Анализ окупаемости рекламы

Несмотря на огромные вложения в рекламу, последние несколько месяцев компания Procrastinate Pro+ терпит убытки.

Задача — 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.

Предстоит 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 клиент

Загрузим данные и подготовим их к анализу

Загрузим данные о визитах, заказах и рекламных расходах из CSV-файлов в переменные.

```
import pandas as pd
import numpy as np
from datetime import datetime, timedelta
from matplotlib import pyplot as plt
```

```
try:
    visits = pd.read_csv('visits_info_short.csv')# журнал сессий
    orders = pd.read_csv('orders_info_short.csv') # покупки
    costs = pd.read_csv('costs_info_short.csv') # траты на рекламу
except:
    visits = pd.read_csv('/datasets/visits_info_short.csv')
    orders = pd.read_csv('/datasets/orders_info_short.csv')
    costs = pd.read_csv('/datasets/costs_info_short.csv')
```

Изучим данные и выполним предобработку

```
visits.info()
```

```
<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
```

```
orders.info()
```

```
<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
```

```
costs.info()
```

```
<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
```

Переименуем столбцы

Приведем все названия столбцов к нижнему регистру

```
visits.columns = visits.columns.str.lower() #приведем все названия столбцов к нижнему регистру
orders.columns = orders.columns.str.lower()
costs.columns = costs.columns.str.lower()
```

Приведем названия столбцов к удобному виду

```
visits.rename(columns = {'session start': 'session_start', 'session end':'session_end', 'user id':'user_id'},\
              inplace = True)
orders.rename(columns = {'user id': 'user_id', 'event dt':'event_dt'}, inplace = True)



# преобразование данных о времени
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
```

Проверим данные на наличие пропусков

```
visits.isna().sum() #проверим пропуски
```

```
user_id          0
region           0
device           0
channel          0
session_start    0
session_end      0
dtype: int64
```

```
orders.isna().sum() #проверим пропуски
```

```
user_id     0
event_dt    0
revenue     0
dtype: int64
```

```
costs.isna().sum() #проверим пропуски
```

```
dt         0
channel    0
costs      0
dtype: int64
```

Проверим данные на дубликаты

```
dupl = visits[visits.duplicated()] #проверим дубликаты
dupl
```

	user_id	region	device	channel	session_start	session_end

```
dupl = orders[orders.duplicated()] #проверим дубликаты
dupl
```

	user_id	event_dt	revenue

```
dupl = costs[costs.duplicated()] #проверим дубликаты
dupl
```

	dt	channel	costs

Дубликатов и пропусков нет. Проверим данные на неявные дубликаты

Проверим уникальные значения

```
visits['region'].value_counts() #проверим уникальные значения
```

```
United States    207327
UK                36419
France            35396
Germany           30759
Name: region, dtype: int64
```

```
visits['device'].value_counts()#проверим уникальные значения
```

```
iPhone     112603
Android     72590
PC          62686
Mac         62022
Name: device, dtype: int64
```

```
costs['channel'].value_counts()#проверим уникальные значения
```

```
LeapBob               180
WahooNetBanner        180
RocketSuperAds        180
MediaTornado          180
FaceBoom              180
TipTop                180
AdNonSense            180
OppleCreativeMedia    180
YRabbit               180
lambdaMediaAds        180
Name: channel, dtype: int64
```

Неявных дубликатов нет

В ходе предобработки данных переименованы столбцы, выполнена проверка на выявление пропусков данных, дубликатов и неявных дубликатов

Зададим функции для расчёта и анализа LTV, ROI, удержания и конверсии.

Это 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.

А также 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.

```
# функция для создания пользовательских профилей

def get_profiles(sessions, orders,  ad_costs, event_names=[]):

    # 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
```

```
# 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 сырые данные
    return result_raw, result_grouped, result_in_time
```

```
# 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 сырые данные
    return result_raw, result_grouped, result_in_time
```

```
# функция для расчёта LTV и ROI

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 = 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 CAC,
        # добавляя параметры из dimensions
        cac = df[['user_id', 'acquisition_cost'] + dims].drop_duplicates()

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

        # считаем ROI: делим LTV на CAC
        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 CAC в таблицу ROI
        roi['cac'] = cac['cac']

        # в финальной таблице оставляем размеры когорт, CAC
        # 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
        )


    # 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
```

```
# функция для визуализации удержания

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 чётвертой 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()
```

```
# функция для визуализации конверсии

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()
```

```
# 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()
```

Исследовательский анализ данных

- Составим профили пользователей. Определим минимальную и максимальную даты привлечения пользователей.
- 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.

- Узнаем, какими устройствами пользуются клиенты и какие устройства предпочитают платящие пользователи. Построим таблицу, отражающую количество пользователей и долю платящих для каждого устройства.
- Изучим рекламные источники привлечения и определите каналы, из которых пришло больше всего платящих пользователей. Построим таблицу, отражающую количество пользователей и долю платящих для каждого канала привлечения.

Составим профили пользователей

```
# получаем профили пользователей
profiles = get_profiles(visits, orders,   costs)
print(profiles.head(5))
```

```
      user_id            first_ts     channel  device         region  \
0    599326 2019-05-07 20:58:57    FaceBoom     Mac  United States
1   4919697 2019-07-09 12:46:07    FaceBoom  iPhone  United States
2   6085896 2019-10-01 09:58:33     organic  iPhone         France
3  22593348 2019-08-22 21:35:48  AdNonSense      PC        Germany
4  31989216 2019-10-02 00:07:44     YRabbit  iPhone  United States

           dt      month  payer  acquisition_cost
0  2019-05-07 2019-05-01   True          1.088172
1  2019-07-09 2019-07-01  False          1.107237
2  2019-10-01 2019-10-01  False          0.000000
3  2019-08-22 2019-08-01  False          0.988235
4  2019-10-02 2019-10-01  False          0.230769
```

```
# доступный интервал привлечения пользователей
min_analysis_date = profiles['dt'].min()
observation_date = profiles['dt'].max()  # момент анализа
min_analysis_date, observation_date
```

```
(datetime.date(2019, 5, 1), datetime.date(2019, 10, 27))
```

Минимальная дата привлечения пользователей 2019-05-01

Максимальная дата привлечения пользователей 2019-10-27

Количество пользователей и доля платящих из каждой страны

```
# формируем таблицу количества пользователей с группировкой по странам
dt = pd.DataFrame()
dt['all'] = profiles.groupby('region')['user_id'].count()
dt['payer'] = profiles.query('payer == True').groupby('region')['user_id'].count()
dt['part'] = (dt['payer'] / dt['all'])*100
dt.sort_values(by='part', ascending=False).style.format({'part': '{:.1f}%'})
```

region	all	payer	part
United States	100002	6902	6.9%
Germany	14981	616	4.1%
UK	17575	700	4.0%
France	17450	663	3.8%

Количество пользователей и доля платящих для каждого устройства

```
# формируем таблицу количества пользователей с группировкой по устройствам
dt = pd.DataFrame()
dt['all'] = profiles.groupby('device')['user_id'].count()
dt['payer'] = profiles.query('payer == True').groupby('device')['user_id'].count()
dt['part'] = (dt['payer'] / dt['all'])*100
dt.sort_values(by='part', ascending=False).style.format({'part': '{:.1f}%'})
```

device	all	payer	part
Mac	30042	1912	6.4%
iPhone	54479	3382	6.2%
Android	35032	2050	5.9%
PC	30455	1537	5.0%

Количество пользователей и доля платящих для каждого канала привлечения

```
# формируем таблицу количества пользователей с группировкой по каналам привлечения
dt = pd.DataFrame()
dt['all'] = profiles.groupby('channel')['user_id'].count()
dt['payer'] = profiles.query('payer == True').groupby('channel')['user_id'].count()
dt['part'] = (dt['payer'] / dt['all'])*100
dt.sort_values(by='part', ascending=False).style.format({'part': '{:.1f}%'})
```

channel	all	payer	part
FaceBoom	29144	3557	12.2%
AdNonSense	3880	440	11.3%
lambdaMediaAds	2149	225	10.5%
TipTop	19561	1878	9.6%
RocketSuperAds	4448	352	7.9%
WahooNetBanner	8553	453	5.3%
YRabbit	4312	165	3.8%
MediaTornado	4364	156	3.6%
LeapBob	8553	262	3.1%
OppleCreativeMedia	8605	233	2.7%
organic	56439	1160	2.1%

Маркетинг

- Посчитаем общую сумму расходов на маркетинг.
- Выясним, как траты распределены по рекламным источникам, то есть сколько денег потратили на каждый источник.
- Построим визуализацию динамики изменения расходов во времени (по неделям и месяцам) по каждому источнику.
- Узнаем, сколько в среднем стоило привлечение одного пользователя (CAC) из каждого источника.

Общая сумма расходов на маркетинг

```
costs['costs'].sum().round(2)
```

```
105497.3
```

Общие затраты на маркетинг составляют 105497 долларов

Распределение трат по рекламным источникам

```
# формируем таблицу трат на рекламу с группировкой по рекламным источникам
dt = pd.DataFrame()
dt['sum'] =costs.groupby('channel')['costs'].sum()
dt['part'] = (dt['sum'] / costs['costs'].sum())*100
dt.sort_values(by='part', ascending=False).style.format({'part': '{:.1f}%'})
```

channel	sum	part
TipTop	54751.300000	51.9%
FaceBoom	32445.600000	30.8%
WahooNetBanner	5151.000000	4.9%
AdNonSense	3911.250000	3.7%
OppleCreativeMedia	2151.250000	2.0%
RocketSuperAds	1833.000000	1.7%
LeapBob	1797.600000	1.7%
lambdaMediaAds	1557.600000	1.5%
MediaTornado	954.480000	0.9%
YRabbit	944.220000	0.9%

Самые высокие затраты на рекламу по каналу TipTop, самые низкие по YRabbit

Визуализация динамики изменения расходов во времени

```
import warnings
warnings.filterwarnings('ignore')
#уберем предупреждение, связанное с особенностью библиотеки, оно не влияет на результат

costs['month'] = pd.DatetimeIndex(costs['dt']).month #выделяем столбец с месяцем
costs['week'] = pd.DatetimeIndex(costs['dt']).week #выделяем столбец с номером недели


dt = pd.DataFrame() #формируем таблицу с затратами на рекламу по неделям
dt['sum'] =costs.groupby('week')['costs'].sum()
dt.sort_values(by='sum', ascending=False)
```

week	sum
39	6784.580
40	6365.370
43	5601.140
38	5473.535
41	5190.355
26	4837.120
35	4732.160
33	4703.960
42	4679.935
34	4670.370
27	4660.335
31	4609.530
32	4446.835
36	4280.635
37	4227.870
28	3516.835
30	3445.460
22	3427.075
25	3421.390
29	3039.780
23	2915.740
24	2706.390
21	2297.120
19	2031.820
20	1976.320
18	1455.640

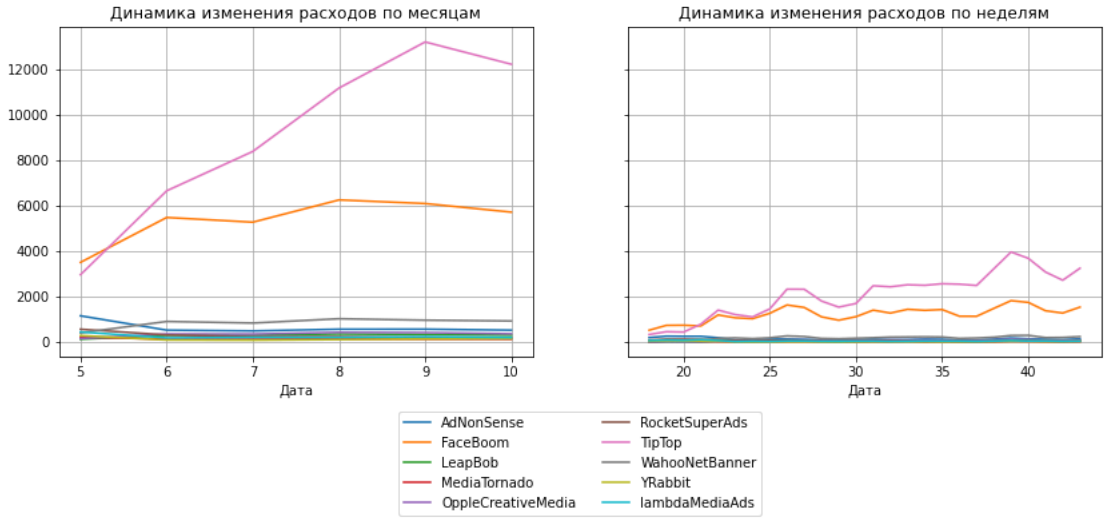
```
dt.plot(grid=True, figsize=(10,5)) # gjcnhjbv uhfabr
plt.xlabel('Дата')
plt.title(' Общая динамика изменения расходов по неделям ')
plt.show()
```

Расходы на рекламу стабильно растут

Построим графики изменения расходов на рекламу по месяцам и неделям с разделением по каналам

```
ax1=plt.subplot(2,2,1)
costs.pivot_table(index='channel', columns='month', values='costs', aggfunc='sum').\
T.plot(grid=True, figsize=(14,10), ax=ax1)
plt.xlabel('Дата')
plt.title(' Динамика изменения расходов по месяцам ')
plt.legend(loc="lower center", bbox_to_anchor=(1.1, -0.5), ncol=2)

ax2=plt.subplot(2,2,2, sharey = ax1)
costs.pivot_table(index='channel', columns='week', values='costs', aggfunc='sum').\
T.plot(grid=True, figsize=(14,10), legend= False, ax=ax2)
plt.xlabel('Дата')
plt.title(' Динамика изменения расходов по неделям ')
plt.show()
```



Затраты на рекламу по каналу TipTop самые высокие и увеличиваются со временем. Затраты по FaceBoom также выше остальных, но остаются стабильно высокими. Затраты на остальные каналы распределены почти равномерно.

Средняя стоимость привлечения одного пользователя (CAC) из каждого источника

```
dt = pd.DataFrame() #сформирует таблицу со средним CAC по каждому каналу
dt['CAC'] =profiles.groupby('channel')['acquisition_cost'].mean()
dt.sort_values(by='CAC', ascending=False)
```

channel	CAC
TipTop	2.799003
FaceBoom	1.113286
AdNonSense	1.008054
lambdaMediaAds	0.724802
WahooNetBanner	0.602245
RocketSuperAds	0.412095
OppleCreativeMedia	0.250000
YRabbit	0.218975
MediaTornado	0.218717
LeapBob	0.210172
organic	0.000000

Самая высокая стоимость привлечения по каналу TipTop. Самая низкая по LeapBob.

Оценим окупаемость рекламы

- Проанализируем окупаемость рекламы с помощью графиков LTV и ROI, а также графики динамики LTV, CAC и ROI.
- Проверим конверсию пользователей и динамику её изменения. То же самое сделаем с удержанием пользователей. Построим и изучим графики конверсии и удержания.
- Проанализируем окупаемость рекламы с разбивкой по устройствам. Построим графики LTV и ROI, а также графики динамики LTV, CAC и ROI.
- Проанализируем окупаемость рекламы с разбивкой по странам. Построим графики LTV и ROI, а также графики динамики LTV, CAC и ROI.
- Проанализируем окупаемость рекламы с разбивкой по рекламным каналам. Построим графики LTV и ROI, а также графики динамики LTV, CAC и ROI.

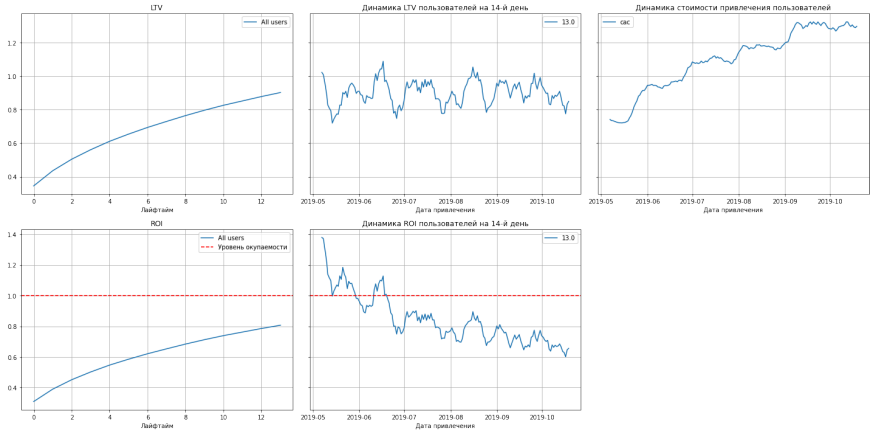
Проанализируем окупаемость рекламы с помощью графиков LTV и ROI и графиков динамики LTV, CAC и ROI

Исключим "органических" пользователей, т.к нас интересуют только платные каналы рекламы, органический трафик бесплатен, поэтому не должен войти в анализ.

```
profiles=profiles.query('channel != "organic"')
```

```
horizon_days=14
observation_date = datetime(2019, 11, 1).date()
# считаем LTV и ROI
ltv_raw, ltv_grouped, ltv_history, roi_grouped, roi_history = get_ltv(
    profiles, 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:

- Рекламный бюджет увеличивается, т.к CAC растет
- Реклама не окупается. Кривая не пересекает порог окупаемости. ROI чуть выше 80%
- LTV достаточно стабилен. Значит, дело не в ухудшении качества пользователей.

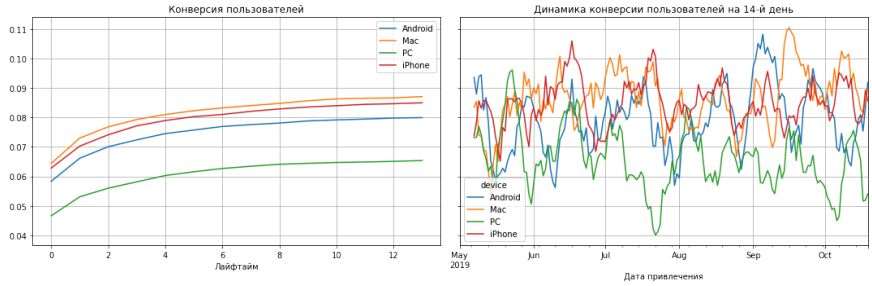
Изучим графики конверсии и удержания, проанализируем окупаемость

Изучим графики с разбивкой по устройствам

График конверсии

```
# смотрим конверсию с разбивкой по устройствам
dimensions=['device']
conversion_raw, conversion_grouped, conversion_history = get_conversion(
    profiles, orders, observation_date, horizon_days, dimensions=dimensions
)

plot_conversion(conversion_grouped, conversion_history, horizon_days)
```



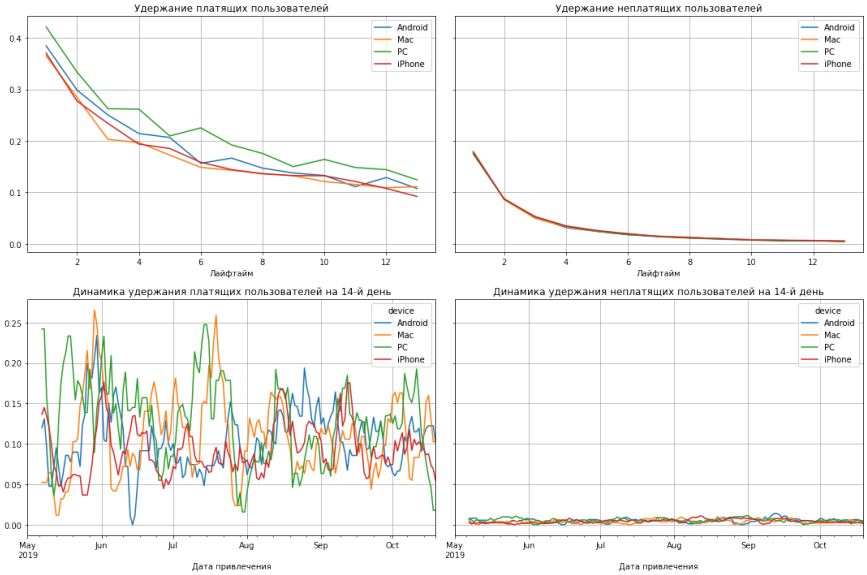
> Пользователи Mac и iPhone становятся покупателями чаще других

> Пользователи PC меньше всего покупали в июле, в остальном данные стабильны

График удержания

```
# смотрим удержание с разбивкой по устройствам
dimensions= ['device']
retention_raw, retention_grouped, retention_history = get_retention(
    profiles, visits, observation_date, horizon_days, dimensions=dimensions
)

plot_retention(retention_grouped, retention_history, horizon_days)
```



По графикам видим, что удержание неплатящих пользователей намного ниже, чем удержание платящих. Удержание неплатящих на графиках истории изменений тоже ниже, чем удержание платящих. Впрочем, как и следовало ожидать.

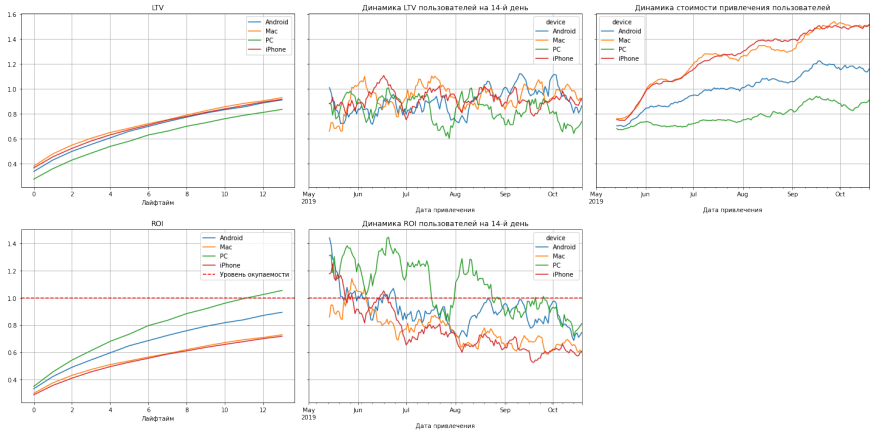
Анализ окупаемости

```
# смотрим окупаемость с разбивкой по устройствам

dimensions = ['device']

ltv_raw, ltv_grouped, ltv_history, roi_grouped, roi_history = get_ltv(
    profiles, orders, observation_date, horizon_days, dimensions=dimensions
)

plot_ltv_roi(
    ltv_grouped, ltv_history, roi_grouped, roi_history, horizon_days, window=14
)
```



> Реклама не окупается для пользователей iPhone, Mac, Android при этом стоимость привлечения для этих пользователей выше, чем для PC

> Окупается только канал PC

> Окупаемость стала падать примерно в одно время для пользователей всех устройств.

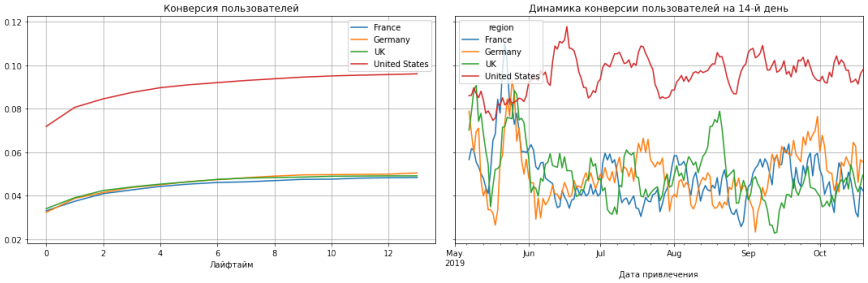
Учитывая, что пользователи iPhone и Mac хорошо конвертируются, возможно имеет смысл сократить затраты на привлечение пользователей, т.к. удержание этих пользователей стабильно.

Изучим графики с разбивкой по странам

График конверсии

```
# смотрим конверсию с разбивкой по странам
dimensions=['region']
conversion_raw, conversion_grouped, conversion_history = get_conversion(
    profiles, orders, observation_date, horizon_days, dimensions=dimensions
)

plot_conversion(conversion_grouped, conversion_history, horizon_days)
```

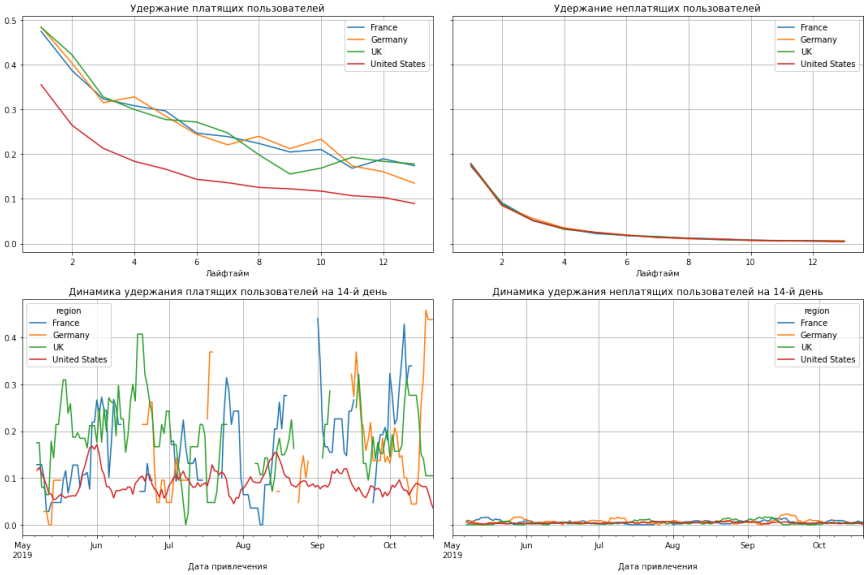


Пользователи из США конвертируются очень хорошо. Остальные пользователи конвертируются примерно одинаково.

График удержания

```
# смотрим удержание с разбивкой по странам
dimensions= ['region']
retention_raw, retention_grouped, retention_history = get_retention(
    profiles, visits, observation_date, horizon_days, dimensions=dimensions
)

plot_retention(retention_grouped, retention_history, horizon_days)
```



> Удержание неплатящих пользователей ниже, чем удержание платящих, что является нормой.

> Удержание неплатящих на графиках динамики тоже ниже, чем удержание платящих. Как и следовало ожидать.

> Обратим внимание, что удержание платящих пользователей в США ниже, чем в других странах.

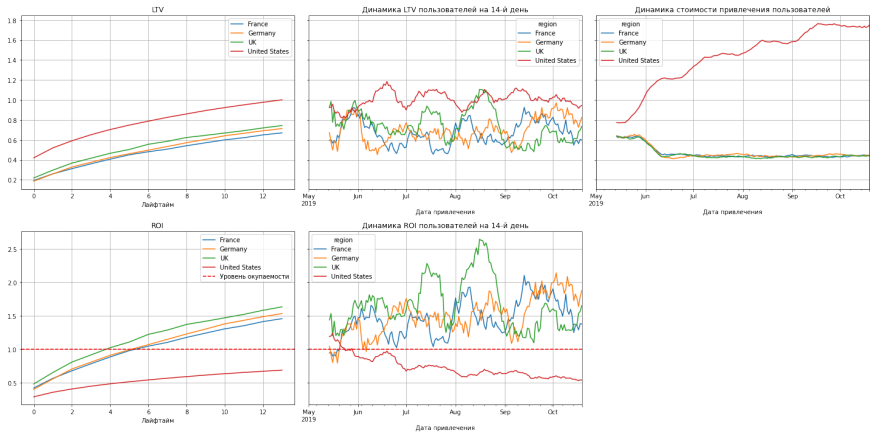
Анализ окупаемости

```
# смотрим окупаемость с разбивкой по странам

dimensions = ['region']

ltv_raw, ltv_grouped, ltv_history, roi_grouped, roi_history = get_ltv(
    profiles, orders, observation_date, horizon_days, dimensions=dimensions
)

plot_ltv_roi(
    ltv_grouped, ltv_history, roi_grouped, roi_history, horizon_days, window=14
)
```



> Окупаются пользователи из всех стран, кроме США

> Затраты на привлечение этих пользователей намного выше, чем пользователей из других стран

> Окупаемость сильно упала в мае и продорлжала падать на фоне стабильности других стран.

Пользователи из США очень хорошо конвертируются, но очень плохо удерживаются. При этом затраты на их привлечение самые высокие и окупаемость самая низкая. Т.к. падение окупаемости характерно только для США, можно предположить влияние какого-то внешнего события, либо техническую неисправность.

Проверим связь между устройствами и страной их использования.

```
dt = pd.DataFrame()
dt['all'] = profiles.groupby('device')['user_id'].count()
dt['usa'] = profiles.query('region == "United States"').groupby('device')['user_id'].count()
dt['part'] = (dt['usa'] / dt['all'])*100
dt.sort_values(by='part', ascending=False).style.format({'part': '{:.1f}%'})
dt
```

```
                all    usa      part

      device
```

Большая часть iPhone и почти все Mac используются в США. Соответственно можем говорить о зависимости конвертации пользователей этих устройств и пользователей из США

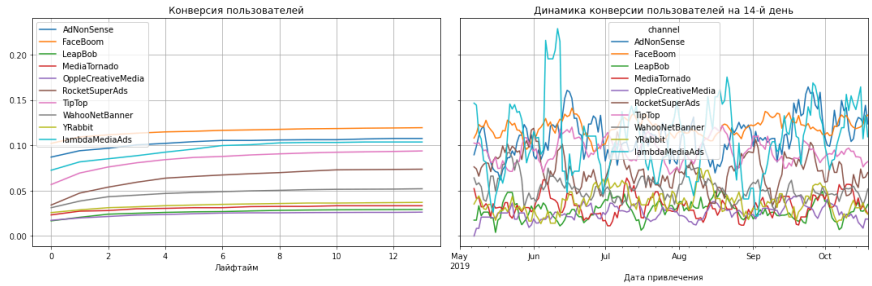
PC 19153 6421 33.524774

Изучим графики с разбивкой по каналам привлечения

График конверсии

```
# смотрим конверсию с разбивкой по каналам привлечения
dimensions=['channel']
conversion_raw, conversion_grouped, conversion_history = get_conversion(
    profiles, orders, observation_date, horizon_days, dimensions=dimensions
)

plot_conversion(conversion_grouped, conversion_history, horizon_days)
```



> Лучше других конвертируются пользователи пришедшие с FaceBoom, AdNonSense, lambdaMediaAds, TipTop

> В июне был скачок конвертации пользователей с lambdaMediaAds. Других аномалий не выявлено

График удержания

```
# смотрим удержание с разбивкой по каналам привлечения
dimensions= ['channel']
retention_raw, retention_grouped, retention_history = get_retention(
    profiles, visits, observation_date, horizon_days, dimensions=dimensions
)

plot_retention(retention_grouped, retention_history, horizon_days)
```