2.889483e+11	0.954097
<b>min</b>	5.993260e+05	0.000000
<b>25%</b>	2.489249e+11	0.000000
<b>50%</b>	4.977046e+11	0.247500
<b>75%</b>	7.494919e+11	1.108943
<b>max</b>	9.999996e+11	3.715385

Минимальная стоимость привлечения составляет 0. Проверим, в каких каналах она равно нулю:

In [44]: `profiles.query('acquisition_cost == 0')['channel'].unique()`

Out[44]: `array(['organic'], dtype=object)`

Все пользователи с нулевой стоимостью привлечения перешли по органическому трафику. Так как на привлечение таких пользователей не производились расходы, исключим таких пользователей из анализа

```
In [45]: profiles.query('channel != "organic"').describe()
```

Out[45]:

	user_id	acquisition_cost
<b>count</b>	9.356900e+04	93569.000000
<b>mean</b>	4.995116e+11	1.127481
<b>std</b>	2.881600e+11	0.990503
<b>min</b>	5.993260e+05	0.124615
<b>25%</b>	2.506301e+11	0.257143
<b>50%</b>	4.972331e+11	1.080531
<b>75%</b>	7.490889e+11	1.128571
<b>max</b>	9.999799e+11	3.715385

До исключения органических пользователей среднее значение стоимости привлечения составляло 0.7, медианное значение - 0.25. Такая разница в характеристиках обусловлена большим количеством нулевых значений стоимости привлечения органических пользователей. После их исключения среднее значение составляет 1.13, медианное - 1.08.

In [46]: # строим график истории изменений САС по каналам привлечения

```
px.line(profiles.query('channel != "organic"').pivot_table(index=['dt', 'channel'], values='acquisition_cost', aggfunc='mean') \
    .reset_index(),
    x="dt",
    y="acquisition_cost",
    color="channel") \
    .update_layout(plot_bgcolor='rgba(0,0,0,0)',
                  legend=dict(
                      title='Источники',
                      yanchor="bottom",
                      y=0.6,
                      xanchor="left",
                      x=0.01
                  ),
                  title = {
                      'text': "Динамика САС по каналам привлечения",
                      'y':0.95,
                      'x':0.5,
                      'xanchor': 'center',
                      'yanchor': 'top'
                  },
                  width=990,
                  height=700,
                  xaxis_title="Дата привлечения",
                  yaxis_title="",
                  font=dict(
                      size=12
                  )) \
    .update_xaxes(showline=True,
                  linewidth=1,
                  linecolor='black',
                  gridcolor='LightGrey')
```

Динамика САС по каналу TipTop сильно отличается от остальных, наблюдается резкое увеличение стоимости пользователя 19-20 мая 2019 с 0.94 до 1.75, 24-25 июня 2019 с 1.87 до 2.66. На последний день 27 октября 2019 стоимость привлечения составляла в среднем 3.46, что значительно выше

In [47]: # найдем общий САС для каждого источника привлечения

```
profiles.query('channel != "organic"') \
    .pivot_table(index='channel', values='acquisition_cost', aggfunc='mean') \
    .sort_values(by='acquisition_cost')
```

Out[47]:

acquisition\_cost

channel	acquisition_cost
LeapBob	0.210172
MediaTornado	0.218717
YRabbit	0.218975
OppleCreativeMedia	0.250000
RocketSuperAds	0.412095
WahooNetBanner	0.602245
IambdaMediaAds	0.724802
AdNonSense	1.008054
FaceBoom	1.113286
TipTop	2.799003

Привлечение одного пользователя из источника TipTop обошлось в среднем в 2.8, источники AdNonSense и FaceBoom - 1 и 1.11 соответственно. Остальные каналы обходятся в сумму от 0.21 до 0.72.

**Вывод:**

- Рекламные расходы носят циклический характер, с мая отмечается рост расходов, к концу мая они достигают максимума, затем расходы уменьшаются. К концу июня, сентября и октября наблюдается та же картина. В середине июня, июля и сентября траты на рекламу заметно снизились. Наибольшие вложения в рекламу произошли в 39 и 40 неделю - 6784.58 и 6365.37 соответственно. На графике присутствует несколько пиков - на 22, 26-27, 39-40 неделях.
- Более половины расходов на рекламу (51.9%) было потрачено на канал TipTop - 54 751. Около трети рекламных расходов (30.75%) приходится на канал FaceBoom - 32 457. Расходы на остальные каналы менее 5% от общих расходов.
- По динамике изменения расходов во времени сильно выделяются каналы FaceBoom и TipTop, причем на канал TipTop расходы увеличиваются и достигли пика 3 октября 2019 года. Рост затрат отмечается у канала WahooNetBanner (максимальное значение трат - 29 сентября 2019 (53.4)). Затраты на канал AdNonSense с мая упали, затем увеличились к концу сентября. У каналов FaceBoom и TipTop характер графиков схож, только на канал TipTop выделялось больше средств.

- Для анализа стоимости привлечения одного пользователя (CAC) исключены органические пользователи, так как их привлечение не требует затрат. После их исключения среднее значение привлечения составляет 1.13, медианное - 1.08. Динамика CAC по каналу TipTop сильно отличается от остальных, наблюдается резкое увеличение стоимости пользователя 19-20 мая 2019 с 0.94 до 1.75, 24-25 июня 2019 с 1.87 до 2.66. На последний день 27 октября 2019 стоимость привлечения составляла в среднем 3.46, что значительно выше CAC остальных каналов. Привлечение одного пользователя из источника TipTop обошлось в среднем в 2.8, источники AdNonSense и FaceBoom - 1 и 1.11 соответственно. Остальные каналы обходятся в сумму от 0.21 до 0.72.

## 5. Оценка окупаемости рекламы

- Проанализируем окупаемость рекламы с помощью графиков LTV и ROI, а также графики динамики LTV, CAC и ROI.

Исключим органических пользователей. Возьмем за горизонт анализа 14 дней, а момент анализа - 1 ноября 2019

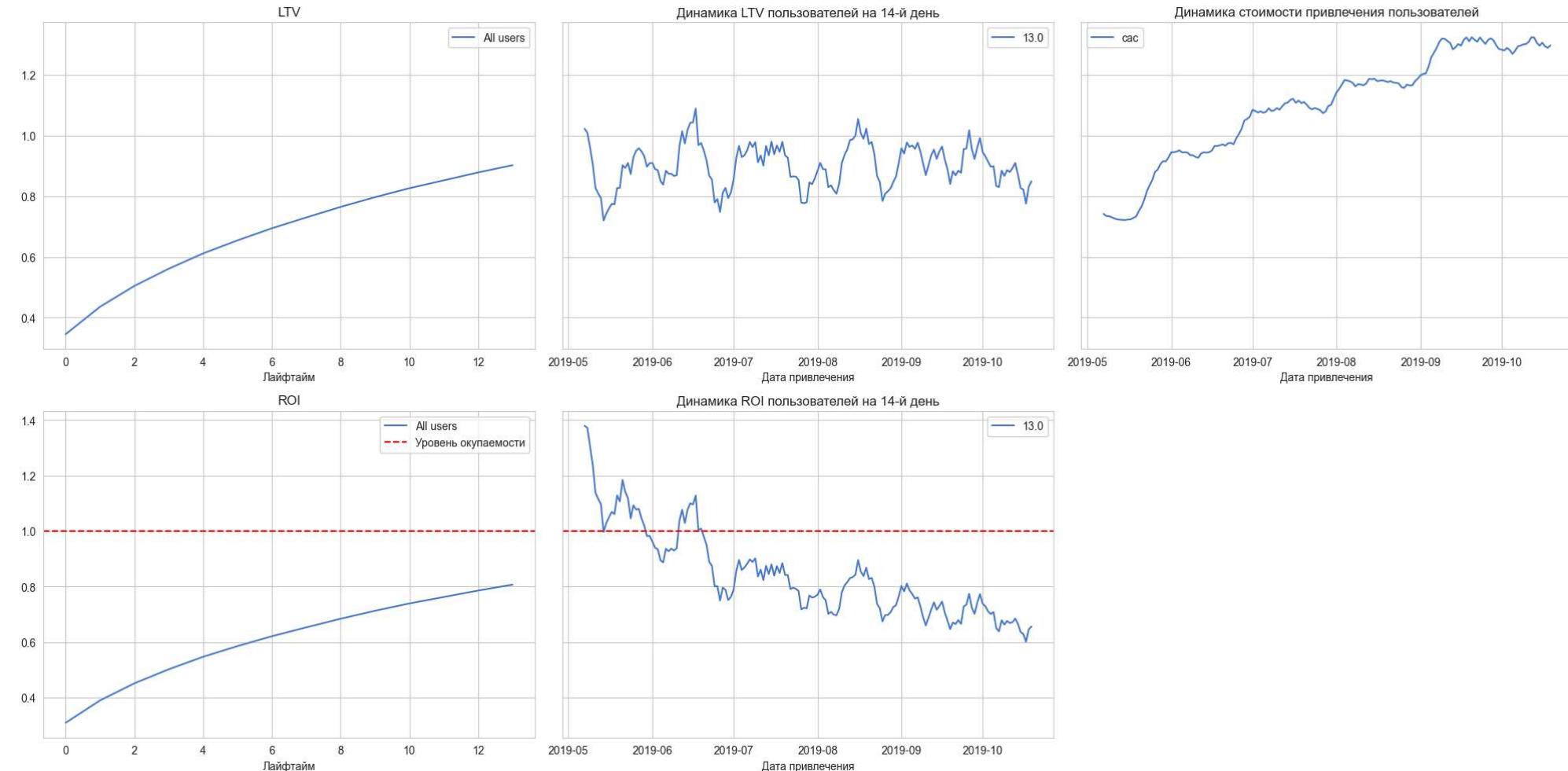
```
In [48]: observation_date = datetime(2019, 11, 1).date() # момент анализа
horizon_days = 14 # горизонт анализа
```

Для начала оценим общую ситуацию — посмотрим на окупаемость рекламы. Рассчитаем и визуализируем LTV и ROI, вызвав функции `get_ltv()` и `plot_ltv_roi()`

In [49]: # считаем LTV и ROI

```
ltv_raw, ltv_grouped, ltv_history, roi_grouped, roi_history = get_ltv(
    profiles.query('channel != "organic"'), orders, observation_date, horizon_days
)

# строим графики
plot_ltv_roi(ltv_grouped, ltv_history, roi_grouped, roi_history, horizon_days)
```



По графикам можно сделать такие выводы:

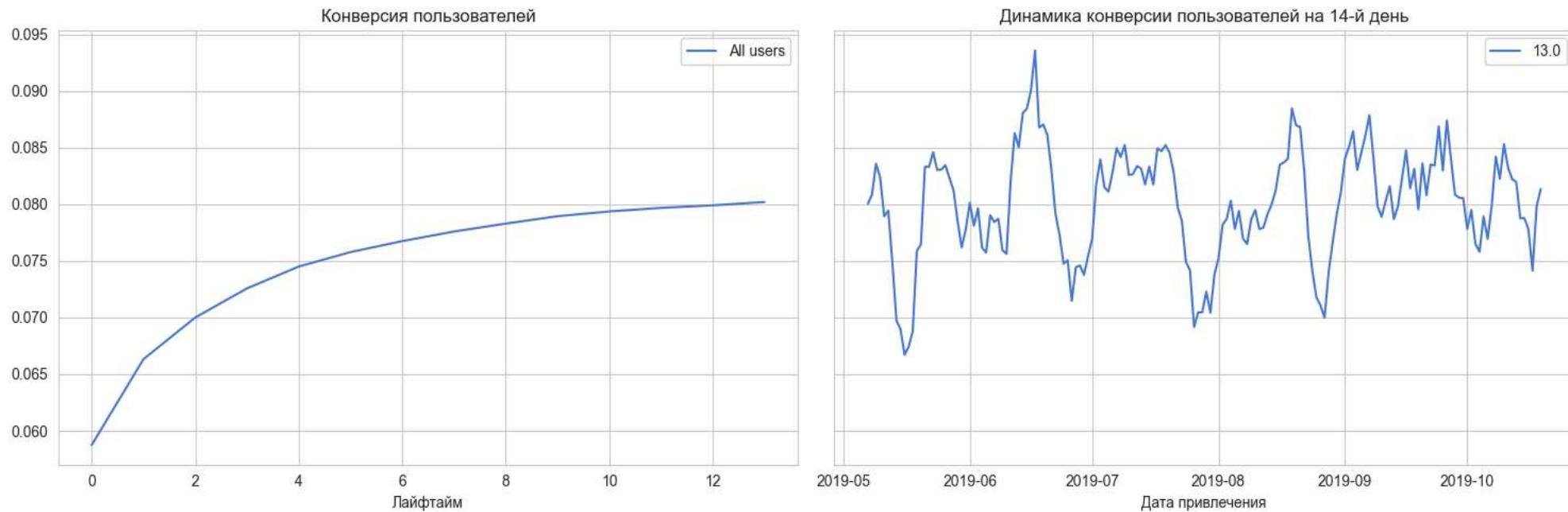
1. Реклама не окупается. ROI в конце недели — чуть выше 80%.
2. САС нестабилен. Значит, дело в увеличении рекламного бюджета. Возможно, это связано с каналом TipTop, расходы на который возрастили со временем.
3. На LTV влияет сезонный фактор, но и этот показатель достаточно стабилен. Значит, дело не в ухудшении качества пользователей.

4. Начиная с середины июня, ROI снижается - реклама перестала окупаться.

- Проверим конверсию пользователей и динамику её изменения. То же самое сделаем с удержанием пользователей. Построим и изучим графики конверсии и удержания.

Посчитаем и визуализируем конверсию, вызвав функции `get_conversion()` и `plot_conversion()`

```
In [50]: conversion_raw, conversion_grouped, conversion_history = get_conversion(  
    profiles.query('channel != "organic"'), orders, observation_date, horizon_days  
)  
  
plot_conversion(conversion_grouped, conversion_history, horizon_days)
```



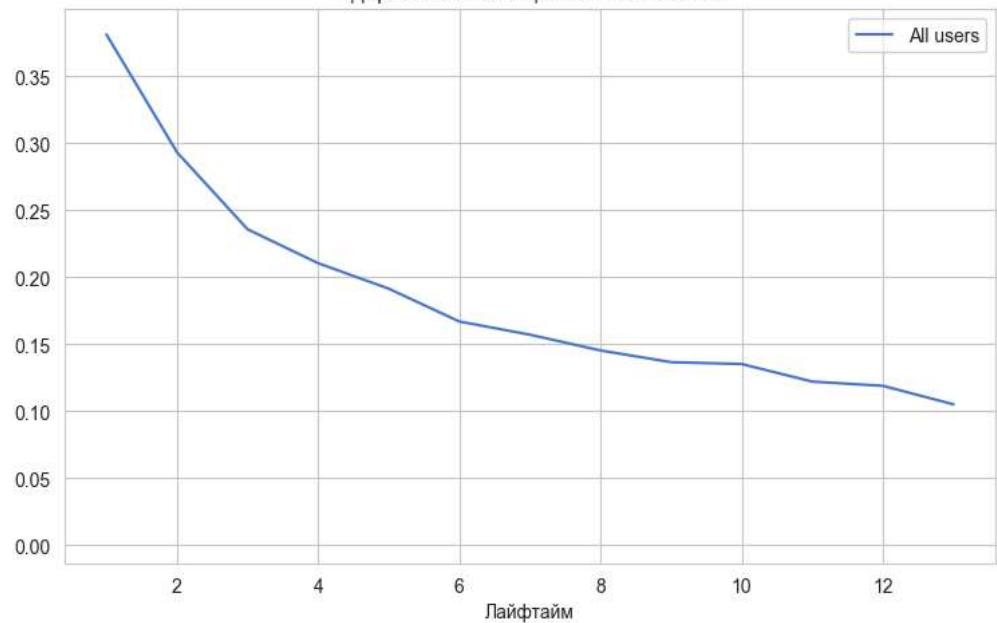
Общий показатель конверсии достаточно низкий.

Вызовем функции `get_retention()` и `plot_retention()`, чтобы рассчитать и отразить на графиках этот показатель.

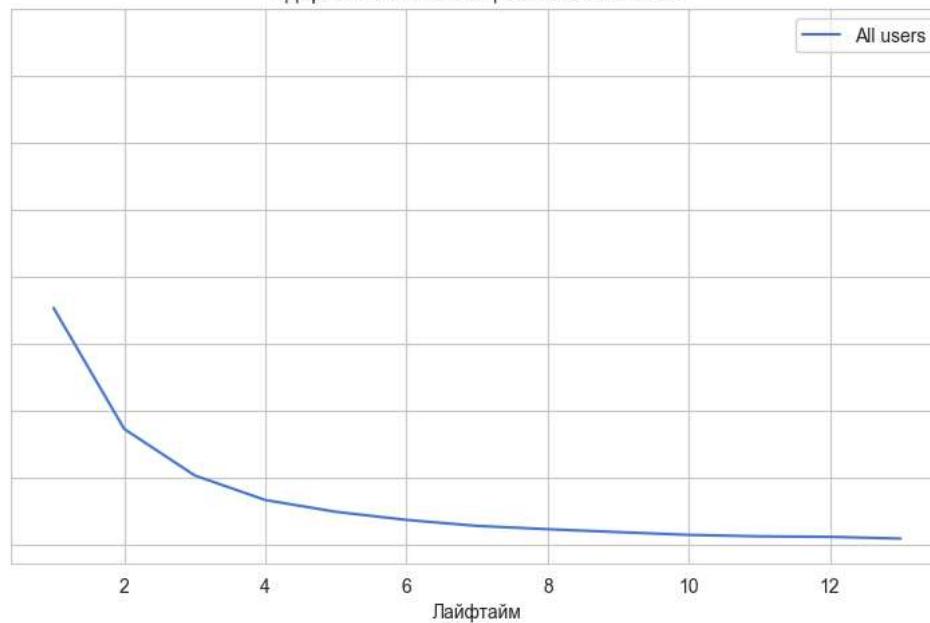
In [51]: # смотрим удержание

```
retention_raw, retention_grouped, retention_history = get_retention(  
    profiles.query('channel != "organic"'), visits, observation_date, horizon_days  
)  
  
plot_retention(retention_grouped, retention_history, horizon_days)
```

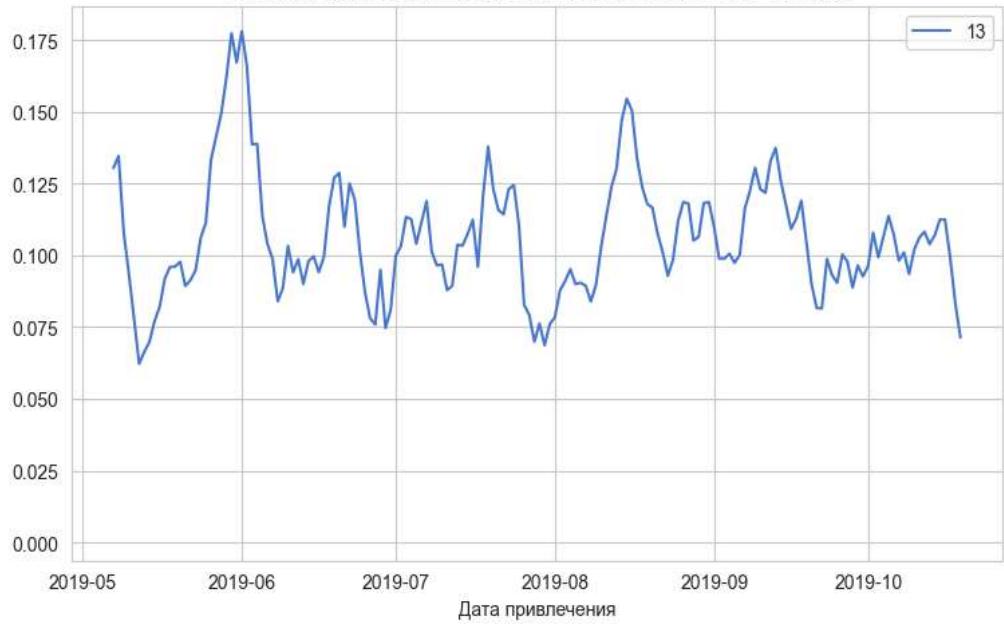
Удержание платящих пользователей



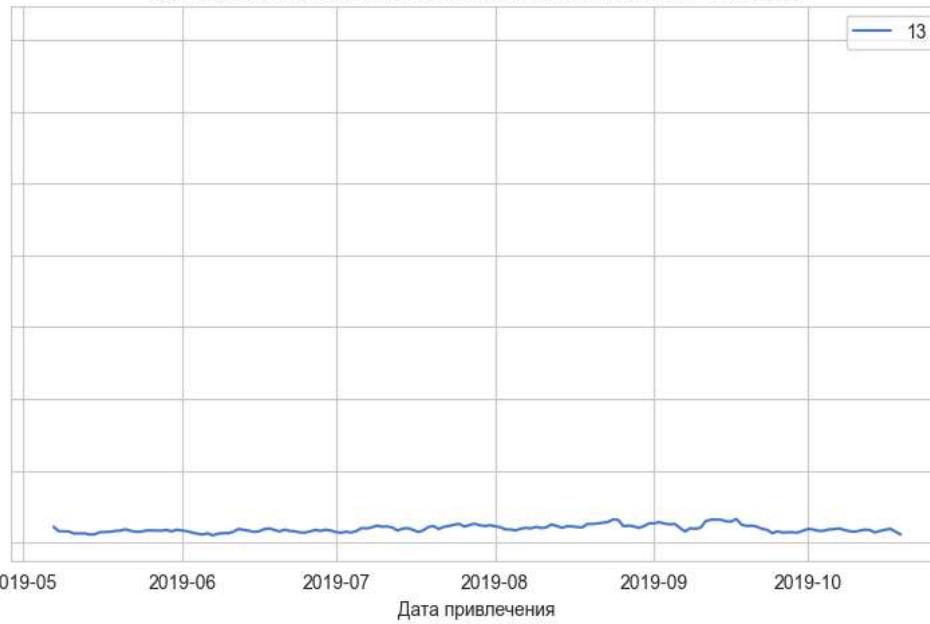
Удержание неплатящих пользователей



Динамика удержания платящих пользователей на 14-й день



Динамика удержания неплатящих пользователей на 14-й день



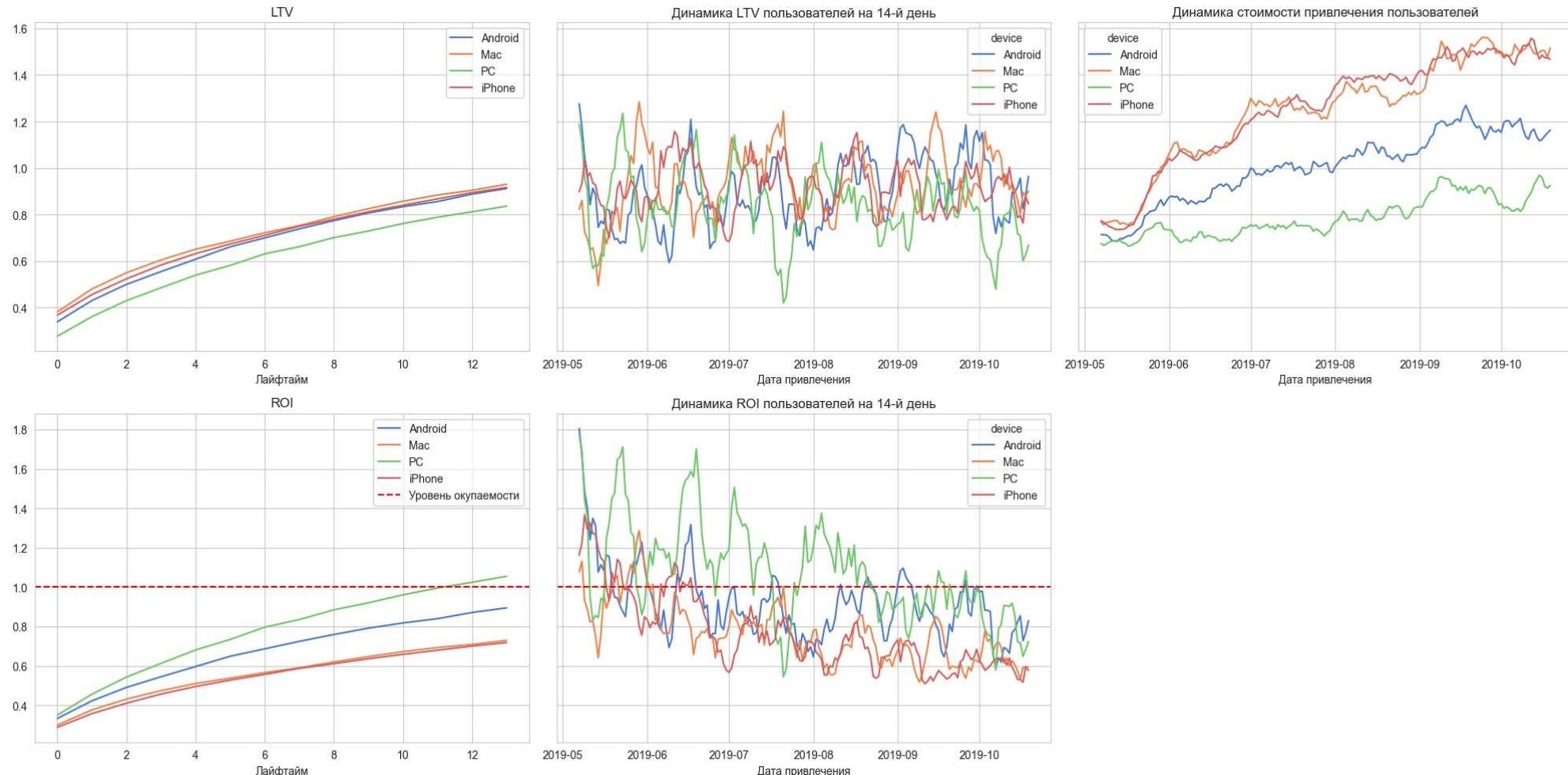
Удержание платящих пользователей выше неплатящих.

- Проанализируем окупаемость рекламы с разбивкой по устройствам. Построим графики LTV и ROI, а также графики динамики LTV, САС и ROI.

In [52]: # считаем LTV и ROI

```
ltv_raw, ltv_grouped, ltv_history, roi_grouped, roi_history = get_ltv(
    profiles.query('channel != "organic"'), orders, observation_date, horizon_days, dimensions=['device']
)

# строим графики
plot_ltv_roi(ltv_grouped, ltv_history, roi_grouped, roi_history, horizon_days)
```

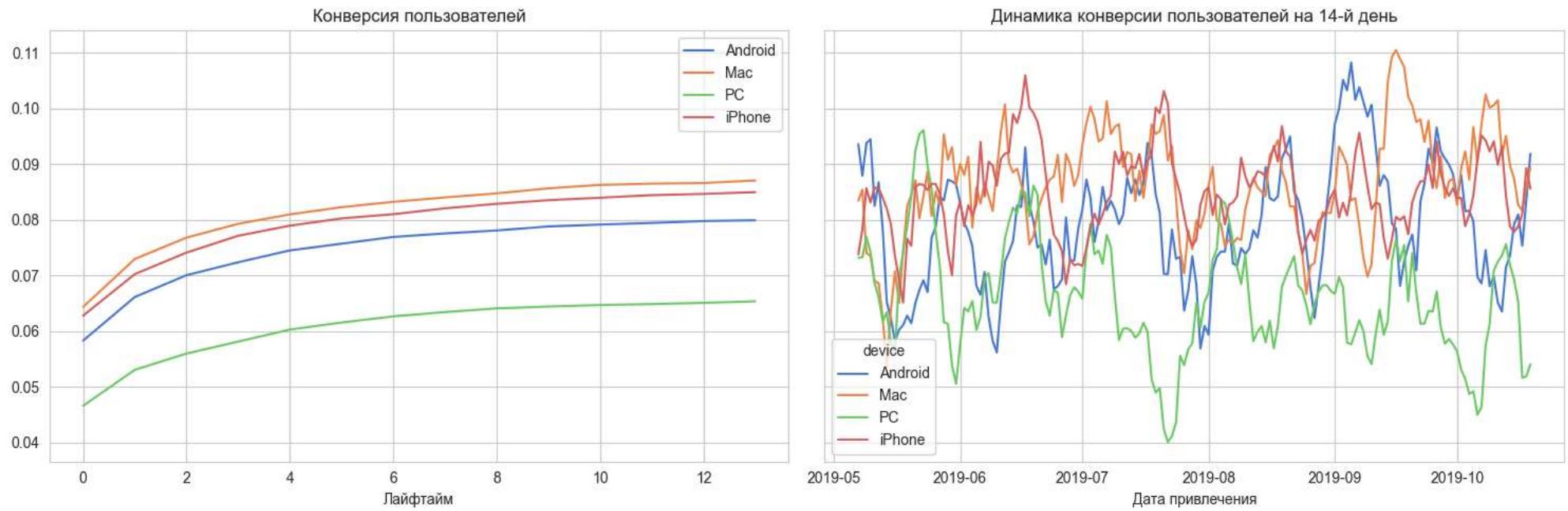


Стоимость привлечения увеличивается по времени для всех устройств, наибольшая стоимость - у устройств Mac и iPhone, наименьшая - PC. Судя по динамике ROI пользователей на 14-й день, к концу августа реклама перестала окупаться на всех устройствах. Пользователи с Mac и iPhone не оплатили рекламные расходы, именно на эти устройства приходится наибольшая стоимость привлечения.

Посчитаем и визуализируем конверсию, вызвав функции `get_conversion()` и `plot_conversion()`

```
In [53]: conversion_raw, conversion_grouped, conversion_history = get_conversion(
    profiles.query('channel != "organic"'), orders, observation_date, horizon_days, dimensions=['device']
)

plot_conversion(conversion_grouped, conversion_history, horizon_days)
```

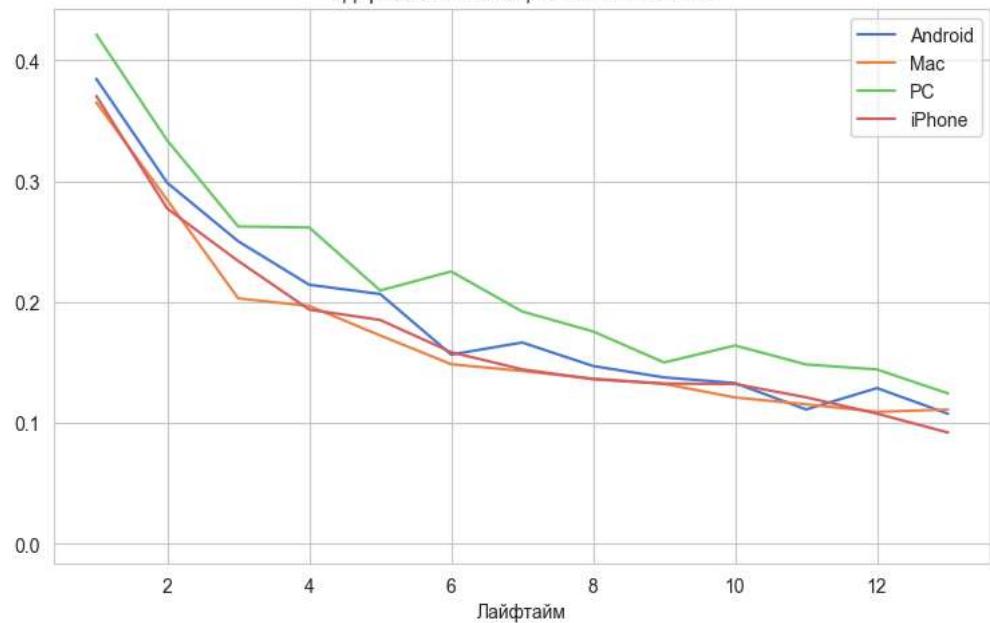


Судя по графикам, наибольшая конверсия у пользователей Mac, iPhone и Android, у PC показатель конверсии меньше. В целом, показатели конверсии низкие. Проверим показатели удержания.

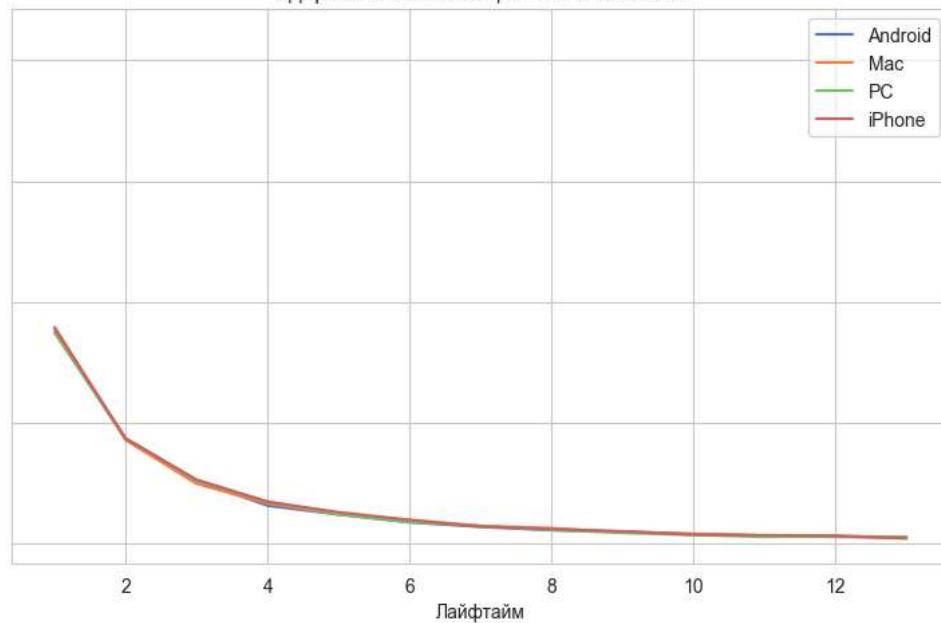
In [54]: # смотрим удержание

```
retention_raw, retention_grouped, retention_history = get_retention(  
    profiles.query('channel != "organic"'), visits, observation_date, horizon_days, dimensions=['device'])  
  
plot_retention(retention_grouped, retention_history, horizon_days)
```

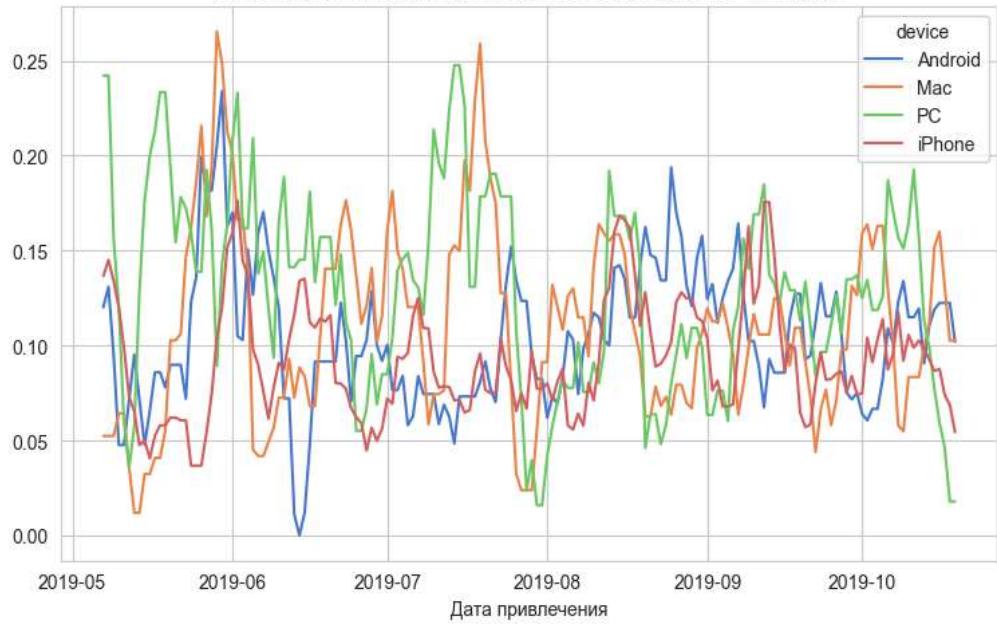
Удержание платящих пользователей



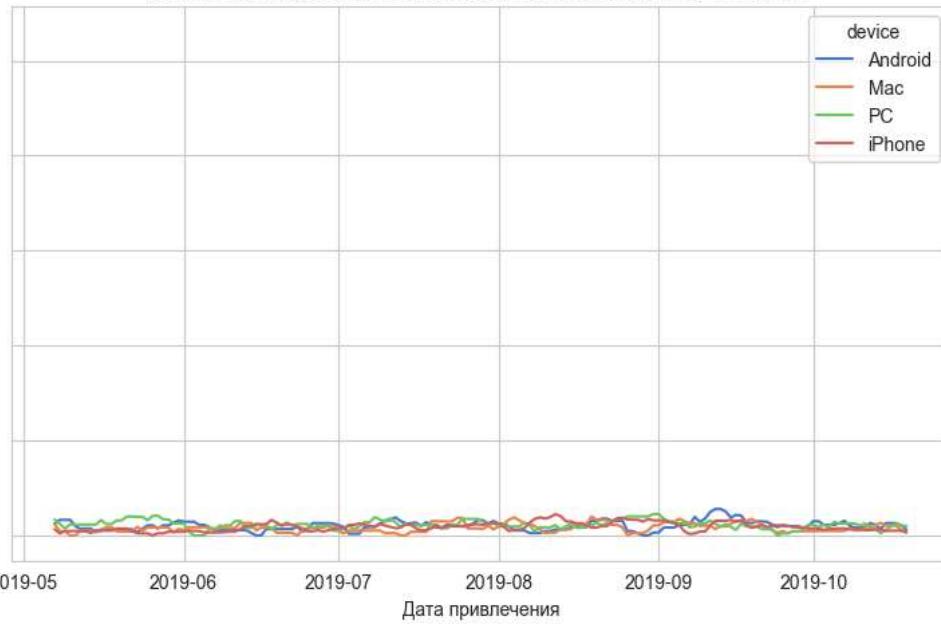
Удержание неплатящих пользователей



Динамика удержания платящих пользователей на 14-й день



Динамика удержания неплатящих пользователей на 14-й день



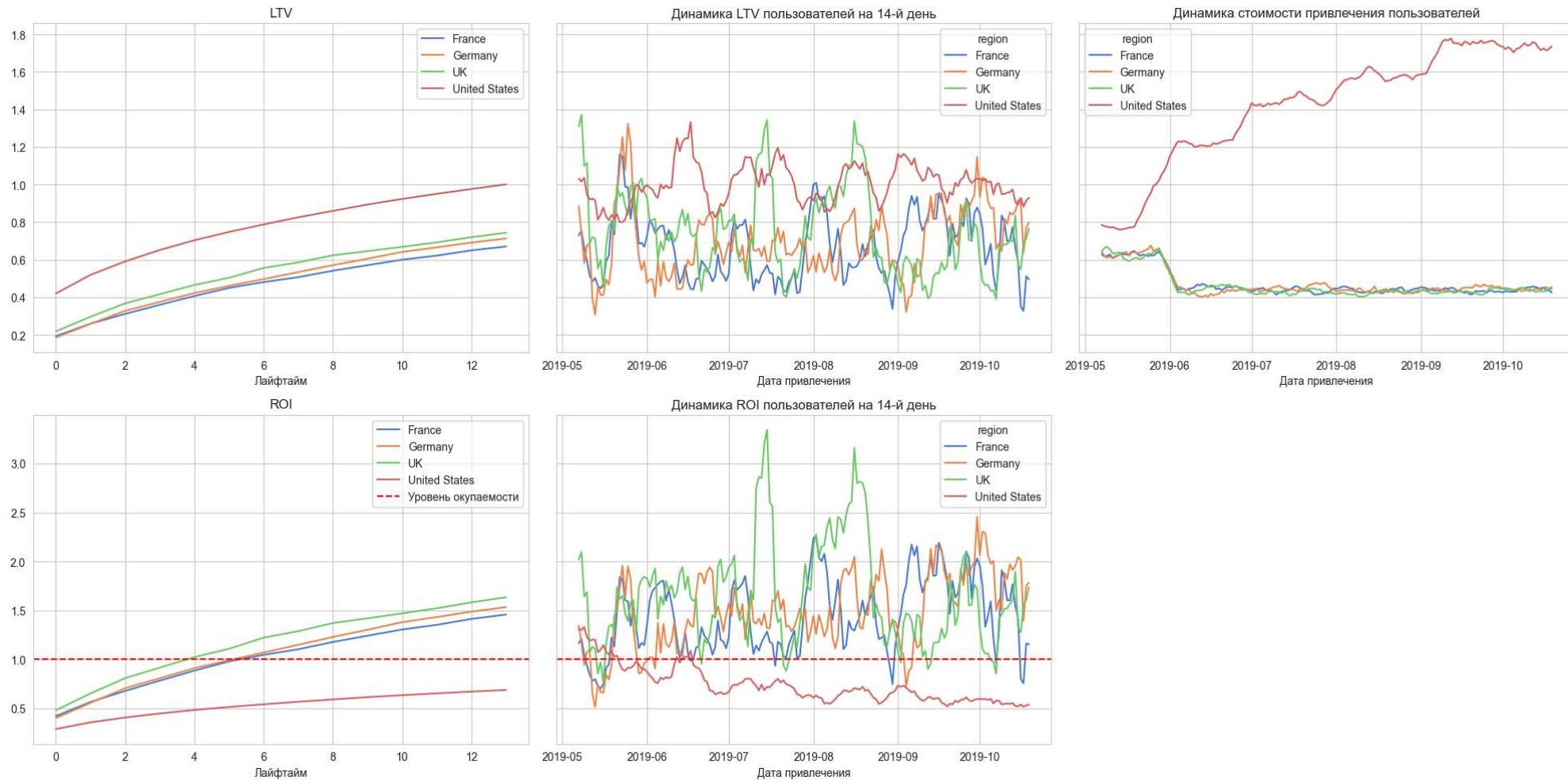
Удержание платящих пользователей выше у пользователей РС, на других устройствах данных показатель чуть ниже. Удержание неплатящих пользователей низкое у всех устройств.

- Проанализируем окупаемость рекламы с разбивкой по странам. Построим графики LTV и ROI, а также графики динамики LTV, СAC и ROI.

In [55]: # считаем LTV и ROI

```
ltv_raw, ltv_grouped, ltv_history, roi_grouped, roi_history = get_ltv(
    profiles.query('channel != "organic"'), orders, observation_date, horizon_days, dimensions=['region']
)

# строим графики
plot_ltv_roi(ltv_grouped, ltv_history, roi_grouped, roi_history, horizon_days)
```



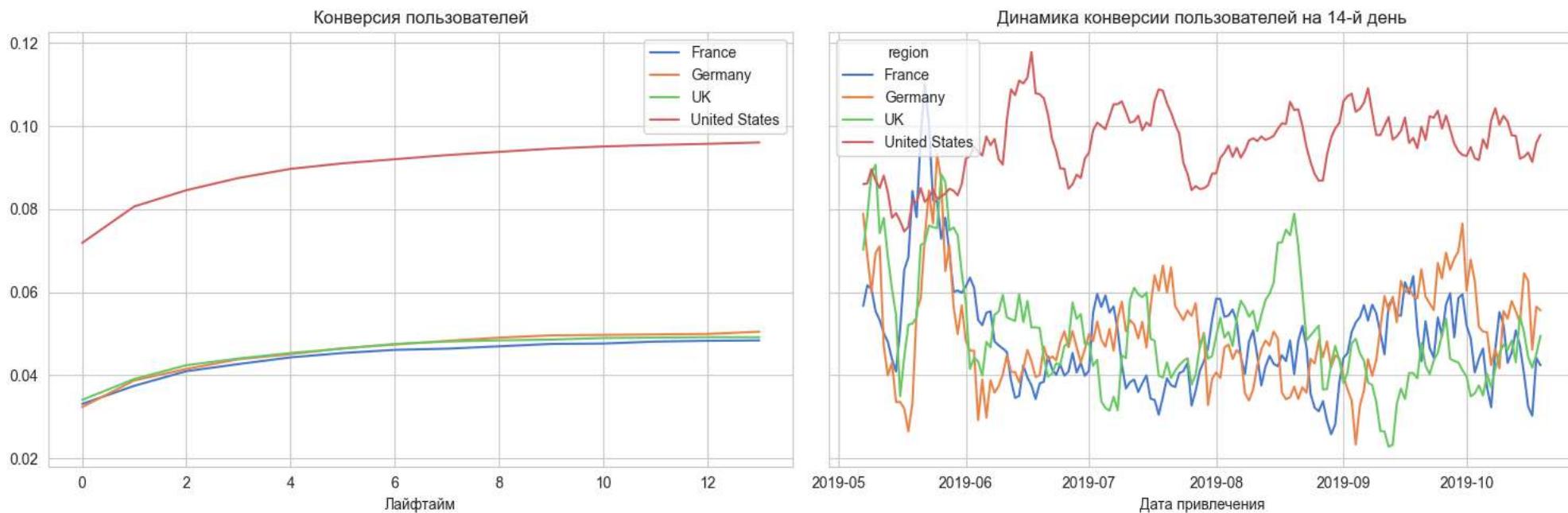
1. Реклама не окупается везде: начиная с июня окупились пользователи из Франции, Германии и Великобритании. Пользователи из Соединенных Штатов не покрыли расходы на рекламу, хотя больше всего пользователей наблюдается именно в Соединенных Штатах - около 100 тысяч (около 2/3 от всех пользователей).

2. Стоимость привлечения пользователей из Соединенных Штатов сильно отличается от остальных стран, с конца мая она непрерывно увеличивается. Стоимость привлечения пользователей из других стран, напротив, начиная с конца мая, снизилась и осталась стабильной на протяжении времени.
3. Из-за растущей стоимости привлечения пользователей из Соединенных Штатов, ROI на 14-ый день для США начиная с конца мая ниже уровня окупаемости.

Посчитаем и визуализируем конверсию, вызвав функции `aet conversion()` и `plot conversion()`

```
In [56]: conversion_raw, conversion_grouped, conversion_history = get_conversion(
    profiles.query('channel != "organic"'), orders, observation_date, horizon_days, dimensions=['region']
)

plot_conversion(conversion_grouped, conversion_history, horizon_days)
```

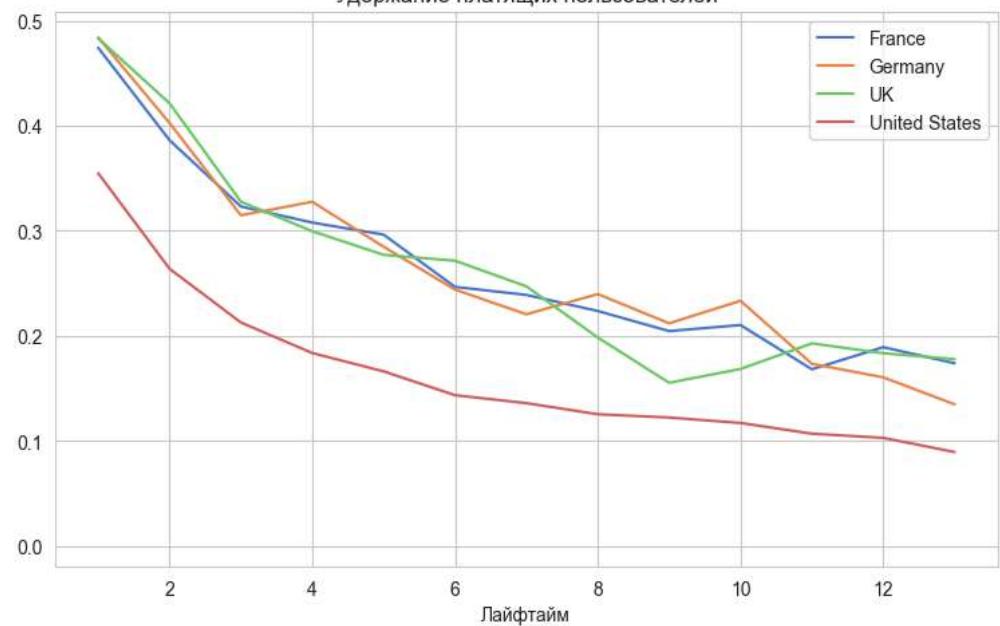


Показатели конверсии низкие у всех стран, но при этом у пользователей из Соединенных Штатов этот показатель выше в 2 раза остальных.

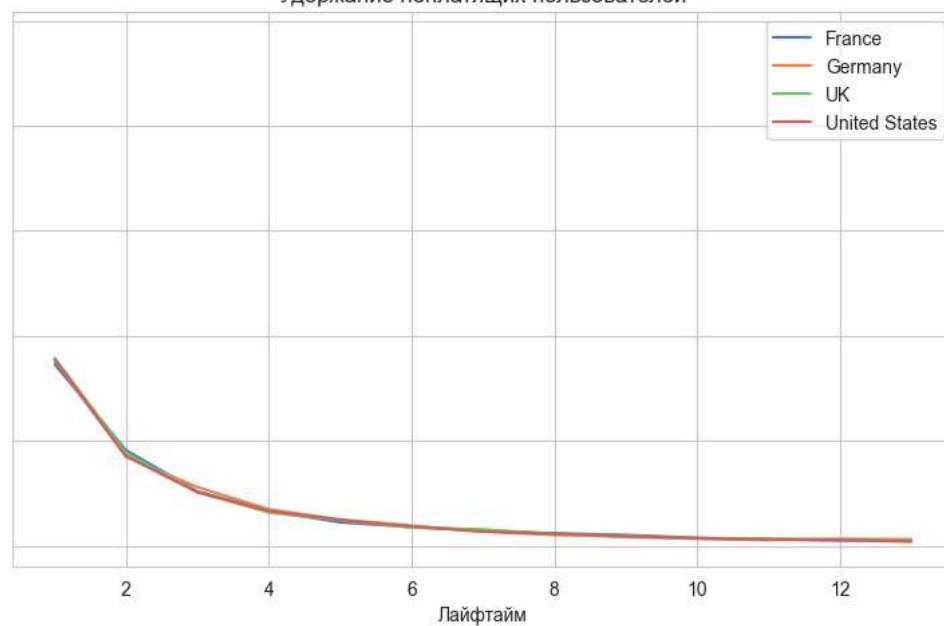
In [57]: # смотрим удержание

```
retention_raw, retention_grouped, retention_history = get_retention(  
    profiles.query('channel != "organic"'), visits, observation_date, horizon_days, dimensions=['region'])  
  
plot_retention(retention_grouped, retention_history, horizon_days)
```

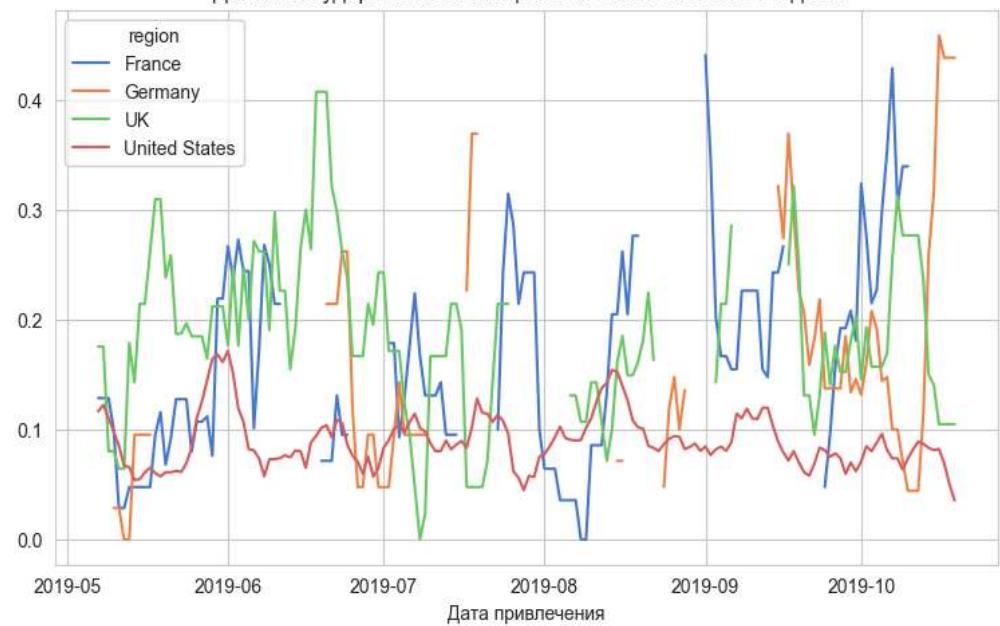
Удержание платящих пользователей



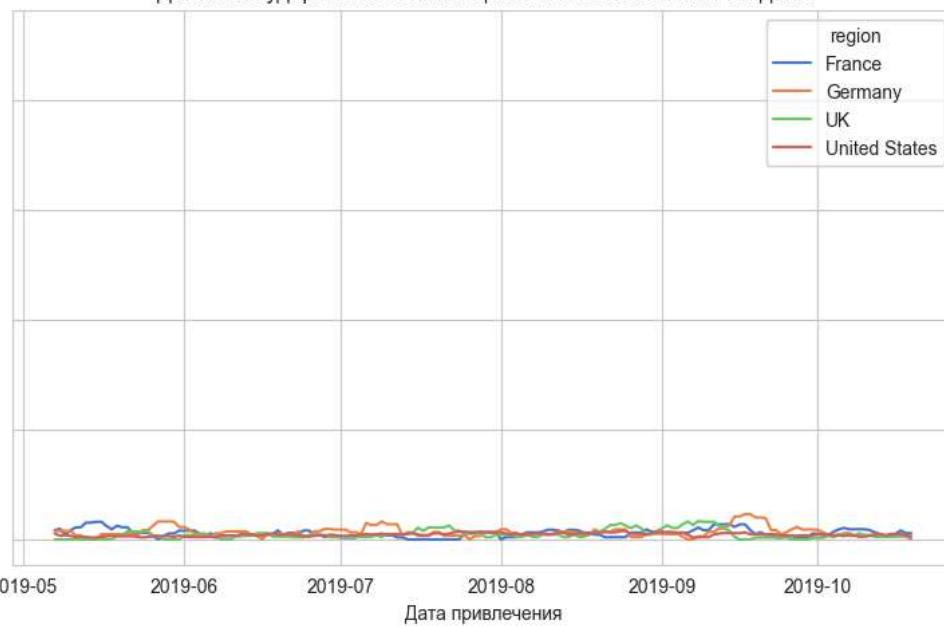
Удержание неплатящих пользователей



Динамика удержания платящих пользователей на 14-й день



Динамика удержания неплатящих пользователей на 14-й день



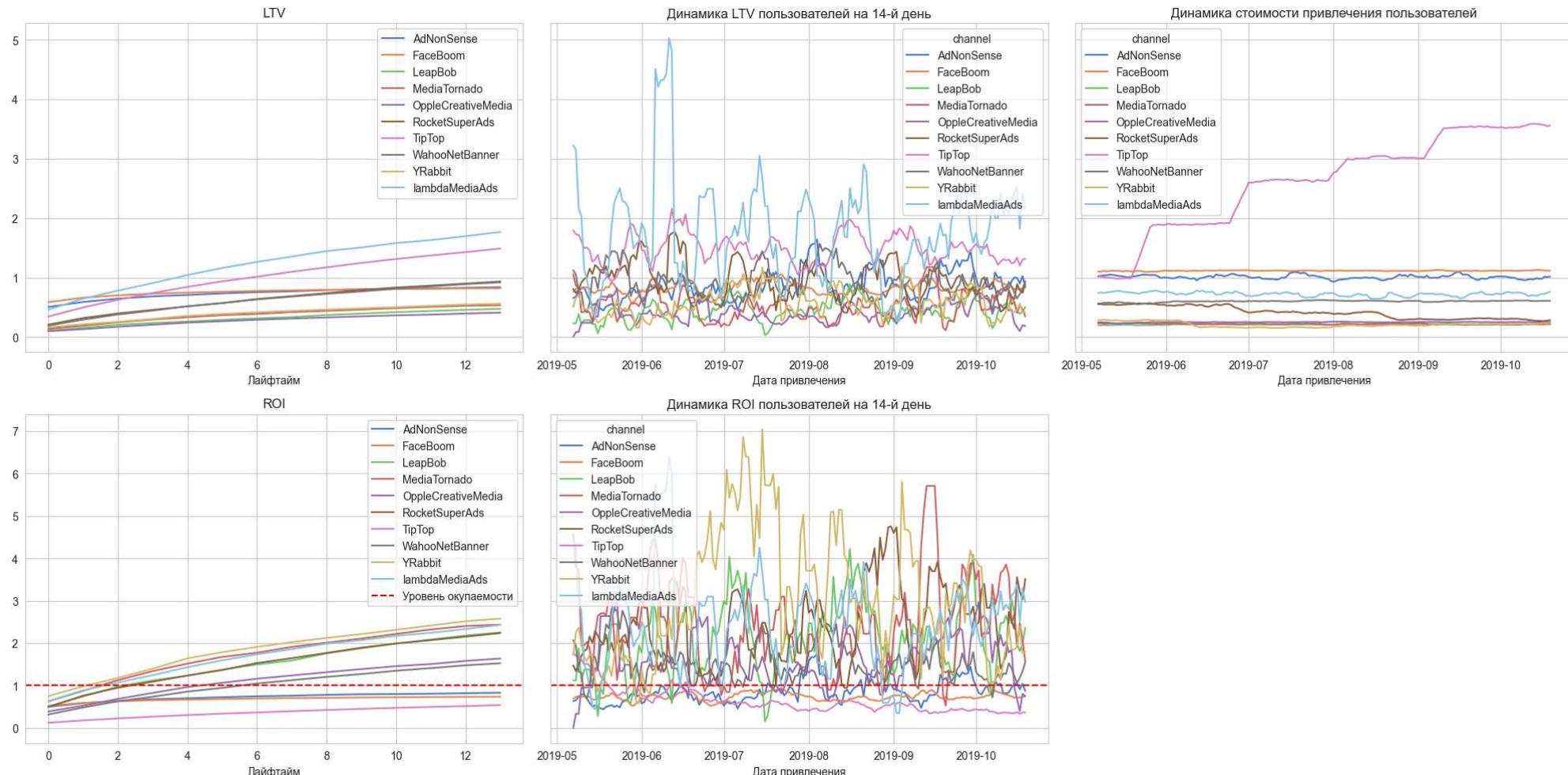
Удержание платящих пользователей из Франции, Германии и Великобритании практически одинаковое, у пользователей из США данный показатель ниже. Удержание неплатящих пользователей низкое у пользователей из всех стран.

- Проанализируем окупаемость рекламы с разбивкой по рекламным каналам. Построим графики LTV и ROI, а также графики динамики LTV, САС и ROI.

In [58]: # считаем LTV и ROI

```
ltv_raw, ltv_grouped, ltv_history, roi_grouped, roi_history = get_ltv(
    profiles.query('channel != "organic"'), orders, observation_date, horizon_days, dimensions=['channel']
)

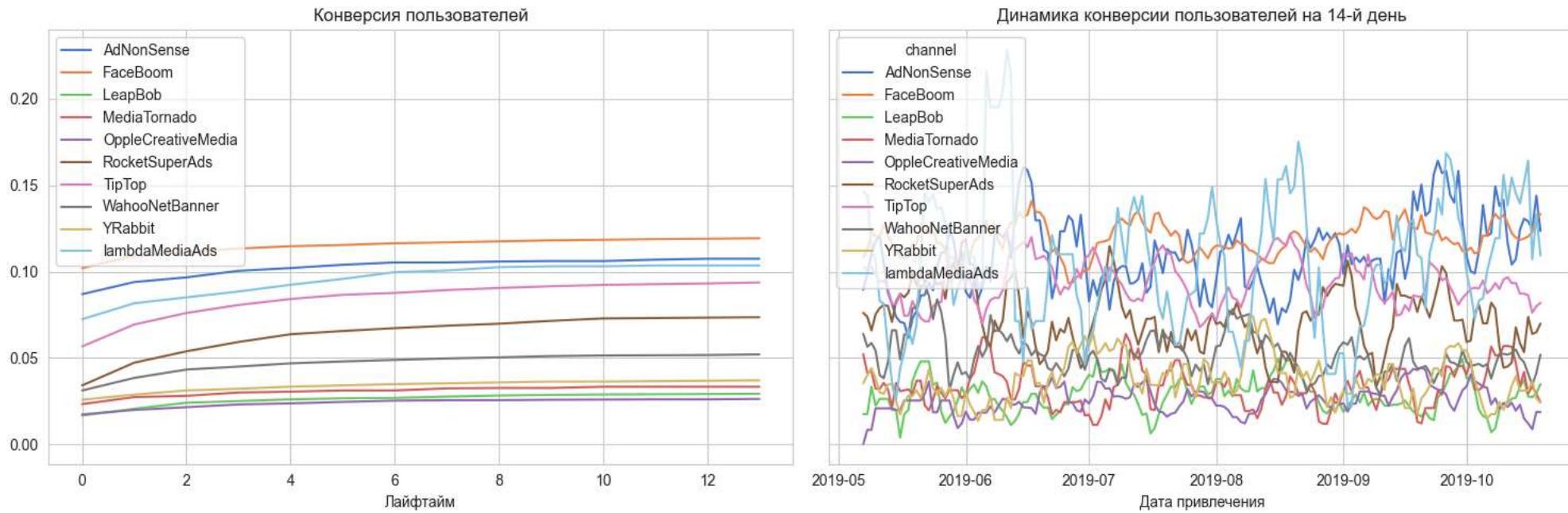
# строим графики
plot_ltv_roi(ltv_grouped, ltv_history, roi_grouped, roi_history, horizon_days)
```



- Такие каналы, как TipTop, FaceBoom и AdNoneSence не окупились. При этом пользователи каналов FaceBoom и TipTop - 29 144 и 19 561 соответственно - из Соединенных Штатов.

2. Стоимость привлечения канала TipTop неизменно растет с середины мая. Также именно с середины мая ROI пользователей на 14-ый день каналов TipTop, FaceBoom, AdNoneSence ниже уровня окупаемости.

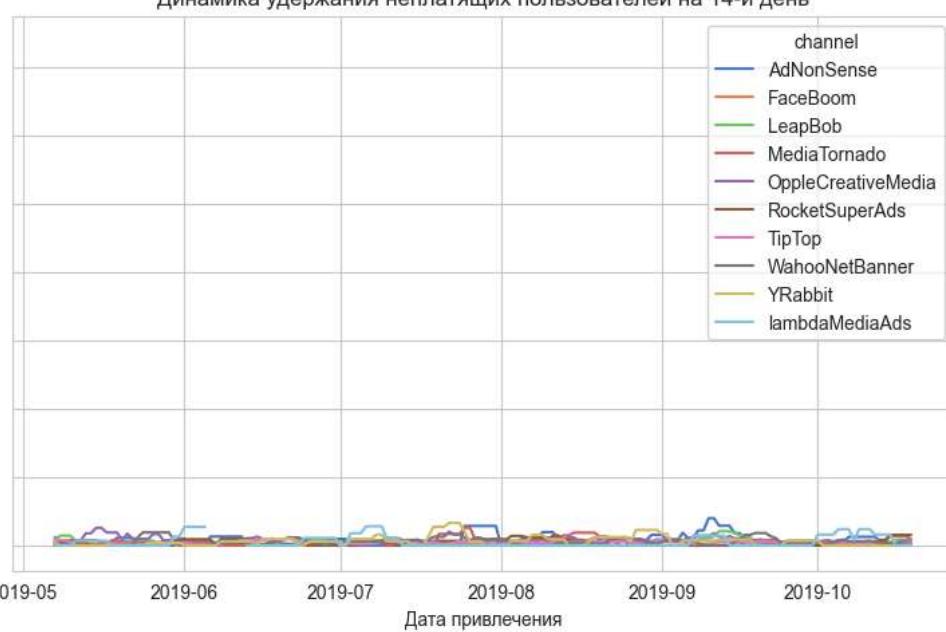
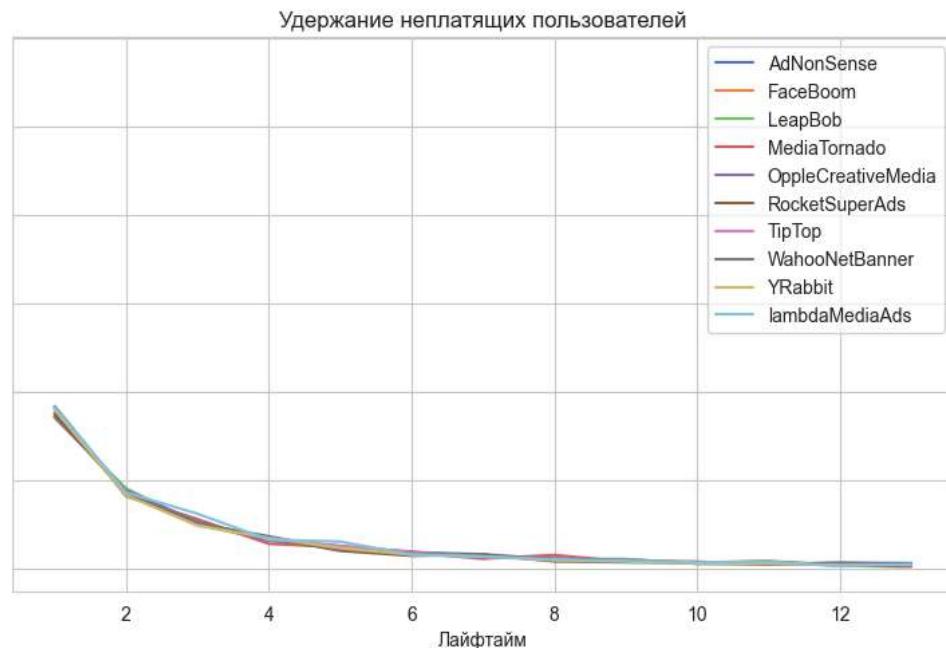
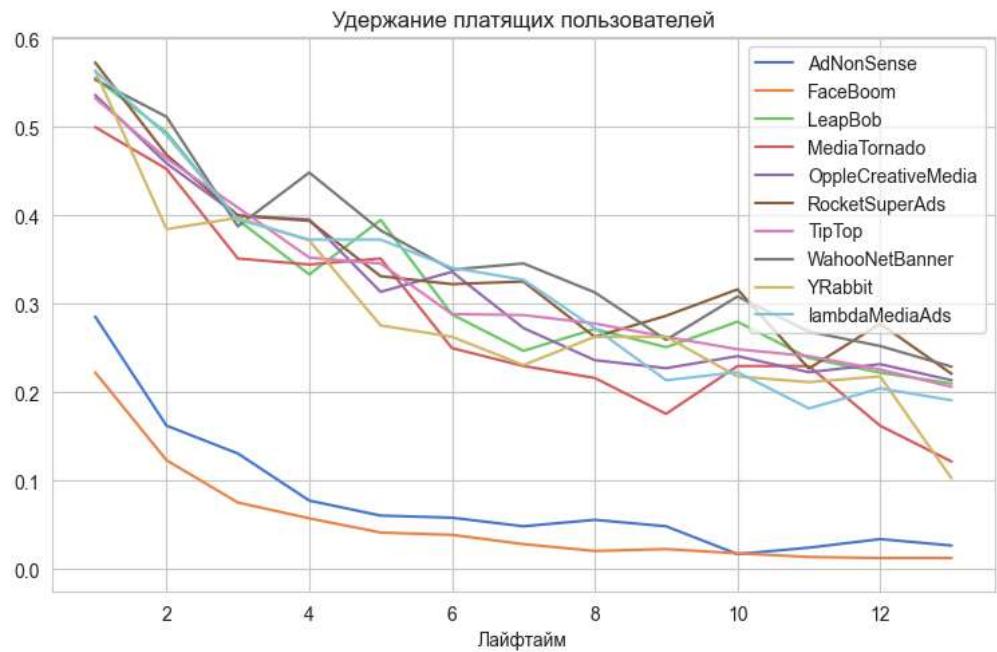
```
In [59]: conversion_raw, conversion_grouped, conversion_history = get_conversion(  
    profiles.query('channel != "organic"'), orders, observation_date, horizon_days, dimensions=['channel'])  
  
plot_conversion(conversion_grouped, conversion_history, horizon_days)
```



Показатели конверсии достаточно низкие у всех каналов привлечения. Наибольшая конверсия - у канала FaceBoom.

In [60]: # смотрим удержание

```
retention_raw, retention_grouped, retention_history = get_retention(  
    profiles.query('channel != "organic"'), visits, observation_date, horizon_days, dimensions=['channel'])  
  
plot_retention(retention_grouped, retention_history, horizon_days)
```



У каналов AdNonSense и FaceBoom - самый низкий показатель удержания. Удержание неплатящих пользователей низкое у пользователей из всех стран.

**Вывод:**

- Реклама с заданным горизонтом анализа и моментом анализа в целом не окупается. ROI на 14ый день — чуть выше 80%. САС нестабилен. Значит, дело в увеличении рекламного бюджета. Возможно, это связано с каналом TipTop, расходы на кототорый возрастили со временем. На LTV влияет сезонный фактор, но и этот показатель достаточно стабилен. Значит, дело не в ухудшении качества пользователей. Начиная с середины июня, ROI снижается - реклама перестала окупаться.
- Самые низкие показатели окупаемости - у устройств Mac и iPhone, так как наибольшая стоимость привлечения именно у данных устройств. Пользователи с PC успевают окупиться в течение 14 дней, скорее всего, дело в невысокой стоимости привлечения таких устройств. Стоимость привлечения по всем устройствам растет практически на протяжении всего периода.
- Такие каналы, как TipTop, FaceBoom, AdNoneSence не окупились. При этом пользователи каналов FaceBoom и TipTop - 29 144 и 19 561 соответственно - из Соединенных Штатов. Стоимость привлечения канала TipTop неизменно растет с середины мая. Также именно с середины мая ROI пользователей на 14-ый день каналов TipTop, FaceBoom, AdNoneSence ниже уровня окупаемости.
- Реклама по странам окупается не везде: начиная с июня окупились пользователи из Франции, Германии и Великобритании. Пользователи из Соединенных Штатов не покрыли расходы на рекламу, хотя больше всего пользователей наблюдается именно в Соединенных Штатах - около 100 тысяч (около 2/3 от всех пользователей). Стоимость привлечения пользователей из Соединенных Штатов сильно отличается от остальных стран, с конца мая она непрерывно увеличивается. Стоимость привлечения пользователей из других стран, напротив, начиная с конца мая, снизилась и осталась стабильной на протяжении времени. Из-за растущей стоимости привлечения пользователей из Соединенных Штатов, ROI на 14-ый день начиная с конца мая ниже уровня окупаемости.
- Удержание платящих пользователей из Франции, Германии и Великобритании практически одинаковое, у пользователей из США данный показатель ниже. Удержание платящих пользователей выше у пользователей PC, на других устройствах данных показатель чуть ниже. Самые низкие показатели удержания платящих пользователей - у каналов AdNonSense и FaceBoom (меньше на 30% по сравнению с другими каналами).

## 6. Общий вывод

Было проведено исследование, чтобы разобраться в причинах убытков компании, несмотря на огромные вложения в рекламу, и помочь приложению Procrastinate Pro+ выйти в плюс. Входные данные - данные о пользователях, привлечённых с 1 мая по 27 октября 2019 года: лог сервера с данными об их посещениях, выгрузка их покупок за этот период, рекламные расходы.

В ходе исследования выявлены следующие закономерности, а также причины неэффективности привлечения пользователей:

- Реклама с заданным горизонтом анализа и моментом анализа в целом не окупается. ROI на 14ый день — чуть выше 80%.
- Самые низкие показатели окупаемости - у устройств **Mac** и **iPhone**, так как наибольшая стоимость привлечения именно у данных устройств. Пользователи с **PC** успевают окупиться в течение 14 дней, скорее всего, дело в невысокой стоимости привлечения таких устройств.
- Такие каналы, как **TipTop**, **FaceBoom**, **AdNoneSence** не окупились. Стоимость привлечения канала **TipTop** неизменно растет с середины мая. Привлечение одного пользователя из источника TipTop обошлось в среднем в 2.8, из источников AdNonSense и FaceBoom - 1 и 1.11

соответственно. Остальные каналы обходятся в сумму от 0.21 до 0.72. Именно по причине высокой стоимости привлечения пользователей реклама данных каналов не успевает окупаться. Также причиной низкой окупаемости является повышение расходов на канал TipTop.

- Пользователи из **Соединенных Штатов** не покрыли расходы на рекламу. Это может быть связано с тем, что такие каналы, как TipTop и FaceBoom не окупили рекламу, так как с данных каналов переходят пользователи только из США. При этом показатели конверсии у пользователей из Соединенных Штатов выше в 2 раза остальных стран.
- Удержание платящих пользователей из Франции, Германии и Великобритании практически одинаковое, у пользователей из США данный показатель ниже. Вероятно, что возможности приложения как-то отличаются для пользователей из США или приложение работает хуже. Также возможно, что в США имеются более дешевые аналоги данного приложения.
- Удержание платящих пользователей выше у пользователей РС, на других устройствах данных показатель чуть ниже. Возможно, что интерфейс приложения для РС более удобный для пользователей.
- Самые низкие показатели удержания платящих пользователей - у каналов **AdNonSense** и **FaceBoom** (меньше на 30% по сравнению с другими каналами). Возможно, что реклама на данных каналах отличается от остальных - как пример, несоответствие информации о приложении в рекламе, что может обнаружиться после покупки.

Рекомендации для отдела маркетинга:

- Снизить расходы на привлечение из источников TipTop и FaceBoom. Сосредоточиться на эффективных каналах привлечения, например, канал YRabbit, который окупается быстрее остальных каналов, и канал lambdaMediaAds, который лидируют по доле платящих пользователей
- Проанализировать рынок в США: узнать, есть ли аналоги приложения Procrastinate Pro+, насколько удобно пользоваться данным приложением пользователям из США.
- Рассмотреть сторонние каналы для привлечения.
- Так как пользователи из Великобритании, Германии и Франция окупаются на 4-6 день, необходимо проанализировать каналы привлечения для данных стран и при возможности увеличить рекламные расходы, направленные на данные страны.
- Сравнить характеристики (быстродействие, баги) приложения на всех устройствах.
- Проанализировать рекламу на каналах AdNonSense и FaceBoom, сравнить ее с другими каналами, насколько данная реклама отличается от остальных, исследовать аудиторию данных каналов.
- Обратить внимание на органических пользователей: исследовать ключевые слова, которые использует целевая аудитория для поиска подобных

```
In [80]: # построим таблицу о количестве пользователей, сгруппированных по признаку
region = profiles.pivot_table(index='region', values='user_id', aggfunc='count') \
    .sort_values(by='user_id') \
    .reset_index()

# отберем платящих пользователей
payers = profiles.query('payer == True') \
    .pivot_table(index='region', values='user_id', aggfunc='count') \
    .sort_values(by='user_id') \
    .reset_index()

# объединим две таблицы по общему полю
overall = region.merge(payers, on='region')
overall.columns = ['region', 'all_users', 'payers']
overall['share'] = round(overall['payers'] / overall['all_users'] * 100, 2)
overall
```

```
Out[80]:   region  all_users  payers  share
0  Germany      14981      616   4.11
1    France      17450      663   3.80
2       UK      17575      700   3.98
3  United States     100002     6902   6.90
```

```
In [77]: from pandas.plotting import table
```

```
In [78]: ax = plt.subplot(111, frame_on=False) # no visible frame
ax.xaxis.set_visible(False) # hide the x axis
ax.yaxis.set_visible(False) # hide the y axis
table(ax, overall, loc='center') # where df is your data frame
plt.savefig('mytable.png')
```

	region	all users	payers	share
0	Germany	14981	616	4.11
1	France	17450	663	3.8
2	UK	17575	700	3.98
3	United States	100002	6902	6.9

```
In [79]: df_styled = overall.style.background_gradient()
dfi.export(df_styled,"mytable.png")
```

```
-----  
FileNotFoundException                                     Traceback (most recent call last)  
Cell In[79], line 2  
      1 df_styled = overall.style.background_gradient()  
----> 2 dfi.export(df_styled,"mytable.png")  
  
File ~\anaconda3\envs\Desktop\da_practicum_env.yml\lib\site-packages\dataframe_image\_pandas_accessor.py:48, in export(obj, file  
ame, fontsize, max_rows, max_cols, table_conversion, chrome_path, dpi)  
    38     def export(  
    39         obj,  
    40         filename,  
    (...)  
    46         dpi=None,  
    47     ):  
---> 48     return _export(  
    49         obj, filename, fontsize, max_rows, max_cols, table_conversion, chrome_path, dpi  
    50     )  
  
File ~\anaconda3\envs\Desktop\da_practicum_env.yml\lib\site-packages\dataframe_image\_pandas_accessor.py:70, in _export(obj, file  
ame, fontsize, max_rows, max_cols, table_conversion, chrome_path, dpi)  
    60     converter = Html2ImageConverter(  
    61         max_rows=max_rows,  
    62         max_cols=max_cols,  
    (...)  
    67         device_scale_factor=(1 if dpi == None else dpi / 100.0),  
    68     ).run  
    69 elif table_conversion == "chrome":  
---> 70     converter = Screenshot(  
    71         max_rows=max_rows,  
    72         max_cols=max_cols,  
    73         chrome_path=chrome_path,  
    74         fontsize=fontsize,  
    75         encode_base64=False,  
    76         limit_crop=False,  
    77         device_scale_factor=(1 if dpi == None else dpi / 100.0),  
    78     ).run  
    79 elif table_conversion == "selenium":  
    80     from .selenium_screenshot import SeleniumScreenshot  
  
File ~\anaconda3\envs\Desktop\da_practicum_env.yml\lib\site-packages\dataframe_image\_screenshot.py:98, in Screenshot.__init__(se  
lf, center_df, max_rows, max_cols, chrome_path, fontsize, encode_base64, limit_crop, device_scale_factor)  
    96     self.max_rows = max_rows  
    97     self.max_cols = max_cols  
---> 98     self.chrome_path = get_chrome_path(chrome_path)  
    99     self.fontsize = fontsize  
100    self.encode_base64 = encode_base64
```

```
File ~\anaconda3\envs\Desktop\da_practicum_env.yml\lib\site-packages\dataframe_image\screenshot.py:76, in get_chrome_path(chrome_path)
    70 locs = [
    71     r"SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\chrome.exe",
    72     r"SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\msedge.exe",
    73     r"SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\brave.exe",
    74 ]
    75 for loc in locs:
---> 76     handle = winreg.OpenKey(winreg.HKEY_LOCAL_MACHINE, loc)
    77     num_values = winreg.QueryInfoKey(handle)[1]
    78     if num_values > 0:
```

FileNotFoundException: [WinError 2] Не удается найти указанный файл

In [ ]: