Оценка окупаемости рекламы приложения Procrastinate Pro+, поиск причин убытков

Описание проекта

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

В ходе 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.

В плане:

- изучить первоначальные данные и выполнить предобработку: привести к удобному для расчетов виду, удалить пропуски и дубликаты при их наличии;
- подобрать функции для проведения маркетингового исследования;
- изучить профили пользователей: зависимость их заинтересованности от используемых устройств, каналов привлечения и региона проживания;
- проанализировать массив с данными о расходах на маркетинг;
- оценить окупаемость рекламы по регионам, по устройствам и каналам привлечения;
- выявить причины убытков и определить перспективные направления.

Задача — разобраться в причинах убытков и помочь компании выйти в плюс

Загрузка данных и подготовка их к анализу

Загрузим данные о визитах, заказах и рекламных расходах из CSV-файлов в переменные. Изучим данные и выполним предобработку. Проверим, есть ли в данных пропуски и дубликаты. Убедимся, что типы данных во всех колонках соответствуют сохранённым в них значениям. Обратим внимание на столбцы с датой и временем.

Импорт библиотек.

In [1]:
```
import pandas as pd
from datetime import datetime, timedelta
import numpy as np
import seaborn as sbn
```



```
import matplotlib.pyplot as plt
from scipy import stats as st
```

Загрузка данных.

In [2]:
```
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')
```

Выведем первые строки массивов для ознакомления.

In [3]:
```
pd.set_option('display.max_columns', None)
visits.head()
#orders.head()
#costs.head()
```

Out[3]:

	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

Создадим функцию для вывода общей информации по датафрейму: размер, типы данных и количество пропусков в солбцах, наличие дубликатов, наименования столбцов отдельно:

In [4]:
```
def describe(df: pd.DataFrame):
    display(f'Общая информация: {df.shape}')
    display()
    display(df.info())
    display(df.duplicated().sum())
    display(df.isna().mean())
    display(f'Названия столбцов: {df.columns}')
```

In [5]:
```
describe(visits)
```

```
'Общая информация: (309901, 6)'
<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
None
0
User Id          0.0
Region           0.0
Device           0.0
Channel          0.0
```

```
Session Start    0.0
Session End      0.0
dtype: float64
"Названия столбцов: Index(['User Id', 'Region', 'Device', 'Channel', 'Session Star
t',\n       'Session End'],\n      dtype='object')"
```

In [6]:
```
describe(orders)
```

```
'Общая информация: (40212, 3)'
<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
None
0
User Id     0.0
Event Dt    0.0
Revenue     0.0
dtype: float64
"Названия столбцов: Index(['User Id', 'Event Dt', 'Revenue'], dtype='object')"
```

In [7]:
```
describe(costs)
```

```
'Общая информация: (1800, 3)'
<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
None
0
dt         0.0
Channel    0.0
costs      0.0
dtype: float64
"Названия столбцов: Index(['dt', 'Channel', 'costs'], dtype='object')"
```

Для начала переименуем столбцы массивов в стиле snake.

In [8]:
```
visits.rename(columns={'User Id': 'user_id', 'Region': 'region', 'Device': 'device',
visits.columns
```

Out[8]:
```
Index(['user_id', 'region', 'device', 'channel', 'session_start',
       'session_end'],
      dtype='object')
```

In [9]:
```
orders.rename(columns={'User Id': 'user_id', 'Event Dt': 'event_dt', 'Revenue': 'rev
orders.columns
```

Out[9]:
```
Index(['user_id', 'event_dt', 'revenue'], dtype='object')
```


In [10]:
```
costs.rename(columns={'Channel': 'channel'}, inplace=True)
costs.columns
```

Out[10]:
```
Index(['dt', 'channel', 'costs'], dtype='object')
```

Приведем данные в столбцах массивов с датами к формату datetime.

In [11]:
```
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
```

In [12]:
```
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  datetime64[ns]
 5   session_end    309901 non-null  datetime64[ns]
dtypes: datetime64[ns](2), int64(1), object(3)
memory usage: 14.2+ MB
```

ВЫВОД:

После загрузки данных о визитах, заказах и рекламных расходах из CSV-файлов в переменные были выполнены операции по их предобработке для дальнейшей работы. В ходе изучения данных не было выявлено пропусков и дубликатов в данных. Столбцы массивов были переименованы в едином стиле. Данные в столбцах с датами переведены в формат для работы с датой и временем.

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

Используем готовые функции.

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.

In [13]:

```
# функция для создания пользовательских профилей

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
```

In [14]:

```
# 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
```

In [15]:

```
# функция для расчёта конверсии

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

    # исключаем пользователей, не «доживших» до горизонта анализа
```


27.04.2023, 09:35 1c96fee4-af67-4c6f-bc39-6dcac175a197

```
    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
```


In [16]:

```
# 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
    )
```

In [17]:

```
# 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
```

In [18]:

```
# 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()
```

In [19]:

```
# 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()
```

In [20]:

```
# 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()
```

ВЫВОД:

Созданы 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.

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.

Получим профили пользователей. Для этого вызовем функцию get_profiles(), передав ей данные о посещениях, покупках и тратах на рекламу.

In [21]:
```
profiles = get_profiles(visits, orders, costs)
profiles.head(5)
```

Out[21]:

	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	2019-05-01	True	1.088172
1	4919697	2019-07-09 12:46:07	FaceBoom	iPhone	United States	2019-07-09	2019-07-01	False	1.107237
2	6085896	2019-10-01 09:58:33	organic	iPhone	France	2019-10-01	2019-10-01	False	0.000000
3	22593348	2019-08-22 21:35:48	AdNonSense	PC	Germany	2019-08-22	2019-08-01	False	0.988235
4	31989216	2019-10-02 00:07:44	YRabbit	iPhone	United States	2019-10-02	2019-10-01	False	0.230769

Определим самую раннюю и самую позднюю даты привлечения пользователей:

In [22]:
```
min = profiles['first_ts'].dt.date.min()
display('Минимальная дата привлечения пользователей {}'.format(min))
```

```
'Минимальная дата привлечения пользователей 2019-05-01'
```


In [23]:
```
max = profiles['first_ts'].dt.date.max()
display('Максимальная дата привлечения пользователей {}'.format(max))
```

```
'Максимальная дата привлечения пользователей 2019-10-27'
```

Видим, что у нас имеются данные с начала мая 2019 года примерно по конец октября этого же года. Около 5 месяцев.

Выясним, из каких стран пользователи приходят в приложение и на какую страну приходится больше всего платящих пользователей.

In [24]:
```
# пользователи по странам
prof1 = profiles.groupby('region').agg({'user_id': 'nunique'})
prof1
```

Out[24]:

region	user_id
France	17450
Germany	14981
UK	17575
United States	100002

In [25]:
```
# плательщики по странам
prof2 = profiles.query('payer == True').groupby('region').agg({'user_id': 'nunique'}
prof2
```

Out[25]:

region	user_id
France	663
Germany	616
UK	700
United States	6902

Построим таблицу, отражающую количество пользователей и долю платящих из каждой страны.

In [26]:
```
prof1['payer_part'] = (prof2['user_id']/prof1['user_id']*100).round(2)
prof1.sort_values(by='payer_part', ascending = False)
```

Out[26]:

region	user_id	payer_part
United States	100002	6.90
Germany	14981	4.11
UK	17575	3.98
France	17450	3.80

Выяснилось, что больше всего пользователей Procrastinate Pro+ родом из США, они же и самые платежеспособные. Стоит обратить внимание на пользователей из Германии, при наименьшем количестве - у них доля плятящих за развлечения на втором месте.

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

In [27]:
```
# пользователи по устройствам
device_grouped1 = profiles.pivot_table(index=['device'], values='user_id', aggfunc='
device_grouped1
```

Out[27]:

	device	user_id
0	Android	35032
1	Mac	30042
2	PC	30455
3	iPhone	54479

In [28]:
```
# плательщики по устройствам
device_grouped2 = profiles.query('payer == True').pivot_table(index=['device'], valu
device_grouped2.rename(columns={'user_id': 'payer'}, inplace=True)
device_grouped2
```

Out[28]:

	device	payer
0	Android	2050
1	Mac	1912
2	PC	1537
3	iPhone	3382

In [29]:
```
# общая таблица по устройствам
device_grouped = device_grouped1.merge(device_grouped2, on='device', how='left')
device_grouped['payer_part'] = (device_grouped['payer']/device_grouped['user_id']*10
device_grouped.sort_values(by='payer_part', ascending = False)
```

Out[29]:

	device	user_id	payer	payer_part
1	Mac	30042	1912	6.36
3	iPhone	54479	3382	6.21

	device	user_id	payer	payer_part
0	Android	35032	2050	5.85
2	PC	30455	1537	5.05

Установили, что самая высокая доля плательщиков среди пользователей Мак, хотя их численность в общем количестве пользователей - самая маленькая. Пользователи айфонов на первом месте по общему количеству использующих наше приложение и на втором по доле платящих за него.

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

In [30]:
```
# пользователи по каналам
channel1 = profiles.groupby('channel').agg({'user_id': 'nunique'})
channel1
```

Out[30]:

channel	user_id
AdNonSense	3880
FaceBoom	29144
LeapBob	8553
MediaTornado	4364
OppleCreativeMedia	8605
RocketSuperAds	4448
TipTop	19561
WahooNetBanner	8553
YRabbit	4312
lambdaMediaAds	2149
organic	56439

In [31]:
```
# плательщики по каналам
channel2 = profiles.query('payer == True').groupby('channel').agg({'user_id': 'nuniq
channel2
```

Out[31]:

channel	user_id
AdNonSense	440
FaceBoom	3557
LeapBob	262
MediaTornado	156
OppleCreativeMedia	233

channel	user_id
RocketSuperAds	352
TipTop	1878
WahooNetBanner	453
YRabbit	165
lambdaMediaAds	225
organic	1160

Общая таблица пользователей и доли плательщиков по каналам привлечения:

In [32]:
```
channel1['payer_part'] = (channel2['user_id']/channel1['user_id']*100).round(2)
channel1.sort_values(by='payer_part', ascending = False)
```

Out[32]:

channel	user_id	payer_part
FaceBoom	29144	12.20
AdNonSense	3880	11.34
lambdaMediaAds	2149	10.47
TipTop	19561	9.60
RocketSuperAds	4448	7.91
WahooNetBanner	8553	5.30
YRabbit	4312	3.83
MediaTornado	4364	3.57
LeapBob	8553	3.06
OppleCreativeMedia	8605	2.71
organic	56439	2.06

Наибольшее количество пользователей приложения органического происхождения, как и следовало ожидать, среди них меньше всего платных пользователей. На втором месте по количеству пользователей и на первом по доле платных пользователей - источник FaceBoom. Стоит обратить внимание на рекламу в таких источниках, как: AdNonSense и lambdaMediaAds!!! Так как при наименьшем количестве пользователей, пришедших из них, доля платящих пользователей там на 2 и 3 местах после FaceBoom. Наименьшее количество плательщиков привлекает канал LeapBob при не самом маленьком количестве пользователей(4 место).

ВЫВОДЫ:

По результатам анализа пользовательских профилей за 5 месяцев можно посоветовать уделить больше внимания рекламе приложения в Германии (большая доля пользователей этой страны готова платить за приложение, хотя самих

пользователей в 6 раз меньше, чем в США), не забывая при этом про США - главный источних пользователей и прямо пропорционально - плательщиков.

Что касается устройств: самой привлекательной целевой аудиторией являются пользователи MAC. При почти в 2 раза меньшем количестве, чем пользователей с айфонами, среди них самая высокая доля готовых платить!

Каналы для размещения рекламы: лучшие поставщики клиентов - FaceBoom, TipTop. НО стоит больше внимания уделить AdNonSense и lambdaMediaAds - привлекают сейчас наименьшее количество пользователей, но доля плательщиков из них суммарно равна доле плательщиков по источникам-лидерам.

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. Используем профили пользователей.

Общие расходы на маркетинг посчитаем, суммируя столбец costs одноименного массива данных:

In [33]:
```
total_market = costs['costs'].sum().round(0)
display(f'Общие затраты на маркетинг: {total_market}')
```

```
'Общие затраты на маркетинг: 105497.0'
```

Распределим расходы по каналам привлечения:

In [34]:
```
costs_channel = costs.groupby('channel')['costs'].sum().reset_index()
costs_channel['part'] = (costs_channel['costs']/total_market*100).round(2)
costs_channel.sort_values(by='costs',ascending = False)
```

Out[34]:

	channel	costs	part
6	TipTop	54751.30	51.90
1	FaceBoom	32445.60	30.75
7	WahooNetBanner	5151.00	4.88
0	AdNonSense	3911.25	3.71
4	OppleCreativeMedia	2151.25	2.04
5	RocketSuperAds	1833.00	1.74
2	LeapBob	1797.60	1.70
9	lambdaMediaAds	1557.60	1.48
3	MediaTornado	954.48	0.90
8	YRabbit	944.22	0.90

Больше 80 % расходов приходится на каналы TipTop и FaceBoom, которые хоть и привлекают большую часть пользователей, но плательщиков привлекают не больше, чем недооцененные AdNonSense и lambdaMediaAds. Возможно стоит перераспределить рекламный бюджет.

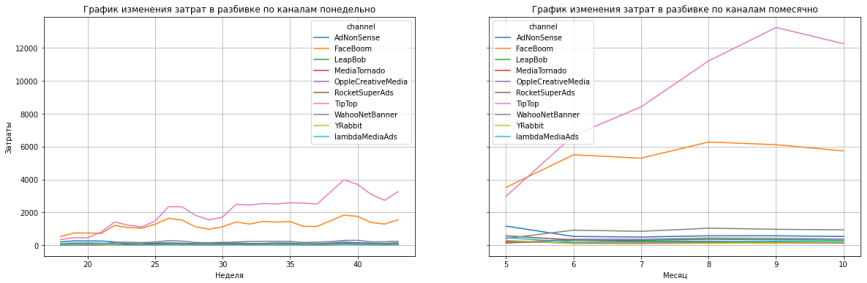
Визуализируем динамику изменения расходов во времени (по неделям и месяцам) по каждому источнику:

In [35]:
```
# добавим столбца с неделями и месяцами
costs['week'] = pd.to_datetime(costs['dt']).dt.isocalendar().week
costs['month'] = pd.to_datetime(costs['dt']).dt.month
costs.head()
```

Out[35]:

	dt	channel	costs	week	month
0	2019-05-01	FaceBoom	113.3	18	5
1	2019-05-02	FaceBoom	78.1	18	5
2	2019-05-03	FaceBoom	85.8	18	5
3	2019-05-04	FaceBoom	136.4	18	5
4	2019-05-05	FaceBoom	122.1	18	5

In [36]:
```
plt.figure(figsize=(20, 6))

ax1 = plt.subplot(1, 2, 1)
costs.pivot_table(
        index='week', columns='channel', values='costs', aggfunc='sum'
    ).plot(grid=True, ax=ax1)

ax2 = plt.subplot(1, 2, 2, sharey=ax1)
costs.pivot_table(
        index='month', columns='channel', values='costs', aggfunc='sum'
    ).plot(grid=True, ax=ax2)

ax1.set_title('График изменения затрат в разбивке по каналам понедельно')
ax2.set_title('График изменения затрат в разбивке по каналам помесячно')
ax1.set_xlabel('Неделя')
ax2.set_xlabel('Месяц')
ax1.set_ylabel('Затраты')
ax2.set_ylabel('Затраты')
plt.show()
```



Видим, что средства затрачивались только на рекламу двух каналов TipTop и FaceBoom, причем эти расходы резко возрастали до сентября по TipTop и плавно росли до августа по

FaceBoom. По остальным каналам расходы практически не менялись во времени.

Узнаем, сколько в среднем стоило привлечение одного пользователя (CAC) из каждого источника. Используем профили пользователей. Исключим из профилей органических пользователей, т.к. на их привлечение ничего не затрачено.

In [37]:
```
profiles2 = profiles.query('channel != "organic"')
profiles2.groupby('channel')['acquisition_cost'].mean().round(2).sort_values(ascendi
```

Out[37]:
```
channel
TipTop               2.80
FaceBoom             1.11
AdNonSense           1.01
lambdaMediaAds       0.72
WahooNetBanner       0.60
RocketSuperAds       0.41
OppleCreativeMedia   0.25
MediaTornado         0.22
YRabbit              0.22
LeapBob              0.21
Name: acquisition_cost, dtype: float64
```

По данным таблицы видно, что самые высокие расходы на привлечение пользователя по каналам TipTop и FaceBoom, пользователи, привлеченные AdNonSense и lambdaMediaAds обходятся в 3 раза дешевле, чем пользователи с TipTop, но почти столько же, как и FaceBoom. Стоит сильно сократить затраты на TipTop. Похоже они не окупаются, уточним далее. И подумать о большем привлечении пользователей с совсем недорогих каналов: MediaTornado, YRabbit, LeapBob.

In [38]:
```
cac = profiles2['acquisition_cost'].mean().round(2)
display(f'Средняя стоимость привлечения одного пользователя (CAC) всего проекта: {ca
```

```
'Средняя стоимость привлечения одного пользователя (CAC) всего проекта: 1.13'
```

Еще раз убеждаемся, что расходы на привлечение пользователей канала TipTop больше, чем в два раза, средних расходов на пользователя по всем каналам привлечения.

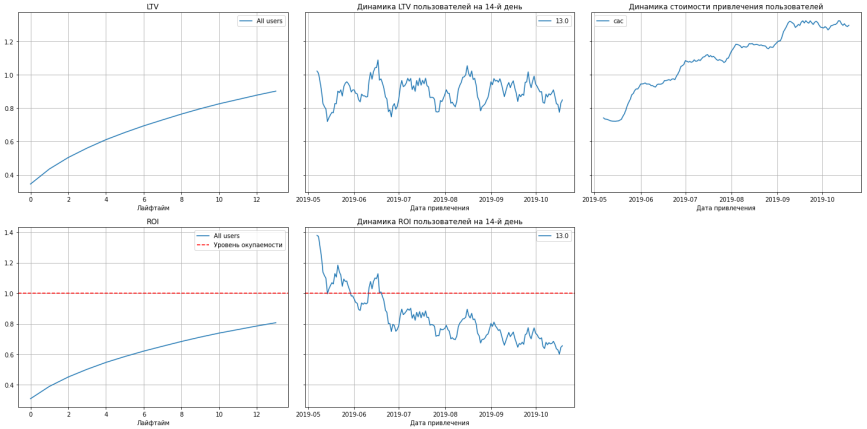
Оценка окупаемости рекламы

Используя графики LTV, ROI и CAC, проанализируем окупаемость рекламы. На календаре 1 ноября 2019 года, а в бизнес-плане заложено, что пользователи должны окупаться не позднее чем через две недели после привлечения.

- 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. Постройте графики LTV и ROI, а также графики динамики LTV, CAC и ROI.

Посмотрим на окупаемость рекламы. Рассчитаем и визуализируем LTV и ROI, вызвав функции get_ltv() и plot_ltv_roi(). На входе функции get_ltv() используем данные массива profiles2, чтобы исключить затраты на органических пользователей.

In [39]:
```
observation_date = datetime(2019, 11, 1).date()
horizon_days = 14
# считаем LTV и ROI
ltv_raw, ltv_grouped, ltv_history, roi_grouped, roi_history = get_ltv(
    profiles2, orders, observation_date, horizon_days
)

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

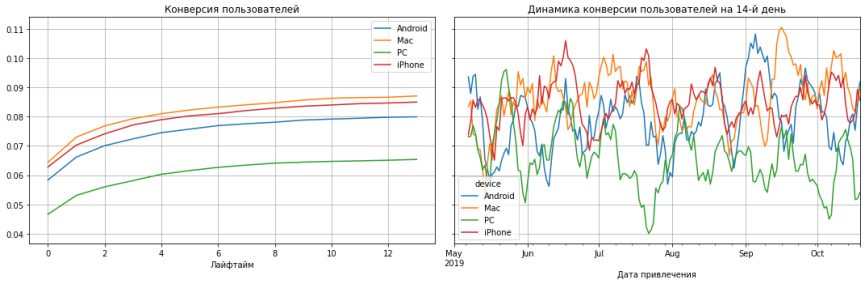


Реклама к концу второй недели горизонта анализа окупается только на 80% (ROI). При этом стоимость привлечения пользователей растет каждый месяц. LTV достаточно стабилен. Значит, дело не в ухудшении качества пользователей. Судя по уменьшению ROI 14-го дня ниже уровня окупаемости, начиная с июня, затраты на рекламу растут быстрее, чем LTV пользователей.

Проверим конверсию пользователей и динамику её изменения. То же самое сделаем с удержанием пользователей. Построим и изучим графики конверсии и удержания. Будем использовать массив профилей пользователей без органического привлечения.

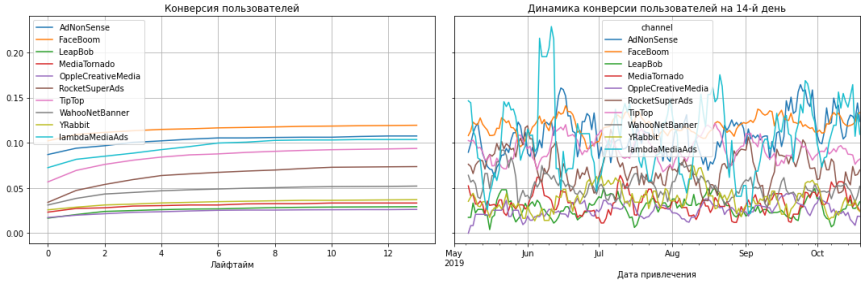
Посчитаем и визуализируем конверсию, вызвав функции get_conversion() и plot_conversion().

In [40]:
```
dimensions = ['device']
conversion_raw, conversion_grouped, conversion_history = get_conversion(
    profiles2, orders, observation_date, horizon_days, dimensions=dimensions)

plot_conversion(conversion_grouped, conversion_history, horizon_days)
```

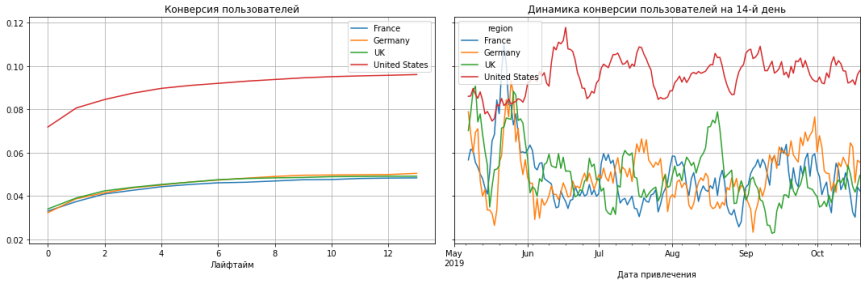
Судя по графикам конверсия пользователей довольно стабильна, наблюдаются сезонные перепады. Самая стабильная и высокая конверсия у пользователей MAC и айфонов. Немного отстает и менее стабилен Андроид. А вот пользоватили PC конвертируются хуже всех остальных.

In [41]:
```
dimensions = ['channel']
conversion_raw, conversion_grouped, conversion_history = get_conversion(
    profiles2, orders, observation_date, horizon_days, dimensions=dimensions)

plot_conversion(conversion_grouped, conversion_history, horizon_days)
```



Еще раз убедились, что самая высокая конверсия у пользователей, привлеченных FaceBoom, AdNonSense, lambdaMediaAds. TipTop замыкает четверку лидеров, у которых итоговая конверсия выше 8%. При этом конверсия по каналам AdNonSense и lambdaMediaAds отличается большими перепадами.

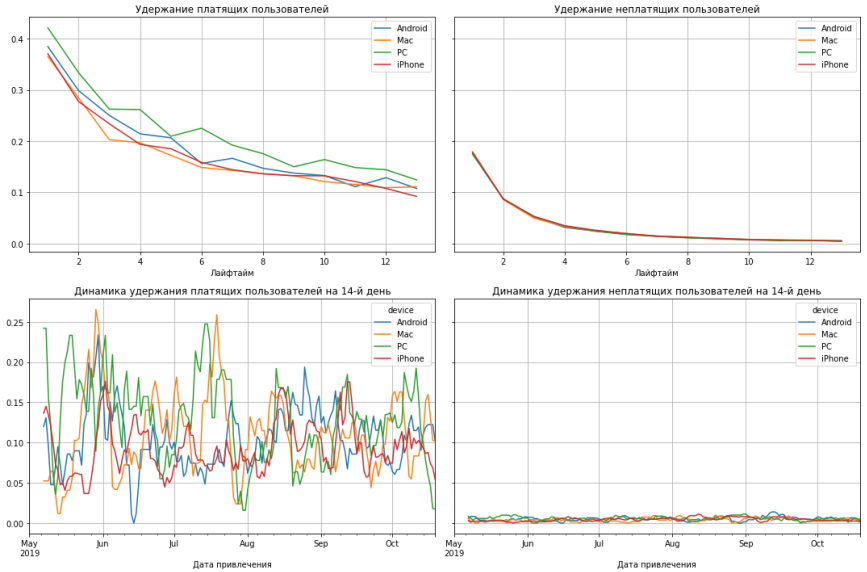
In [42]:
```
dimensions = ['region']
conversion_raw, conversion_grouped, conversion_history = get_conversion(
    profiles2, orders, observation_date, horizon_days, dimensions=dimensions)

plot_conversion(conversion_grouped, conversion_history, horizon_days)
```



Еще раз подтвердили данные, полученные в таблицах выше. Самая высокая и стабильная конверсия у пользователей из США, что скорее всего связано с их наибольшей долей в общем количестве. Из остальных стран с примерно равным количеством пользователей конверсия наилучшая у пользователей Германии.

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

In [43]:
```
dimensions = ['device']
retention_raw, retention_grouped, retention_history = get_retention(
    profiles2, visits, observation_date, horizon_days, dimensions=dimensions)

plot_retention(retention_grouped, retention_history, horizon_days)
```

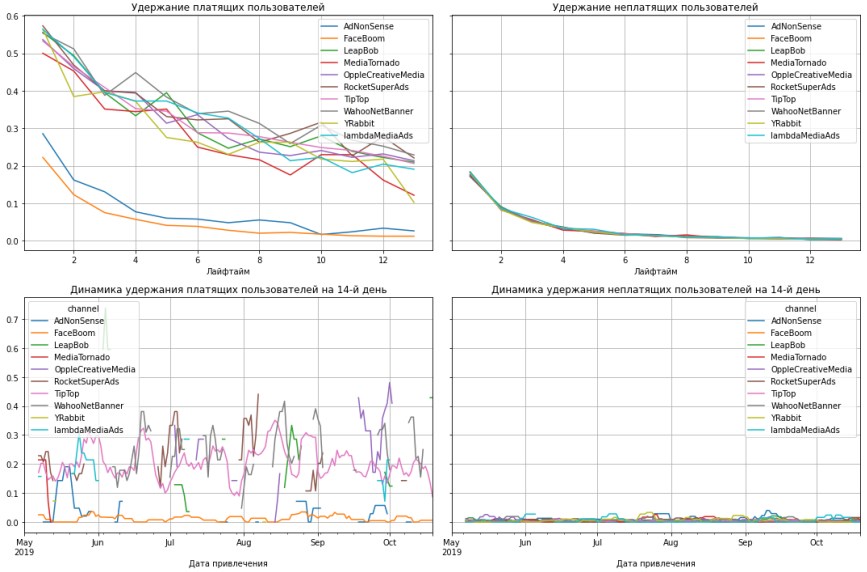


А вот удержание платящих пользователей лучше всего по устройствам PC, у пользователей которых самая низкая конверсия. Возможно, с остальных устройств не так удобно пользоваться приложением? Удержание неплатящих пользователей примерно одинаковое.

In [44]:
```
dimensions = ['channel']
retention_raw, retention_grouped, retention_history = get_retention(
    profiles2, visits, observation_date, horizon_days, dimensions=dimensions)
```

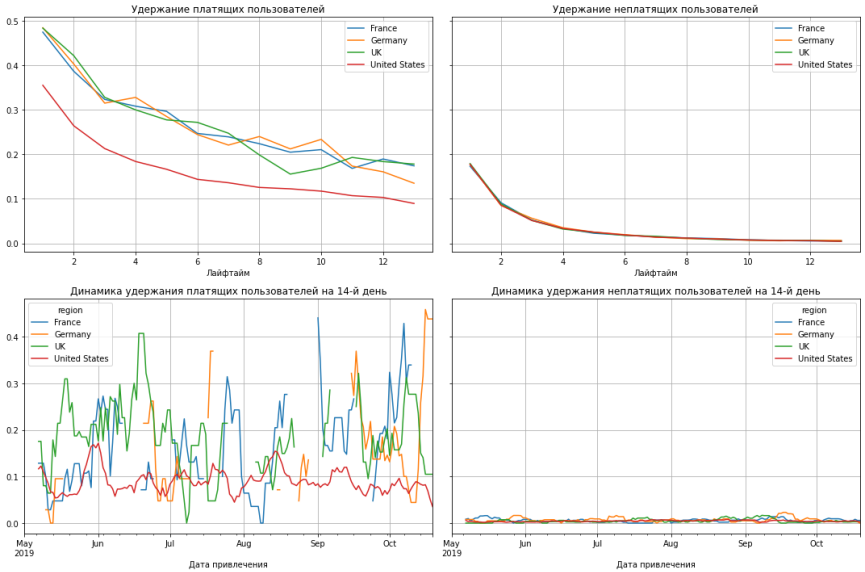


```
plot_retention(retention_grouped, retention_history, horizon_days)
```



Еще раз убедились в том, что вкладывать так много средств в канал FaceBoom не стоит, самое низкое удержание платящих пользователей! Также плохое удержание у еще одного не самого дешевого канала AdNonSense.

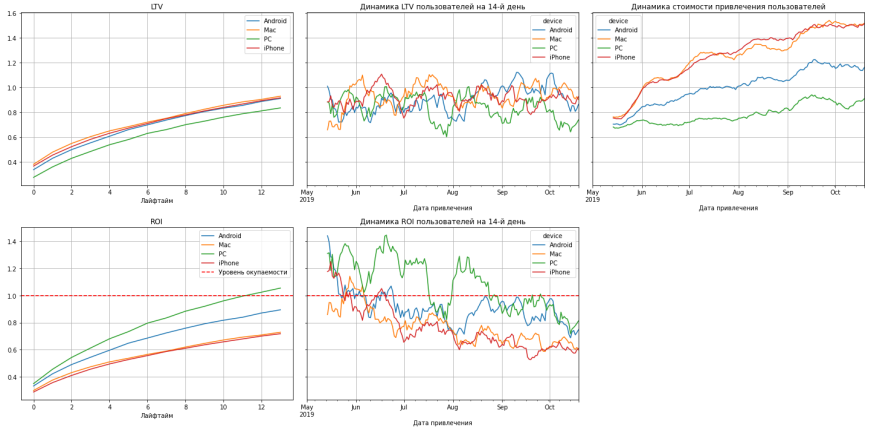
In [45]:
```
dimensions = ['region']
retention_raw, retention_grouped, retention_history = get_retention(
    profiles2, visits, observation_date, horizon_days, dimensions=dimensions)

plot_retention(retention_grouped, retention_history, horizon_days)
```



Еще одно проблемное место: при самой высокой конверсии - самое низкое удержание платящих пользователей из США!

Теперь проанализируем окупаемость рекламы с разбивкой по устройствам. Построим графики LTV и ROI, а также графики динамики LTV, CAC и ROI.

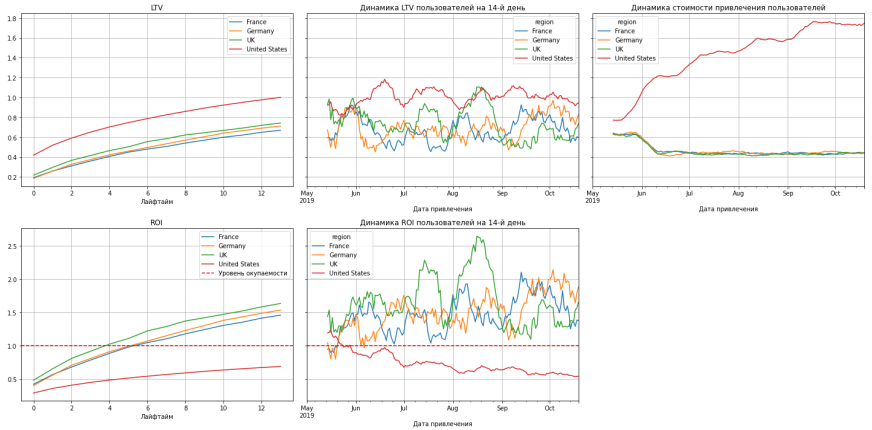
In [46]:
```
# смотрим окупаемость с разбивкой по устройствам

dimensions = ['device']

ltv_raw, ltv_grouped, ltv_history, roi_grouped, roi_history = get_ltv(
    profiles2, orders, observation_date, horizon_days, dimensions=dimensions
)

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

Окупаемость рекламы достигается к концу второй недели только по PC. По нему же и наименьшие затраты на привлечение пользователей.

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

In [47]:
```
# смотрим окупаемость с разбивкой по странам

dimensions = ['region']

ltv_raw, ltv_grouped, ltv_history, roi_grouped, roi_history = get_ltv(
    profiles2, orders, observation_date, horizon_days, dimensions=dimensions
)

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



Очень дорого нам обходятся и совсем не окупаются пользователи из США. Похоже именно они и портят всю статистику по окупаемости рекламы. При хорошем качестве клиенов (высокий LTV и хорошая конверсия) большие затраты на них не окупаются.

In [48]:
```
profiles3 = profiles2.query('region == "United States"')
profiles3.groupby('channel')['user_id'].nunique().sort_values(ascending=False)
```

Out[48]:
```
channel
FaceBoom          29144
TipTop            19561
RocketSuperAds     4448
MediaTornado       4364
YRabbit            4312
Name: user_id, dtype: int64
```

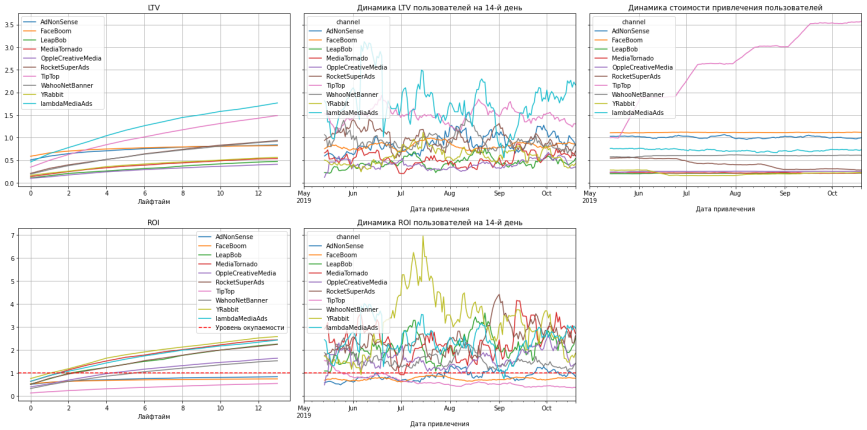
In [49]:
```
profiles3.groupby('device')['user_id'].nunique().sort_values(ascending=False)
```

Out[49]:
```
device
iPhone     27548
Mac        15424
Android    12436
PC          6421
Name: user_id, dtype: int64
```

Видим, что большая часть пользователей США привлечены наиболее популярными каналами FaceBoom и TipTop. И почти половина заходит с Айфонов.

Проанализируем окупаемость рекламы с разбивкой по рекламным каналам. Постройте графики LTV и ROI, а также графики динамики LTV, CAC и ROI.

In [50]:
```
# смотрим окупаемость с разбивкой по источникам привлечения

dimensions = ['channel']

ltv_raw, ltv_grouped, ltv_history, roi_grouped, roi_history = get_ltv(
    profiles2, orders, observation_date, horizon_days, dimensions=dimensions
)

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



Не окупаются расходы на пользователей, привлеченных с каналов FaceBoom, AdNonSense, TipTop. При постоянно растущих расходах на TipTop и стабильно высоких на два остальных канала.

ВЫВОД:

Реклама, направленная на привлечение пользователей в целом, не окупается. Негативно сказывается на окупаемости вложение средств в пользователей устройств Mac и Айфон (возможно, есть проблемы с использованием приложения на этих устройствах). Что подтверждается и более низким уровнем удержания платящих пользователей при хорошей конверсии.

Также не окупаются и высокие расходы на рекламу в США, при хорошем качестве пользователей, очень низкое удержание платящих.

И похоже провальным было решение вкладывать большую часть средств в рекламу на каналах FaceBoom, AdNonSense, TipTop, большая стоимость привлечения пользователей и дорогой трафик при отсутствии окупаемости и очень низком уровне удержания платящих пользователей каналом FaceBoom.

Выводы

После загрузки данных о визитах, заказах и рекламных расходах из CSV-файлов в переменные были выполнены операции по их предобработке для дальнейшей работы. В ходе изучения данных не было выявлено пропусков и дубликатов в данных. Столбцы массивов были переименованы в едином стиле. Данные в столбцах с датами переведены в формат для работы с датой и временем.

Созданы 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.

Наибольшее количество пользователей приложения органического происхождения, как и следовало ожидать, среди них меньше всего платных пользователей. На втором месте по количеству пользователей и на первом по доле платных пользователей - источник FaceBoom. Стоит обратить внимание на рекламу в lambdaMediaAds, т.к. как при наименьшем количестве пользователей, доля платящих пользователей там на 3 местах после FaceBoom и AdNonSense. Наименьшее количество плательщиков привлекает канал LeapBob при не самом маленьком количестве пользователей(4 место).

По результатам анализа пользовательских профилей за 5 месяцев можно посоветовать уделить больше внимания рекламе приложения в Германии (большая доля пользователей этой страны готова платить за приложение, хотя самих пользователей в 6 раз меньше, чем в США).

Что касается устройств: самой привлекательной целевой аудиторией являются пользователи MAC. При почти в 2 раза меньшем количестве, чем пользователей с айфонами, среди них самая высокая доля готовых платить!

Каналы для размещения рекламы: лучшие поставщики клиентов - FaceBoom, TipTop. НО! По данным дальнейшего анализа видно, что самые высокие расходы на привлечение пользователей по каналам TipTop и FaceBoom, AdNonSense себя не оправдали! Стоит сильно сократить затраты на них. И подумать о большем привлечении пользователей с менее дорогих каналов: lambdaMediaAds, MediaTornado, YRabbit.

Реклама, направленная на привлечение пользователей в целом, не окупается. Негативно сказывается на окупаемости вложение средств в пользователей устройств Mac и Айфон (возможно, есть проблемы с использованием приложения на этих устройствах). Что подтверждается и более низким уровнем удержания платящих пользователей при хорошей конверсии. Также не окупаются и высокие расходы на рекламу в США, при хорошем качестве пользователей, очень низкое удержание платящих. Что интересно, большая часть пользователей из США пользуются Айфонами и привлечены с каналов с дорогим трафиком и низкой окупаемостью FaceBoom и TipTop. И похоже провальным было решение вкладывать большую часть средств в рекламу на каналах FaceBoom, AdNonSense, TipTop, большая стоимость привлечения пользователей и дорогой трафик при отсутствии окупаемости и низком уровне удержания платящих.

Рекомендации для отдела маркетинга: - искать новые каналы с низкой стоимостью трафика, обратить внимание на lambdaMediaAds, MediaTornado, YRabbit; - расследовать, нет ли технических проблем у пользователей Мак и айфон, привлекать больше пользователей Андроидов и PC (реклама окупается уже в конце первой недели); - сделать своей целью привлечение аудитории из Германии, Англии и Франции; - понять причину низкого интереса к приложению платящих пользователей из США.