Procrastinate Pro+ причины убытков

Анализируя бизнес показатели компании(LTV, CAC и ROI) нам необходимо выявить причины убытков компании, дать рекомендации по их устранению.

Для анализа у нас есть три датасета: посещения, заказы и траты на рекламу.

План работы

1)Подготовить данные для анализа

2)Задать необходимые функции

3)Проанализировать источники привлечения, устройства, страны привлечения пользователей

4)Посчитать метрики(LTV, CAC и ROI)

Загрузите данные и подготовьте их к анализу

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

Пути к файлам

- визиты: `/datasets/visits_info_short.csv`. Скачать датасет;
- заказы: `/datasets/orders_info_short.csv`. Скачать датасет;
- расходы: `/datasets/costs_info_short.csv`. Скачать датасет.

Изучите данные и выполните предобработку. Есть ли в данных пропуски и дубликаты? Убедитесь, что типы данных во всех колонках соответствуют сохранённым в них значениям. Обратите внимание на столбцы с датой и временем.

Импортируем необходимые библиотеки

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from datetime import datetime, timedelta
events = None  # других событий нет
```

Загрузим данные о визитах, заказах и тратах на рекламу

```
sessions = 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')
```

```
sessions.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
```

Столбцы со временем имеют неправильный формат, преобразуем их

```
sessions['Session Start'] = pd.to_datetime(sessions['Session Start'])
sessions['Session End'] = pd.to_datetime(sessions['Session End'])
orders['Event Dt'] = pd.to_datetime(orders['Event Dt'])
costs['dt'] = pd.to_datetime(costs['dt'])
```

Заменим пробелы в названиях столбцов, приведем все к нижниму регистру

```
for elem in [sessions, orders, costs]:
    elem.columns = elem.columns.str.replace(' ', '_').str.lower()

for elem in [sessions, orders, costs]: #наличие явных дупликатов
    print(elem.duplicated().sum())
```

```
0
0
0
```

```
for elem in [sessions, orders, costs]:   #наличие пропусков
    print(elem.isna().sum())
```

```
user_id          0
region           0
device           0
channel          0
session_start    0
session_end      0
dtype: int64
user_id     0
event_dt    0
revenue     0
dtype: int64
dt         0
channel    0
costs      0
dtype: int64
```

Мы не нашли пропусков и дубликатов, заменили неподходящие типы данных(время), убрали пробелы в названиях столбцов и привели их к нижнему регистру. Все готово для дальнейшего анализа.

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

Разрешается использовать функции, с которыми вы познакомились в теоретических уроках.

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 сглаживания графиков

```
def filter_data(df, window):
    for column in df.columns.values:
        df[column] = df[column].rolling(window).mean()
    return df
```

Функция для создания профилей пользователей

```
# добавляем параметр ad_costs — траты на рекламу
def get_profiles(sessions, orders, events, ad_costs, event_names=[]):

    # сортируем сессии по ID пользователя и дате привлечения
    # группируем по ID и 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',
            }
        )
         # время первого посещения назовём first_ts
        .rename(columns={'session_start': 'first_ts'})
        .reset_index()  # возвращаем user_id из индекса
    )

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

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

    # добавляем флаги для всех событий из event_names
    for event in event_names:
        if event in events['event_name'].unique():
            # проверяем, встречается ли каждый пользователь
            # среди тех, кто совершил событие event
            profiles[event] = profiles['user_id'].isin(
                events.query('event_name == @event')
```

```
['user_id'].unique()
            )

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

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

    # делим рекламные расходы на число привлечённых пользователей
    # результаты сохраним в столбец acquisition_cost (CAC)
    ad_costs['acquisition_cost'] = ad_costs['costs'] /
ad_costs['unique_users']

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

    # органические пользователи не связаны с данными о рекламе,
    # поэтому в столбце acquisition_cost у них значения NaN
    # заменим их на ноль, ведь стоимость привлечения равна нулю
    profiles['acquisition_cost'] =
profiles['acquisition_cost'].fillna(0)

    return profiles  # возвращаем профили с CAC
```

Функция для расчета 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 расчета 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 расчета ltv, roi

```
def get_ltv(
    profiles,
    purchases,
    observation_date,
    horizon_days,
    dimensions=[],
    ignore_horizon=False,
):

    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')
```

```
    result_raw = result_raw.merge(
        purchases[['user_id', 'event_dt', 'revenue']], on='user_id',
how='left'
    )
    result_raw['lifetime'] = (
        result_raw['event_dt'] - result_raw['first_ts']
    ).dt.days

    if len(dimensions) == 0:
        result_raw['cohort'] = 'All users'
        dimensions = dimensions + ['cohort']

    def group_by_dimensions(df, dims, horizon_days):
        result = df.pivot_table(
            index=dims, columns='lifetime', values='revenue',
aggfunc='sum'
        )
        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)
        result = result.div(result['cohort_size'], axis=0)
        result = result[['cohort_size'] + list(range(horizon_days))]
        result['cohort_size'] = cohort_sizes
        cac = df[['user_id', 'acquisition_cost'] +
dims].drop_duplicates()
        cac = (
            cac.groupby(dims)
            .agg({'acquisition_cost': 'mean'})
            .rename(columns={'acquisition_cost': 'cac'})
        )
        roi = result.div(cac['cac'], axis=0)
        roi = roi[~roi['cohort_size'].isin([np.inf])]
        roi['cohort_size'] = cohort_sizes
        roi['cac'] = cac['cac']
        roi = roi[['cohort_size', 'cac'] + list(range(horizon_days))]
        return result, roi

    result_grouped, roi_grouped = group_by_dimensions(
        result_raw, dimensions, horizon_days
    )
    if 'cohort' in dimensions:
        dimensions = []
    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 постройки 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 четвёртой 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 постройки графиков 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 постройки графиков ltv, roi, cac

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

    plt.figure(figsize=(20, 10))

    ltv = ltv.drop(columns=['cohort_size'])
    ltv_history = ltv_history.drop(columns=['cohort_size'])[[horizon -
1]]
    cac_history = roi_history[['cac']]
    roi = roi.drop(columns=['cohort_size', 'cac'])
```

```
    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)
    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)
    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)
    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

- Составьте профили пользователей. Определите минимальную и максимальную даты привлечения пользователей.
- Выясните, из каких стран пользователи приходят в приложение и на какую страну приходится больше всего платящих пользователей. Постройте таблицу, отражающую количество пользователей и долю платящих из каждой страны.
- Узнайте, какими устройствами пользуются клиенты и какие устройства предпочитают платящие пользователи. Постройте таблицу, отражающую количество пользователей и долю платящих для каждого устройства.
- Изучите рекламные источники привлечения и определите каналы, из которых пришло больше всего платящих пользователей. Постройте таблицу, отражающую количество пользователей и долю платящих для каждого канала привлечения.

После каждого пункта сформулируйте выводы.

С помощью функции составим профили пользователей

```
users = get_profiles(sessions, orders, events, costs)
```

Определим максимальную и минимальную дату привлечения

```
min_date = users['dt'].min()
max_date = users['dt'].max()

min_date, max_date
```

```
(Timestamp('2019-05-01 00:00:00'), Timestamp('2019-10-27 00:00:00'))
```

В таблице профилей содержаться данные пользователей привлеченных с 1 мая 2019 года до 27 октбря 2019

В техническом задании указаны даты 1 ноября 2019 + 2 недели, благодаря таблице профилей мы сможем увидеть развитие трат и проследить долгосрочные изменения.

Определим страны привлечения пользователей и найдем на какую страну приходится больше всего платящих пользователей

```
users.query('payer==True').groupby('region').sum().sort_values(by='pay
er', ascending=False).drop(columns='user_id')
```

region	payer	acquisition_cost
United States	6902	9407.933684
UK	700	340.642514
France	663	341.556884
Germany	616	313.677099

Таблица с долей платящих пользоватлей по странам

```
region_table = pd.pivot_table(users, values=['payer'],
index=['region'],
                    aggfunc={'payer': ['count', sum]})
region_table.columns = list(map("_".join, region_table.columns))
region_table['payer_share'] = round(region_table['payer_sum'] /
region_table['payer_count']*100, 3)

region_table = region_table.rename(columns={region_table.columns[0]:
'total_users'})


region_table.sort_values(by='payer_sum', ascending=False)
```

region	total_users	payer_sum	payer_share
United States	100002	6902	6.902
UK	17575	700	3.983
France	17450	663	3.799
Germany	14981	616	4.112

Все пользователи распределены по четырем странам (США, Великобритания, Франция и Германия). Больше всего платящих пользоватей в США, доля платящих в Америке также самая высокая(6.9%). Доли оставшихся стран достаточно схожи(Германия = 4.1%, Франция = 3.8% и Великобритания = 4%)

Определим устройства пользователей и найдем количество платящих распределенное по устройству

Создадим таблицу с долей платящих пользователей по устройствам

```
device_table = pd.pivot_table(users, values=['payer'],
index=['device'],
                    aggfunc={'payer': ['count', sum]})
device_table.columns = list(map("_".join, device_table.columns))
device_table['payer_share'] = round(device_table['payer_sum'] /
device_table['payer_count']*100, 3)

device_table = device_table.rename(columns={device_table.columns[0]:
```

```
'total_users'})

device_table.sort_values(by='payer_sum', ascending=False)
```

device	total_users	payer_sum	payer_share
iPhone	54479	3382	6.208
Android	35032	2050	5.852
Mac	30042	1912	6.364
PC	30455	1537	5.047

Пользаватели пользуются четыремя устройствами(Iphone, Android, Mac, Pc). Больше всего платящих пользователей предпочитают Iphone. Самая большая доля платящих пользователей у Mac(6,36%) и Iphone(6.2%), после идет Andriod(5.85%), замыкает таблицу Pc с долей 5%.

```
users.query('payer==True').groupby('device').sum().sort_values(by='pay
er', ascending=False).drop(columns='user_id')
```

device	payer	acquisition_cost
iPhone	3382	4278.846811
Android	2050	2340.421094
Mac	1912	2380.162082
PC	1537	1404.380194

Определим все источники привлечения и количество привлеченных пользователей

Создадим таблицу с долей платящих пользователей по источникам

```
channel_table = pd.pivot_table(users, values=['payer'],
index=['channel'],
                    aggfunc={'payer': ['count', sum]})
channel_table.columns = list(map("_".join, channel_table.columns))
channel_table['payer_share'] = round(channel_table['payer_sum'] /
channel_table['payer_count']*100, 3)

channel_table =
channel_table.rename(columns={channel_table.columns[0]:
'total_users'})


channel_table.sort_values(by='payer_share', ascending=False)
```

channel	total_users	payer_sum	payer_share
FaceBoom	29144	3557	12.205
AdNonSense	3880	440	11.340
lambdaMediaAds	2149	225	10.470
TipTop	19561	1878	9.601

```
RocketSuperAds              4448        352         7.914
WahooNetBanner              8553        453         5.296
YRabbit                     4312        165         3.827
MediaTornado                4364        156         3.575
LeapBob                     8553        262         3.063
OppleCreativeMedia          8605        233         2.708
organic                    56439       1160         2.055
```

```
users.query('payer==True').groupby('channel').sum().sort_values(by='payer', ascending=False).drop(columns='user_id')
```

```
                     payer  acquisition_cost
channel
FaceBoom              3557       3959.805291
TipTop                1878       5232.035624
organic               1160          0.000000
WahooNetBanner         453        272.061112
AdNonSense             440        444.892444
RocketSuperAds         352        147.309385
LeapBob                262         55.119071
OppleCreativeMedia     233         58.356318
lambdaMediaAds         225        165.447552
YRabbit                165         34.870741
MediaTornado           156         33.912643
```

Больше всего пользователей было привлечено из (FaceBoom, TipTop). Доля платящих пользователей выше всего в FaceBoom(12.2%), AdNonSense(11.3%), lamdaMediaAds(10.5%). В TipTop доля равна 9.6%.

Маркетинг

- Посчитайте общую сумму расходов на маркетинг.
- Выясните, как траты распределены по рекламным источникам, то есть сколько денег потратили на каждый источник.
- Постройте визуализацию динамики изменения расходов во времени (по неделям и месяцам) по каждому источнику. Постарайтесь отразить это на одном графике.
- Узнайте, сколько в среднем стоило привлечение одного пользователя (CAC) из каждого источника. Используйте профили пользователей.

Напишите промежуточные выводы.

Общая сумма затрат на маркетинг:

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

```
105497.30000000002
```

Распределение трат по рекламным источникам. Больше всего денег потрачено на привлечение пользователей из источников с высокой долей платящих пользователей(FaceBoom, TipTop, AdNonSense).

```
costs.groupby('channel')['costs'].sum().sort_values(ascending=False)
```

```
channel
TipTop               54751.30
FaceBoom             32445.60
WahooNetBanner        5151.00
AdNonSense            3911.25
OppleCreativeMedia    2151.25
RocketSuperAds        1833.00
LeapBob               1797.60
lambdaMediaAds        1557.60
MediaTornado           954.48
YRabbit                944.22
Name: costs, dtype: float64
```

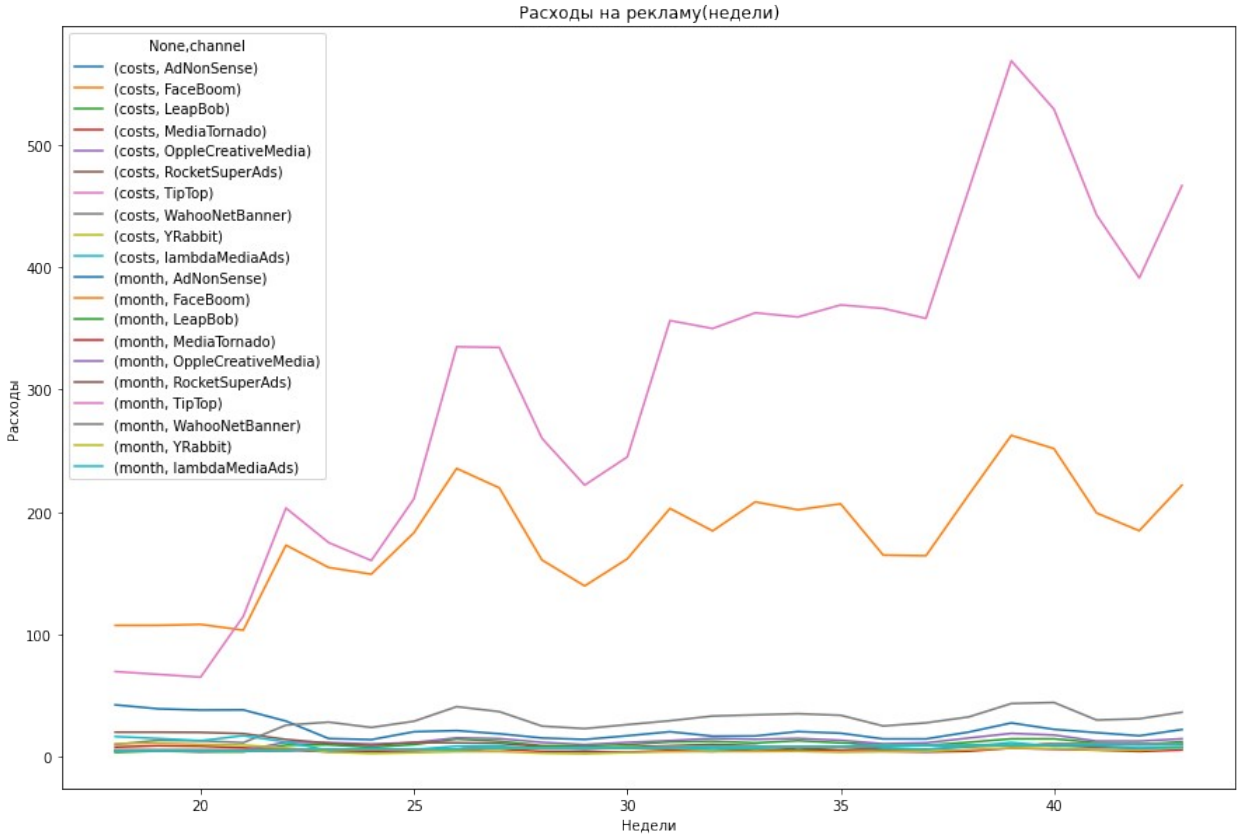
Создадим таблицу содержащую недели и месяцы привлечения

```
costs_time = costs
costs_time['week'] = costs['dt'].dt.isocalendar().week
costs_time['month'] = costs['dt'].dt.month
```

Построим графики изменения расходов на рекламу

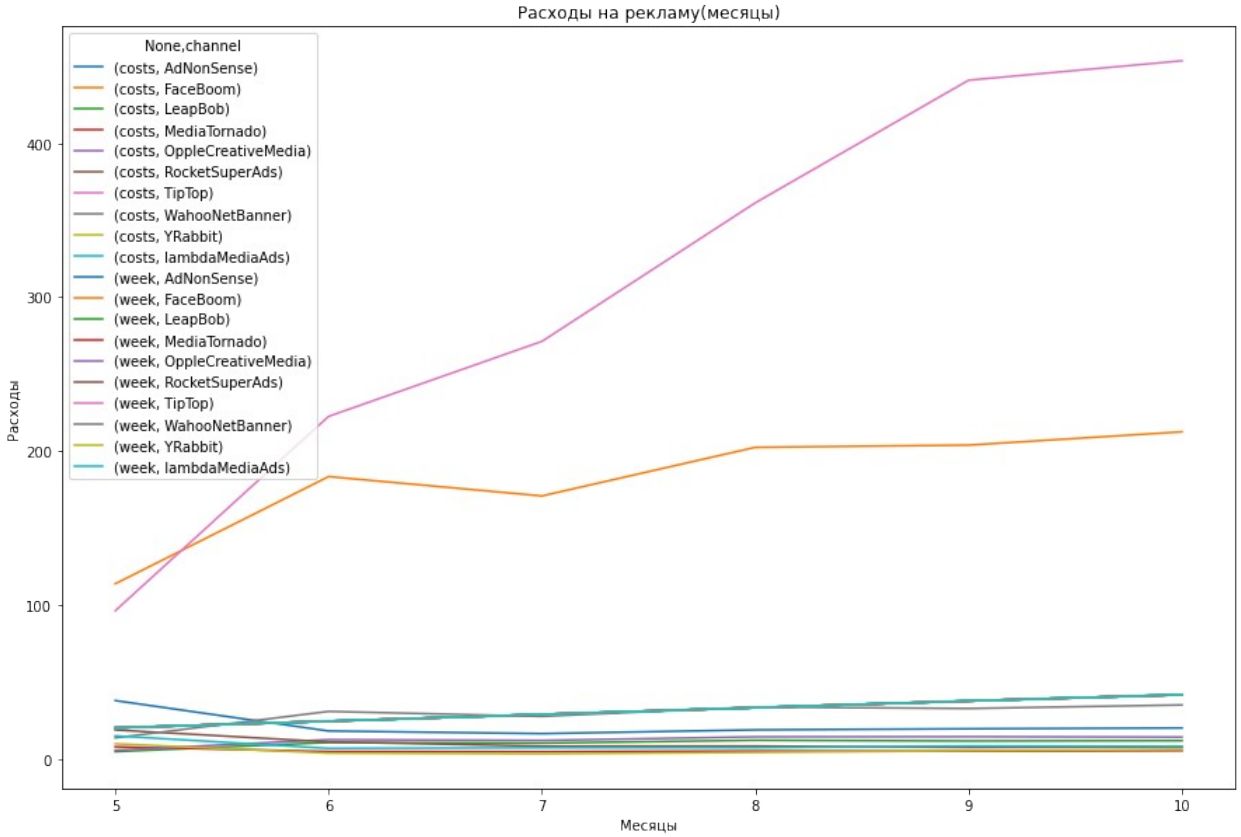
```
costs_table = pd.pivot_table(costs_time, index=['week'],
columns=['channel'])
costs_table.plot(figsize=(15,10), title='Расходы на рекламу(недели)',
ylabel="Расходы", xlabel="Недели")
```

```
<AxesSubplot:title={'center':'Расходы на рекламу(недели)'},
xlabel='Недели', ylabel='Расходы'>
```




```
costs_table = pd.pivot_table(costs_time, index=['month'],
columns=['channel'])
costs_table.plot(figsize=(15,10), title='Расходы на рекламу(месяцы)',
ylabel="Расходы", xlabel="Месяцы")
```

```
<AxesSubplot:title={'center':'Расходы на рекламу(месяцы)'},
xlabel='Месяцы', ylabel='Расходы'>
```



Количество рекламных вложений в источники(FaceBoom, TipTop) постоянно растет.

```
users_channel = pd.pivot_table(users, columns=['channel'],
values=['user_id'], aggfunc={'user_id': ['count']}).T
cost_user = pd.pivot_table(costs, columns=['channel'],
values=['costs'], aggfunc={'costs': [sum]}).T
new_table = users_channel.merge(cost_user, how='inner', on='channel')
new_table.columns = list(map("_".join, new_table.columns))
new_table['CAC'] = new_table['costs_sum']/new_table['user_id_count']
new_table.sort_values(by='CAC', ascending=False)
```

channel	user_id_count	costs_sum	CAC
TipTop	19561	54751.30	2.799003
FaceBoom	29144	32445.60	1.113286
AdNonSense	3880	3911.25	1.008054
lambdaMediaAds	2149	1557.60	0.724802
WahooNetBanner	8553	5151.00	0.602245
RocketSuperAds	4448	1833.00	0.412095
OppleCreativeMedia	8605	2151.25	0.250000
YRabbit	4312	944.22	0.218975
MediaTornado	4364	954.48	0.218717
LeapBob	8553	1797.60	0.210172

Дороже всего стоят пользователи их TipTop и FaceBoom.

Общая сумма расходов на маркетинг 105497.

На TipTop и FaceBoom потрачено более 80% рекалмного бюджета, оставшиеся деньги распределены между остальными источниками.

Вышеупомянутые источники также имеют самый высокий CAC.

Шаг 5. Оцените окупаемость рекламы

```
observation_date = datetime(2019, 10, 27).date()
horizon_date = 14
```

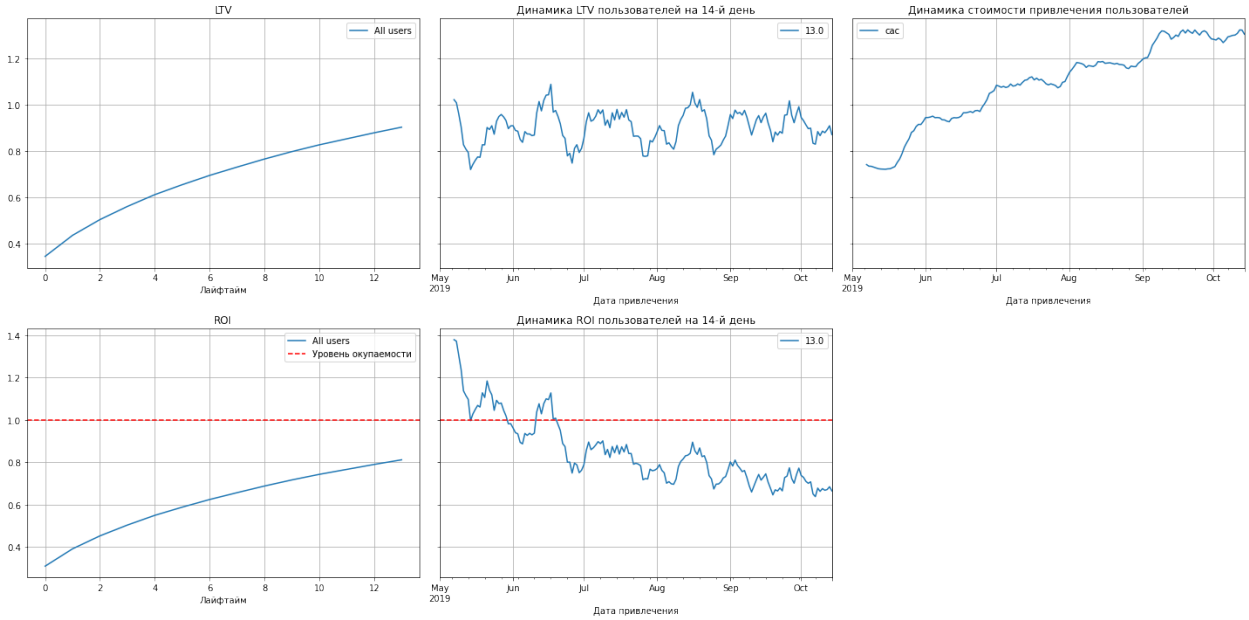
Построим графики LTV, ROI и CAC

```
users_not_organic = users.loc[users['channel']!='organic']
users_not_organic.loc[users_not_organic['channel']=='organic'].count()
```

```
user_id              0
first_ts             0
channel              0
device               0
region               0
dt                   0
month                0
payer                0
acquisition_cost     0
dtype: int64
```

```
ltv_raw, ltv, ltv_history, roi, roi_history =
get_ltv(users_not_organic, orders, observation_date, horizon_date)

plot_ltv_roi(ltv, ltv_history, roi, roi_history, horizon_date)
```



LTV пользователей стабильно, проблема не в этом параметре.

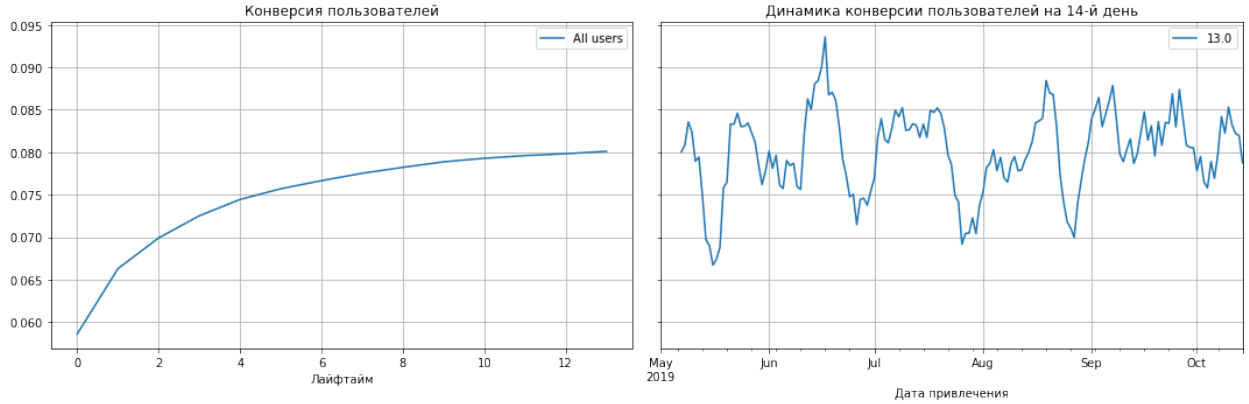
Цена привлечения пользователя резко растет, вместе с ней падает окупаемость.

После июня ROI меньше уровня окупаемости.

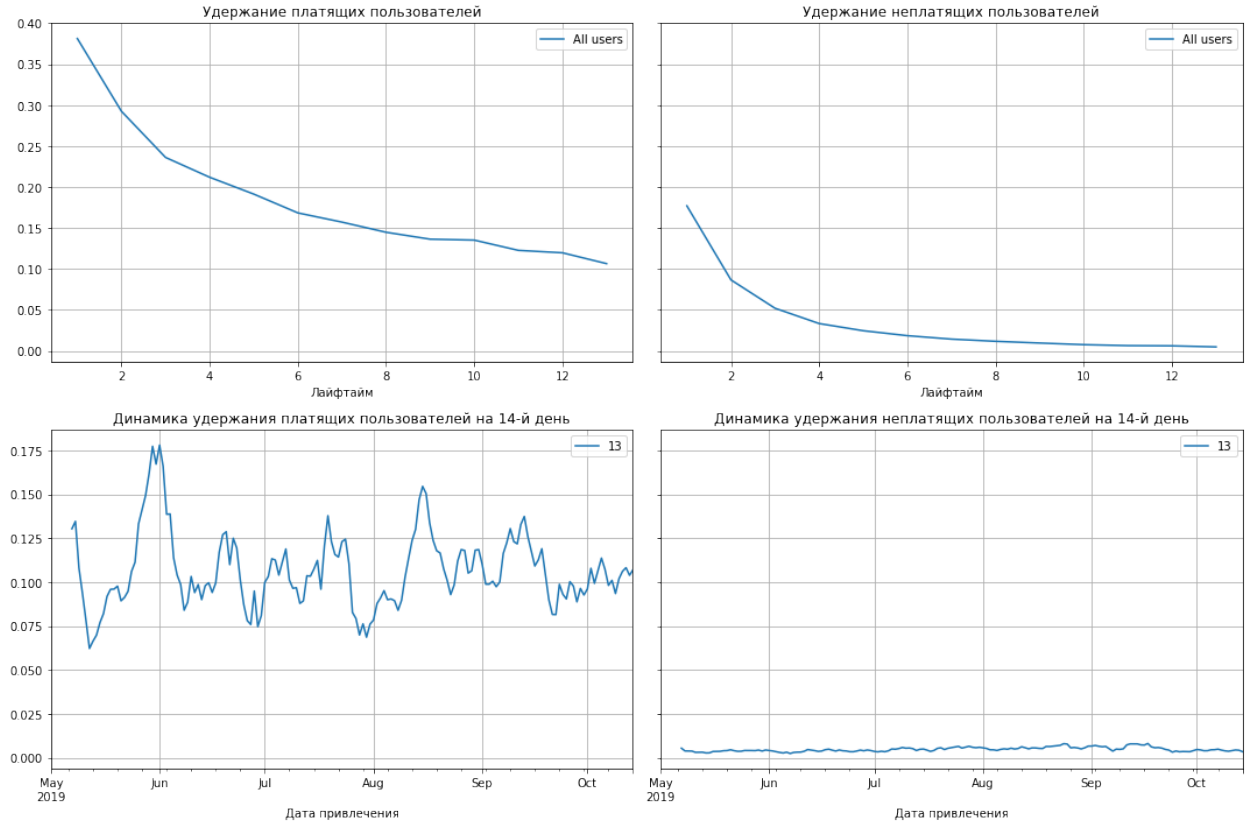
Построим графики конверсии и удержания

```
conversion_raw, conversion_grouped, conversion_history =
get_conversion(users_not_organic, orders, observation_date,
horizon_date)
retention_raw, retention_grouped, retention_history =
get_retention(users_not_organic, sessions, observation_date,
horizon_date)

plot_conversion(conversion_grouped, conversion_history, horizon_date)
```



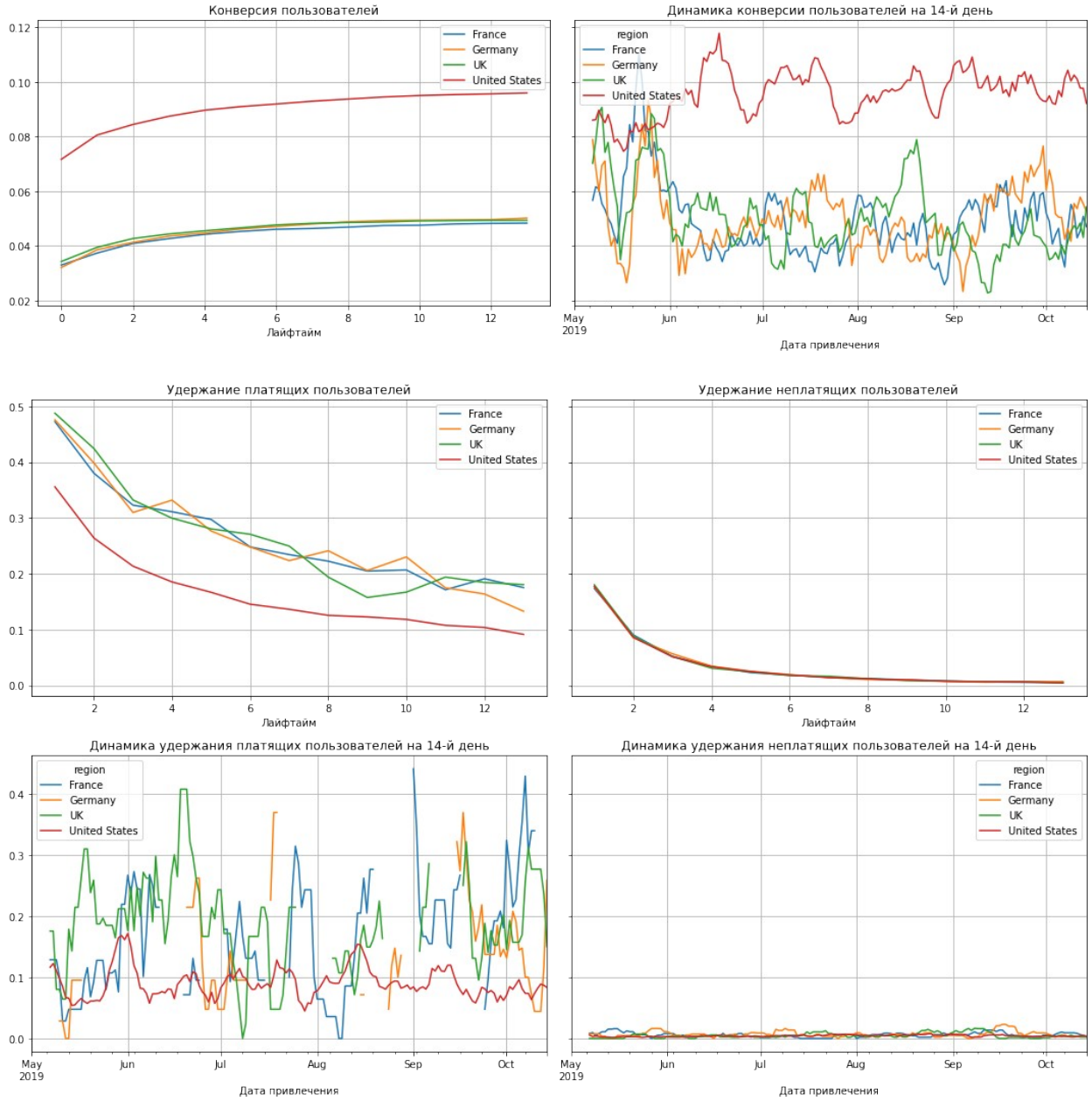
```
plot_retention(retention_grouped, retention_history, horizon_date)
```



Параметры конверсии и удержания на протяжении исследуемого периода остаются стабильными, вряд ли они являются причиной отклонения от бизнес-плана. Однако, следует посмотреть как они изменяться если построить их по устройству, региону или каналу привлечения.

По региону

```
conversion_raw, conversion_grouped, conversion_history =
get_conversion(users_not_organic, orders, observation_date,
horizon_date, dimensions=['region'])
retention_raw, retention_grouped, retention_history =
get_retention(users_not_organic, sessions, observation_date,
horizon_date, dimensions=['region'])
plot_conversion(conversion_grouped, conversion_history, horizon_date)
plot_retention(retention_grouped, retention_history, horizon_date)
```

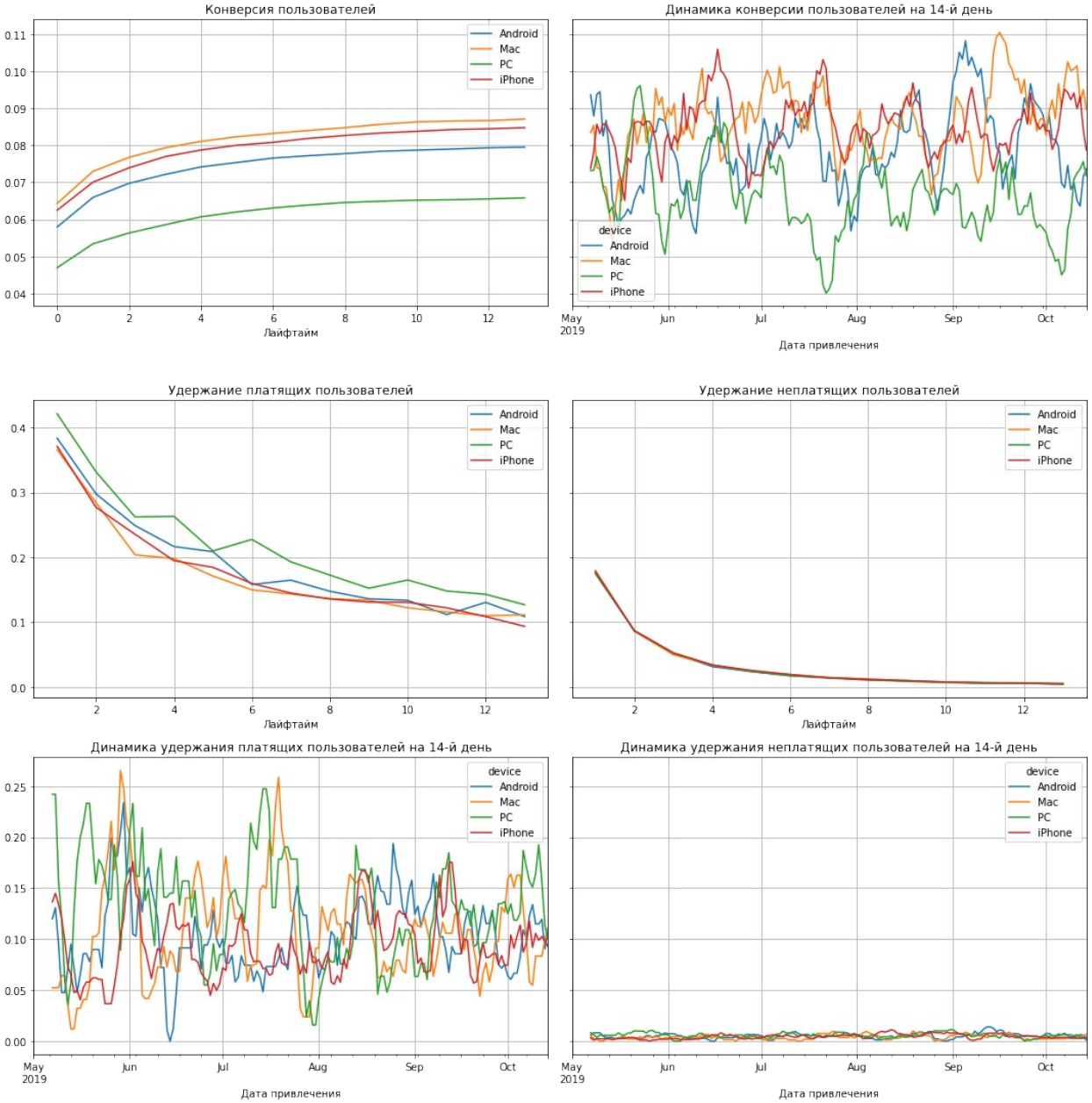


Конверсия пользователей из США значительно выше чем у пользователей из Европы. Однако удержание платящих пользователей меньше чем в Европе. Возможно, высокая конверсия послужила причиной увелечения расходов на рекламу в Америке.

По устройствам

```
conversion_raw, conversion_grouped, conversion_history =
get_conversion(users_not_organic, orders, observation_date,
horizon_date, dimensions=['device'])
retention_raw, retention_grouped, retention_history =
get_retention(users_not_organic, sessions, observation_date,
```

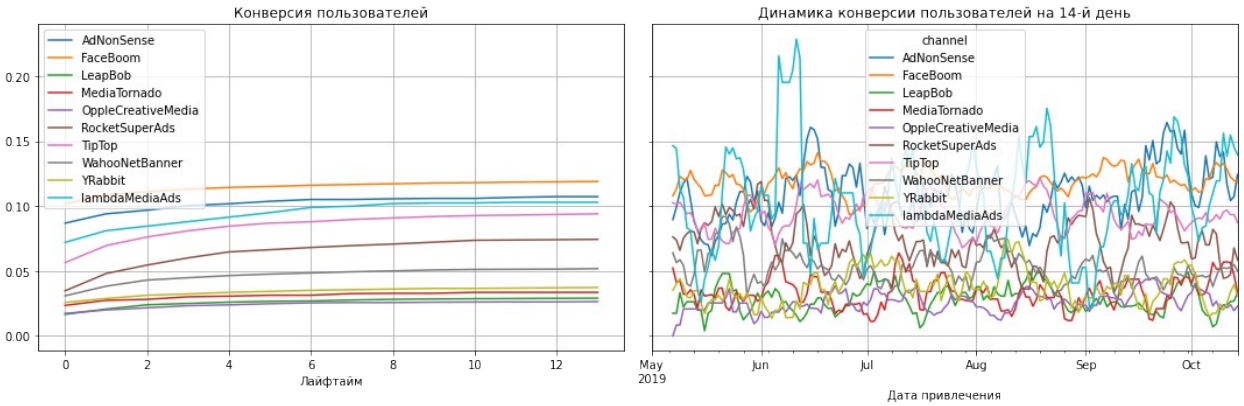
```
horizon_date, dimensions=['device'])
plot_conversion(conversion_grouped, conversion_history, horizon_date)
plot_retention(retention_grouped, retention_history, horizon_date)
```

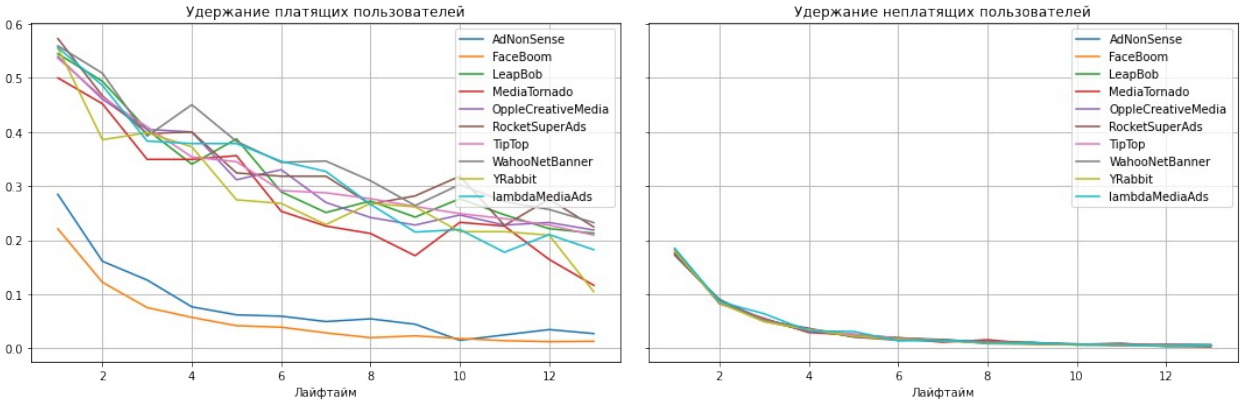


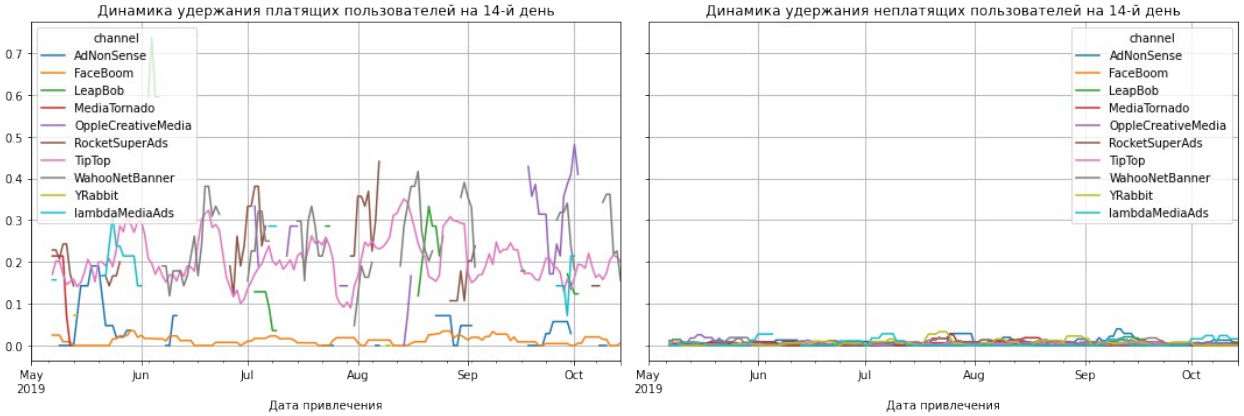
Данные по устройствам повторяют характеристику регионов, так конверсия ios чуть выше, а удержание меньше чем у устройств android и пк. Это может быть связяно с распостраненностью различных устройств в Америке/Европе.

По каналу привлечения

```
conversion_raw, conversion_grouped, conversion_history =
get_conversion(users_not_organic, orders, observation_date,
horizon_date, dimensions=['channel'])
retention_raw, retention_grouped, retention_history =
get_retention(users_not_organic, sessions, observation_date,
horizon_date, dimensions=['channel'])
plot_conversion(conversion_grouped, conversion_history, horizon_date)
plot_retention(retention_grouped, retention_history, horizon_date)
```





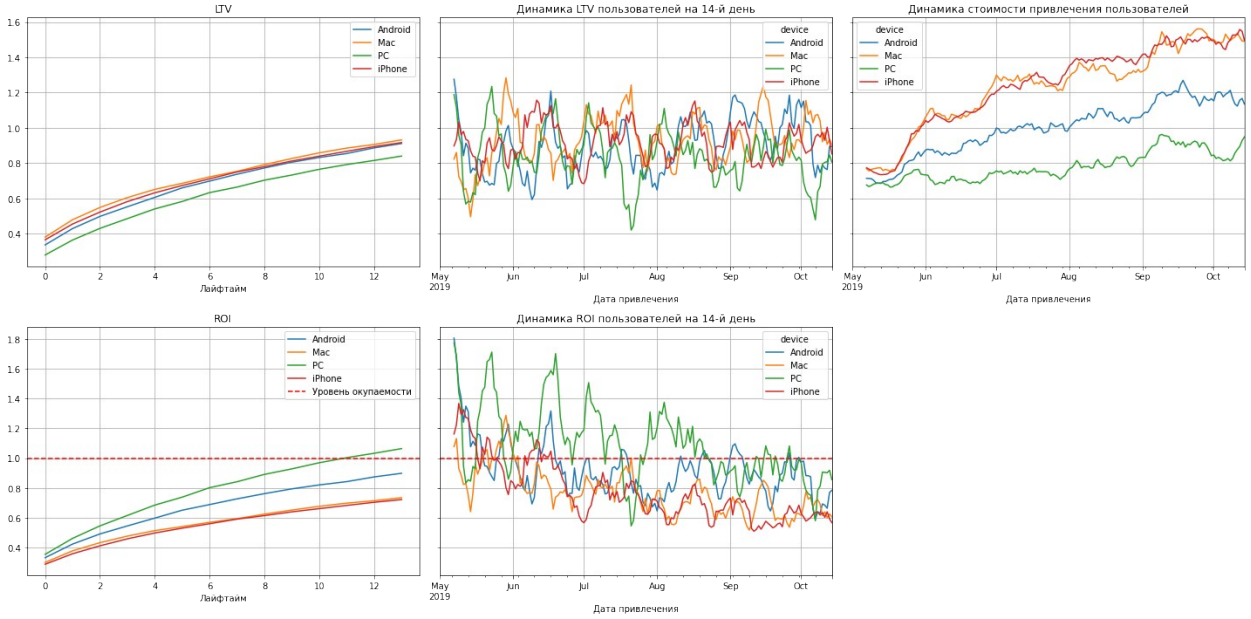


Несмотря на высокую конверсию FaceBoom имеет очень низкое удержание. TipTop же обладает средними показателями, которые врядли стоят такой высокой стоимости привлечения.

Окупаемость рекламы с разбивкой по устройствам

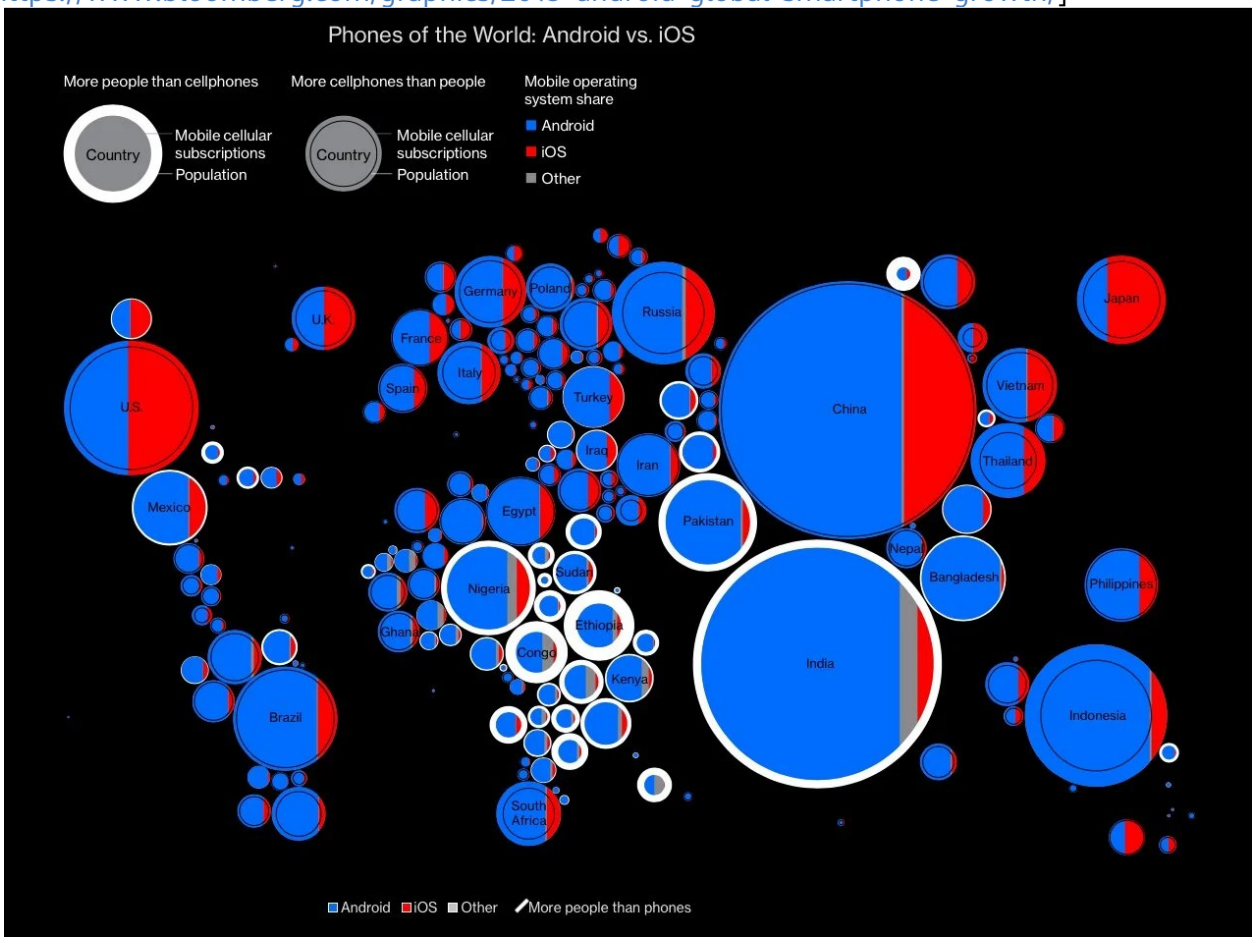
```
ltv_raw, ltv, ltv_history, roi, roi_history =
get_ltv(users_not_organic, orders, observation_date, horizon_date,
dimensions=['device'])

plot_ltv_roi(ltv, ltv_history, roi, roi_history, horizon_date)
```



Ценность пользователя слабо различается от устройства к устройству, а также достаточно стабильна. Возврат на инвестиции падает, и разделеятся по операционным системам ios-android, падение Iphone-Mac более значительно. Стоимость привлечения пользователься в начале лета начинает расти для всех устрйоств, однако позже вперед вырываются Mac и Iphone, менее популярные в Европе и более популярные в США.

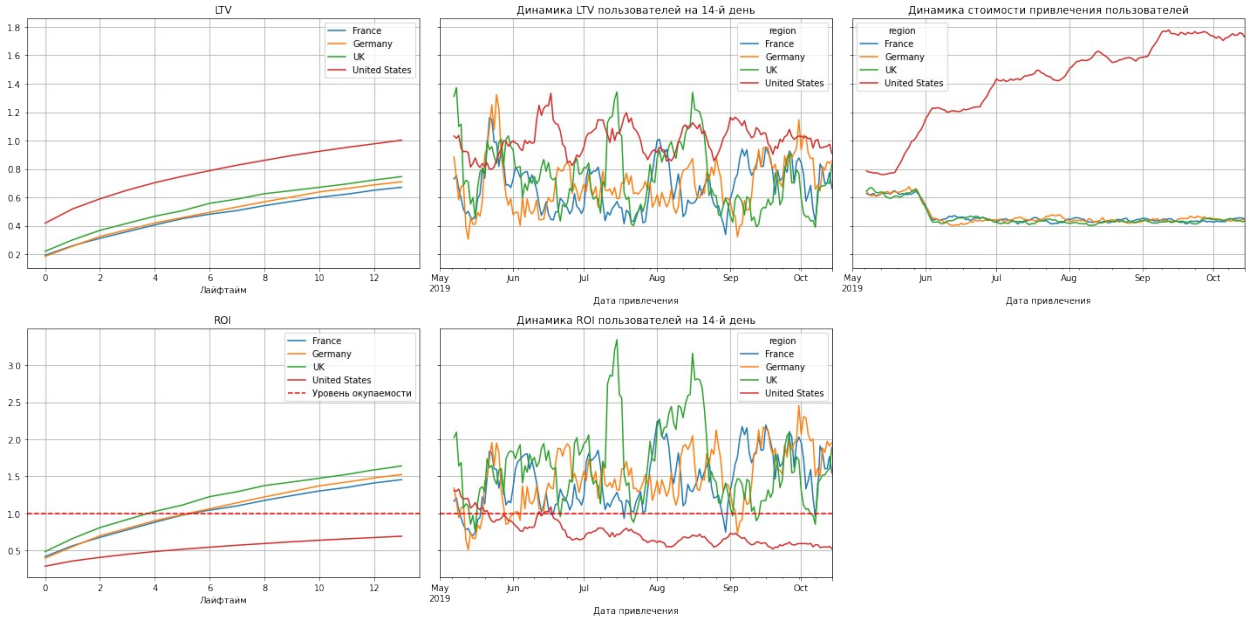
Данные за 2019 год, однако координальных изменений не произошло.
[https://www.bloomberg.com/graphics/2019-android-global-smartphone-growth/]



Окупаемость рекламы с разбивкой по странам

```
ltv_raw, ltv, ltv_history, roi, roi_history =
get_ltv(users_not_organic, orders, observation_date, horizon_date,
dimensions=['region'])

plot_ltv_roi(ltv, ltv_history, roi, roi_history, horizon_date)
```



1)Ценность пользователей в различных странах остается стабильной

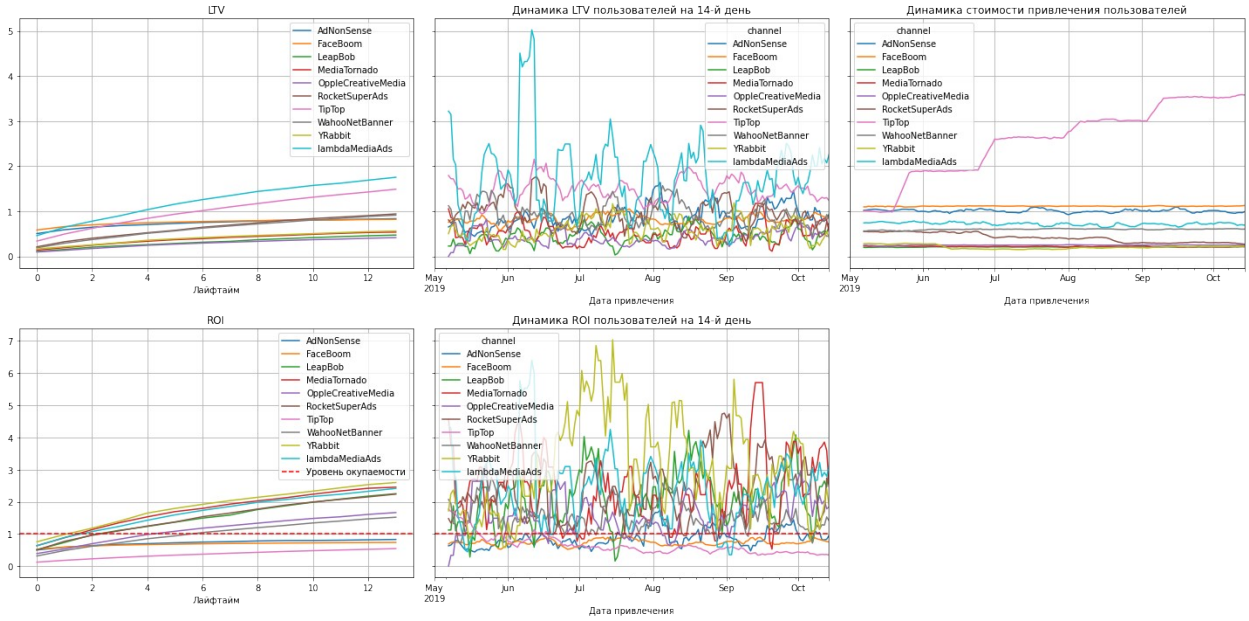
2)Стоимость привлечения пользователя значительно растет только в США

3)Окупавшиеся в мае, привлеченные из США пользователи перестают окупаться к середине лета. В отличии от пользователей из остальных регионов.

Таким образом не окупаются только пользователи из США, привлеченные после середины лета. В тот же период когда увеличиваются закупки рекламы в TipTop и FaceBoom, соцсетях где стоимость привлечения пользователя наивысшая.

Окупаемость рекламы по каналам

```
ltv_raw, ltv, ltv_history, roi, roi_history =
get_ltv(users_not_organic, orders, observation_date, horizon_date,
dimensions=['channel'])
plot_ltv_roi(ltv, ltv_history, roi, roi_history, horizon_date)
```



1)Ценность клиента для большинства каналов остается стабильной.

2)Стоимость привлечения клинетов из TipTop постоянно увеличивается, остальные каналы привлечения стабильны.

3)Не окупаются клиенты привлеченные из FaceBoom, TipTop и AdNonSense.

4)ROI других каналов значительно лушче чем у вышеперечисленных.

Окупается ли реклама, направленная на привлечение пользователей в целом? В целом реклама не окупается, ROI за исследуемый период не преодолевает границу окупаемости.

Какие устройства, страны и рекламные каналы могут оказывать негативное влияние на окупаемость рекламы? Устройства: Iphone и Mac, Страны: США, Реклманые каналы:FaceBoom, TipTop и AdNonSense.

Чем могут быть вызваны проблемы окупаемости? Большая часть рекламного бюджета(80> %) используется для привлечения пользователей из Америки. Несмотря на высокую долю покупателей в регионе(5% в Европе 8% в Америке), их активность не окупает расходы на рекламу.

```
users.query('region == "United States"').groupby('channel').sum()
```

channel	user_id	payer	acquisition_cost
FaceBoom	14606558389460580	3557	32445.60
MediaTornado	2217083852497856	156	954.48
RocketSuperAds	2231345012998483	352	1833.00
TipTop	9708289783241686	1878	54751.30
YRabbit	2143810696485552	165	944.22
organic	19043028301868244	794	0.00

Напишите выводы

- Выделите причины неэффективности привлечения пользователей.
- Сформулируйте рекомендации для отдела маркетинга.

Самая большая доля платящих пользователей из Америки, однако и привлечение подобных пользователей с помощью TipTop стоит слишком дорого, нивелируя их покупки. Возможно следует рассмотреть оставшиеся страны, с меньшей долей платящих, но и с меньшей стоимостью привлечения.

Самая большая доля платящих пользователей у Mac(6,36%) и Iphone(6.2%), однако сложно сказать связано ли это с популярностью и распространённостью этих устройств в Америке, или с другим фактором.

Более половины средств, выделенных на рекламу направленны в TipTop, реклама в котором не окупается. Количество рекламных вложений в источники(FaceBoom, TipTop) постоянно растет. В то же время цена привлечения одного пользователя выше всего в вышеупомянутых TipTop, FaceBoom.

На графике динамики CAC, а также на графике динамики ROI, в период с начала июня до середины июня можно увидеть рост окупаемости. На графике распределения трат по неделям именно в этот период сокращается объём закупки рекламы в источники(FaceBoom, TipTop).

Не окупаются только пользователи из США, привлеченные после середины лета, в период, когда общая сумма закупаемой рекламы в регионе начала значительно увеличиваться. Пользователи покупают меньше, чем стоит их привлечение.

Следует отказаться от рекламы в TipTop, FaceBoom и AdNonSense(компания сразу выйдет в плюс), и попытаться получить пользователей из Америки через другие, более дешевые источники, даже если доля потенциальных покупателей среди них будет меньше. Следует оценить каналы со стабильно высоким ROI (YRabbit, MediaTornado, RocketSuperAds).