Инструкция по выполнению проекта:

Вы - маркетинговый аналитик развлекательного приложения Procrastinate Pro+. Несколько прошлых месяцев ваш бизнес постоянно нес убытки - в привлечение пользователей была вложена куча денег, а толку никакого. Вам нужно разобраться в причинах этой ситуации.

У вас в распоряжении есть лог сервера с данными о посещениях приложения новыми пользователями, зарегистрировавшимися в период с 2019-05-01 по 2019-10-27, выгрузка их покупок за этот период, а также статистика рекламных расходов. Вам предстоит изучить, как люди пользуются продуктом, когда они начинают покупать, сколько денег приносит каждый клиент, когда он окупается и какие факторы отричательно влияют на привлечение пользователей.

Шаг 1. Загрузите данные и подготовьте их к анализу

Загрузите данные о визитах, заказах и расходах в переменные. Оптимизируйте данные для анализа. Убедитесь, что тип данных в каждой колонке — правильный. Путь к файлам:

- /datasets/visits_info_short.csv. Скачать датасет
- /datasets/orders_info_short.csv. Скачать датасет
- /datasets/costs_info_short.csv. Скачать датасет

Шаг 2. Задайте функции для расчета и анализа LTV, ROI, удержания и конверсии

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

Шаг 3. Проведите исследовательский анализ данных

Постройте профили пользователей. Определите минимальную и максимальную дату привлечения пользователей.

Выясните:

- Из каких стран приходят посетители? Какие страны дают больше всего платящих пользователей?
- Какими устройствами они пользуются? С каких устройств чаще всего заходят платящие пользователи?
- По каким рекламным каналам шло привлечение пользователей? Какие каналы приносят больше всего платящих пользователей?.

Шаг 4. Маркетинг

Выясните:

- Сколько денег потратили? Всего / на каждый источник / по времени
- Сколько в среднем стоило привлечение одного покупателя из каждого источника?

Шаг 5. Оцените окупаемость рекламы для привлечения пользователей

С помощью LTV и ROI:

- Проанализируйте общую окупаемость рекламы;
- Проанализируйте окупаемость рекламы с разбивкой по устройствам;
- Проанализируйте окупаемость рекламы с разбивкой по странам;
- Проанализируйте окупаемость рекламы с разбивкой по рекламным каналам.

Опишите проблемы, которые вы обнаружили. Ответьте на вопросы:

- Окупается ли реклама, направленная на привлечение пользователей в целом?
- Какие устройства, страны и рекламные каналы могут оказывать негативное влияние на окупаемость рекламы?
- Чем могут быть вызваны проблемы окупаемости? Изучите конверсию и удержание с разбивкой по устройствам, странам, рекламным каналам.

Опишите возможные причины обнаруженных проблем и сформируйте рекомендации для рекламного отдела. При решении этого шага считайте, что вы смотрите данные 1-го ноября 2019 года и что в вашей организации принято считать, что окупаемость должна наступать не позднее, чем через 2 недели после привлечения пользователей.

Подумайте, нужно ли включать в анализ органических пользователей?

Шаг 6. Напишите выводы

- Выделите причины неэффективности привлечения пользователей;
- Сформируйте рекомендации для отдела маркетинга для повышения эффективности.

Оформление:

Задание выполните в Jupyter Notebook. Программный код заполните в ячейках типа code, текстовые пояснения — в ячейках типа markdown. Примените форматирование и заголовки.

Описание данных

Таблица visits_log_short (лог сервера с информацией о посещениях сайта):

```
User Id — уникальный идентификатор пользователя
Device — категория устройства пользователя
Session start — дата и время начала сессии
Session End — дата и время окончания сессии
Channel — идентификатор рекламного источника, из которого пришел пользователь
Region - страна пользователя
```

Таблица orders_log_short (информация о заказах):

```
User Id — уникальный id пользователя, который сделал заказ
Event Dt — дата и время покупки
Revenue — выручка
```

Таблица costs_short (информация о затратах на маркетинг):

```
Channel — идентификатор рекламного источника
Dt — дата
Costs — затраты на этот рекламный источник в этот день
```

Самостоятельный проект

Загрузка данных и подготовка их к анализу

Загрузим данные о визитах, заказах и расходах в переменные. Оптимизируем данные для анализа. Убедимся, что тип данных в каждой колонке — правильный. Путь к файлам:

- /datasets/visits_info_short.csv.
- /datasets/orders_info_short.csv.
- /datasets/costs_info_short.csv.

Загружаем данные

In [1]:
```
import pandas as pd
import numpy as np
from datetime import datetime, timedelta
from matplotlib import pyplot as plt
%config InlineBackend.figure_format = 'retina'
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
visits, orders, costs = (
                          pd.read_csv('/datasets/visits_info_short.csv'),# журнал сессий
                          pd.read_csv('/datasets/orders_info_short.csv'),  # покупки
                          pd.read_csv('/datasets/costs_info_short.csv'),  # траты на рекламу
)
```

Предобработка данных

- В нашем распоряжении три датасета. Файл visits_info_short.csv хранит лог сервера с информацией о посещениях сайта, orders_info_short.csv — информацию о покупках, а costs_info_short.csv — информацию о расходах на рекламу.
  - Структура visits_info_short.csv
    - User Id — уникальный идентификатор пользователя,
    - Region — страна пользователя,
    - Device — тип устройства пользователя,
    - Channel — идентификатор источника перехода,
    - Session Start — дата и время начала сессии,
    - Session End — дата и время окончания сессии.
  - Структура orders_info_short.csv
    - User Id — уникальный идентификатор пользователя,
    - Event Dt — дата и время покупки,
    - Revenue — сумма заказа.
  - Структура costs_info_short.csv
    - Channel — идентификатор рекламного источника,
    - Dt — дата проведения рекламной кампании,
    - Costs — расходы на эту кампанию.

In [3]:
```
#получаем информацию
visits.info()
```

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

```
dtypes: int64(1), object(5)
memory usage: 14.2+ MB
```

In [4]:
```
#получаем информацию
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
```

In [5]:
```
#получаем информацию
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
```

Видим, что пропусков нет.Но время указано в формате object и названия столбцов указаны неправильно. Исправим.

In [6]:
```
# приведем названия столбцов к нижнему регистру и перемеинуем названия:
visits = visits.rename(columns={'User Id':'User_Id', 'Session Start':'Session_Start', 'Session
visits.columns = map(str.lower, visits.columns)
```

In [7]:
```
visits.columns# проверка результатов - перечень названий столбцов
```

Out[7]:
```
Index(['user_id', 'region', 'device', 'channel', 'session_start',
       'session_end'],
      dtype='object')
```

In [8]:
```
# приведем названия столбцов к нижнему регистру и перемеинуем названия:
orders = orders.rename(columns={'User Id':'User_Id','Event Dt':'Event_Dt'})# переименование сто
orders.columns = map(str.lower, orders.columns)
```

In [9]:
```
orders.columns# проверка результатов - перечень названий столбцов
```

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

In [10]:
```
# приведем названия столбцов к нижнему регистру :
costs.columns = map(str.lower, costs.columns)
```

In [11]:
```
# проверим на дубликаты
visits.duplicated().sum()
```

Out[11]:
```
0
```

In [12]:
```
costs.duplicated().sum()
```

Out[12]: 0

In [13]:
```
orders.duplicated().sum()
```

Out[13]: 0

In [14]:
```
costs.columns# проверка результатов - перечень названий столбцов
```

Out[14]: Index(['dt', 'channel', 'costs'], dtype='object')

In [15]:
```
# преобразование данных о времени
visits['session_start'] = pd.to_datetime(visits['session_start'])
orders['event_dt'] = pd.to_datetime(orders['event_dt'])
costs['dt'] = pd.to_datetime(costs['dt']).dt.date
```

- В процессе предобработки данных были выявлены следующие ошибки:
  - время указано в формате object,
  - названия столбцов указаны неправильно
- Привели названия столбцов к нижнему регистру и перемеиновали.
- Преобразовали данные о времени.
- Проверили на дубликаты. Дубликатов не обнаружено.

Зададим функции для расчета и анализа LTV, ROI, удержания и конверсии

In [16]:
```
# функция для создания пользовательских профилей

def get_profiles(sessions, orders, costs):

    # находим параметры первых посещений
    profiles = (
        visits.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())

    # добавляем флаги для всех событий из event_names
    #for event in event_dt:
     #   if event in events['event_dt'].unique():
      #      profiles[event] = profiles['user_id'].isin(
       #         events.query('event_dt == @event')['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
    costs = costs.merge(new_users, on=['dt', 'channel'], how='left')

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

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

    # стоимость привлечения органических пользователей равна нулю
    profiles['acquisition_cost'] = profiles['acquisition_cost'].fillna(0).round(3)

    return profiles
```

In [17]:

```
# 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 [18]:

```
 # 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 [19]:

```
# функция для расчёта конверсии

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

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

    # определяем дату и время первой покупки для каждого пользователя
    first_purchases = (
```

```
        purchases.sort_values(by=['user_id', 'event_dt'])
        .groupby('user_id')
        .agg({'event_dt': 'first'})
        .reset_index()
    )

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

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

    # группируем по cohort, если в dimensions ничего нет
    if len(dimensions) == 0:
        result_raw['cohort'] = 'All users'
        dimensions = dimensions + ['cohort']
       # функция для группировки таблицы по желаемым признакам
    def group_by_dimensions(df, dims, horizon_days):
        result = df.pivot_table(
            index=dims, columns='lifetime', values='user_id', aggfunc='nunique'
        )
        result = result.fillna(0).cumsum(axis = 1)
        cohort_sizes = (
            df.groupby(dims)
            .agg({'user_id': 'nunique'})
            .rename(columns={'user_id': 'cohort_size'})
        )
        result = cohort_sizes.merge(result, on=dims, how='left').fillna(0)
        # делим каждую «ячейку» в строке на размер когорты
        # и получаем conversion rate
        result = result.div(result['cohort_size'], axis=0)
        result = result[['cohort_size'] + list(range(horizon_days))]
        result['cohort_size'] = cohort_sizes
        return result

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

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

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

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

In [20]:

```
# 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
```

```python
    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 [21]:

```
# 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 [22]:

```
# 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 [23]:

```
# 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 [24]:

```
# 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()
```

Проведем исследовательский анализ данных

Построим профили пользователей. Определим минимальную и максимальную дату привлечения пользователей.

Выясним:

- Из каких стран приходят посетители? Какие страны дают больше всего платящих пользователей?
- Какими устройствами они пользуются? С каких устройств чаще всего заходят платящие пользователи?
- По каким рекламным каналам шло привлечение пользователей? Какие каналы приносят больше всего платящих пользователей?

Получаем профили пользователей

In [25]:
```
# получаем профили пользователей
profiles = get_profiles(visits, orders, costs)
profiles.head()
```

Out[25]:

	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	1.088
1	4919697	2019-07-09 12:46:07	FaceBoom	iPhone	United States	2019-07-09	2019-07-01	False	1.107
2	6085896	2019-10-01 09:58:33	organic	iPhone	France	2019-10-01	2019-10-01	False	0.000
3	22593348	2019-08-22 21:35:48	AdNonSense	PC	Germany	2019-08-22	2019-08-01	False	0.988
4	31989216	2019-10-02 00:07:44	YRabbit	iPhone	United States	2019-10-02	2019-10-01	False	0.231

Определим минимальную и максимальную даты привлечения пользователей.

In [26]:
```
profiles.agg({'dt': 'min'}) # минимальная дата привлечения пользователей.
```

Out[26]:
```
dt    2019-05-01
dtype: object
```

In [27]:
```
profiles.agg({'dt': 'max'}) # максимальная дата привлечения пользователей.
```

Out[27]:
```
dt    2019-10-27
dtype: object
```

Временной интервал соответствует описанию.

Установим момент и горизонт анализа данных.

In [28]:
```
# момент анализа
observation_date = datetime(2019, 11, 1).date()
horizon_days = 14  # горизонт анализа
```

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

In [29]:
```
report = profiles.groupby('region').agg({'user_id':'nunique', 'payer': ['sum','mean']})
report.columns = ['Пользователи', 'Платящие пользователи', '% Платящие']
report.sort_values(by= 'Пользователи', ascending = False).style.format({'Платящие пользователи'
```

Out[29]:

region	Пользователи	Платящие пользователи	% Платящие
United States	100002	6902	6.90%
UK	17575	700	3.98%
France	17450	663	3.80%
Germany	14981	616	4.11%

Наибольшее число пользователей пришло из США. У них же самый большой процент платящих пользователей, самый маленький во Франции. В Европе процент платящих пользователей не сильно различается.

Построим таблицу, отражающую количество пользователей и долю платящих для каждого канала привлечения.

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]:
```
report = profiles.groupby('channel').agg({'user_id':'nunique', 'payer': ['sum','mean']})
report.columns = ['Пользователи', 'Платящие пользователи', '% Платящие']
report.sort_values(by= 'Пользователи', ascending = False).style.format({'Платящие пользователи'
```

Out[30]:

channel	Пользователи	Платящие пользователи	% Платящие
organic	56439	1160	2.06%
FaceBoom	29144	3557	12.20%
TipTop	19561	1878	9.60%
OppleCreativeMedia	8605	233	2.71%
LeapBob	8553	262	3.06%

channel	Пользователи	Платящие пользователи	% Платящие
WahooNetBanner	8553	453	5.30%
RocketSuperAds	4448	352	7.91%
MediaTornado	4364	156	3.57%
YRabbit	4312	165	3.83%
AdNonSense	3880	440	11.34%
lambdaMediaAds	2149	225	10.47%

Подавляющее число пользователей пришли самостоятельно. Из каналов явный лидер FaceBoom и TipTop, меньше всего пользователей пришло из lambdaMediaAds. Самый низкий процент платящих у OppleCreativeMedia, самый высокий - у AdNonSense и FaceBoom. .

Построим таблицу, отражающую количество пользователей и долю платящих для каждого устройства.

In [31]:
```
report = profiles.groupby('device').agg({'user_id':'nunique', 'payer': ['sum','mean']})
report.columns = ['Пользователи', 'Платящие пользователи', '% Платящие']
report.sort_values(by= 'Пользователи', ascending = False).style.format({'Платящие пользователи'
```

Out[31]:

device	Пользователи	Платящие пользователи	% Платящие
iPhone	54479	3382	6.21%
Android	35032	2050	5.85%
PC	30455	1537	5.05%
Mac	30042	1912	6.36%

По количеству пользователей в лидерах iPhone. Подавляющее большинство пользователей зашли с мобильных устройств. Наибольшую конверсию имеют пользователи, которые пользуются Mac, а затем следуют пользователи iPhone, т.е. мы можем сказать, что в целом пользователи Apple имеют лучшую конверсию в покупателей. Возможно, тут есть плюсы ApplePay. Процент платящих пользователей для каждого устройства не сильно различается. Самый большой -Mac, самый маленький- PC

- Общий вывод:
  - больше всего приходит пользователей из США и они лучше других конвертируется.
  - большая часть пользователей заходит с мобильных устройств
  - наибольшую конверсию имеют пользователи, которые пользуются Mac, а затем следуют пользователи iPhone, т.е. мы можем сказать, что в целом пользователи Apple имеют лучшую конверсию в покупателей. Возможно, тут есть плюсы ApplePay.

Маркетинг

Общая сумма расходов на маркетинг

In [32]:
```
daily_costs_total = costs.sum()
daily_costs_total
```

Out[32]:
```
channel    FaceBoomFaceBoomFaceBoomFaceBoomFaceBoomFaceBo...
costs                                                105497.3
```

```
dtype: object
```

Сумма затрат на рекламу по каналам в каждый день

In [33]:

```
ltv_raw, ltv, ltv_history, roi, roi_history = get_ltv(
    profiles, orders, datetime(2019, 11, 1).date(), 14
)

costs.head()  # таблица ROI
```

Out[33]:

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

Стоимость привлечения одного пользователя

In [34]:

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

    # сортируем сессии по ID пользователя и дате привлечения
    # группируем по ID и находим параметры первых посещений
    profiles = (
        visits.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['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).round(3)

    return profiles  # возвращаем профили с CAC
```

In [35]:

```
profiles = get_profiles(visits, orders, costs)

profiles.head()
```

Out[35]:

	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	1.088
1	4919697	2019-07-09 12:46:07	FaceBoom	iPhone	United States	2019-07-09	2019-07-01	False	1.107
2	6085896	2019-10-01 09:58:33	organic	iPhone	France	2019-10-01	2019-10-01	False	0.000
3	22593348	2019-08-22 21:35:48	AdNonSense	PC	Germany	2019-08-22	2019-08-01	False	0.988
4	31989216	2019-10-02 00:07:44	YRabbit	iPhone	United States	2019-10-02	2019-10-01	False	0.231

В сводной таблице видно, сколько денег тратилось на каждый канал в каждый интервал времени для привлечения каждого посетителя. Столбец acquisition_cost показывает стоимость привлечения каждого посетителя.

Средний CAC по всему проекту

In [36]:

```
profiles['acquisition_cost'].mean().round(3)
```

Out[36]: `0.703`

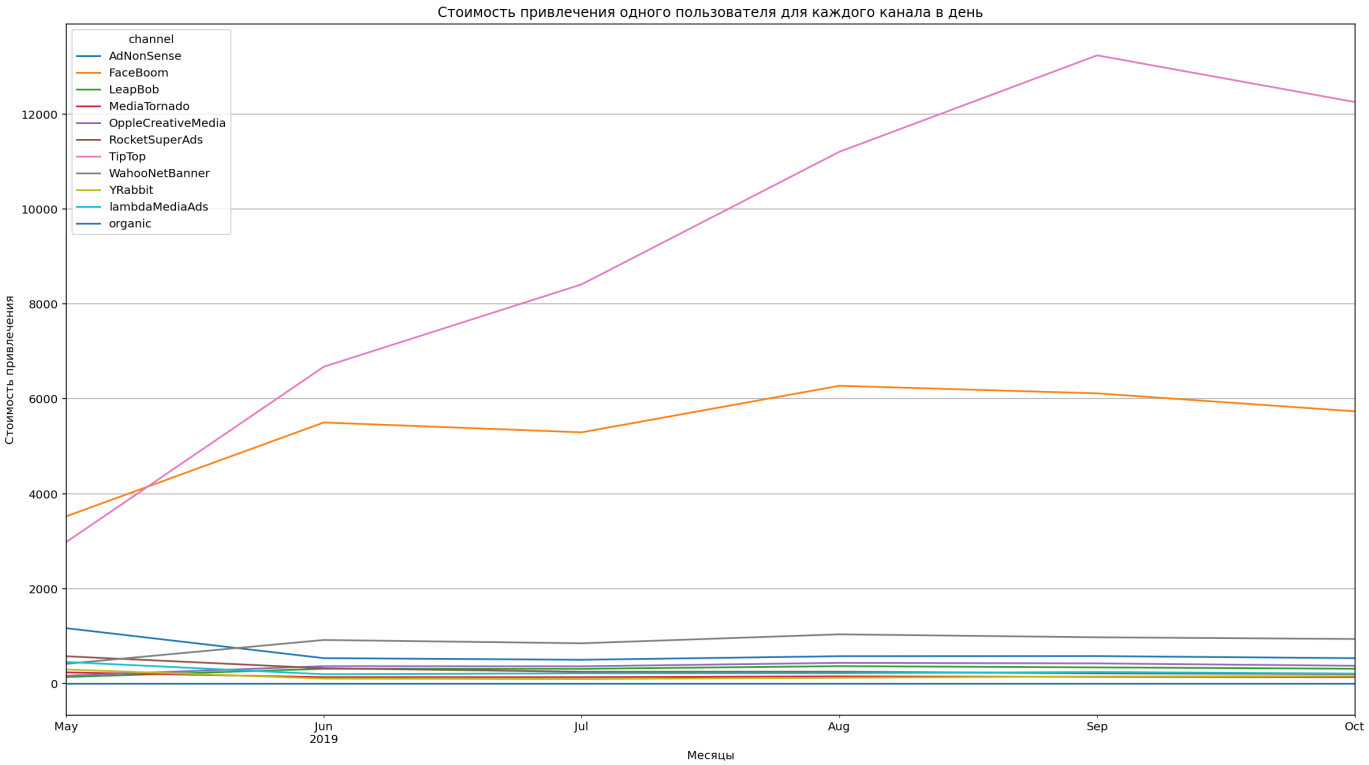
Средняя стоимость привлечения пользователей по всему проекту.

Строим график

Стоимость привлечения одного пользователя для каждого канала в день

In [37]:

```
acquisition_cost_sum = profiles.pivot_table(index='month', columns='channel', values='acquisiti
                                    ).plot(grid=True, figsize=(20, 11))
plt.title('Стоимость привлечения одного пользователя для каждого канала в день')
plt.xlabel('Месяцы')
plt.ylabel('Стоимость привлечения')
plt.show()
```

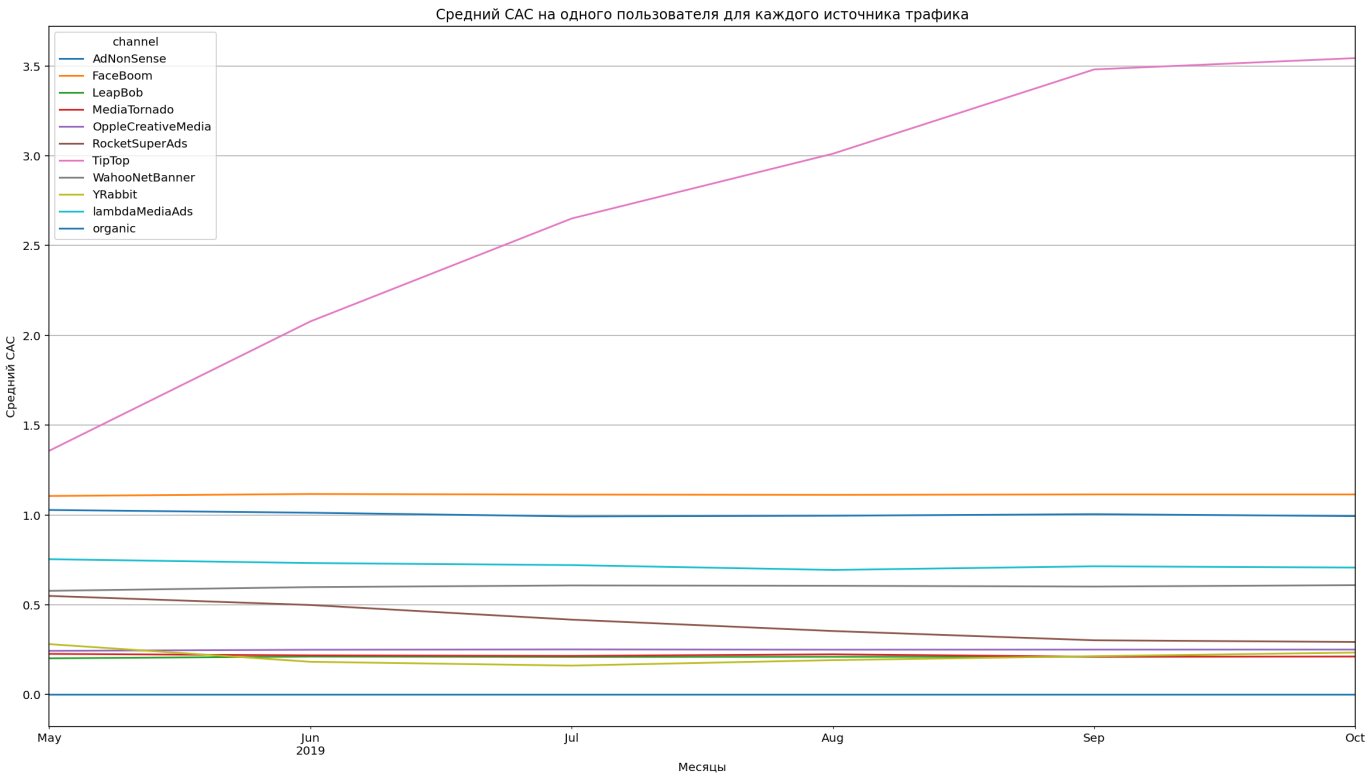


Стоимость привлечения пользователей по каналам неоднозначна. У TipTop и FaceBoom она в несколько раз больше, чем у других пользователей. Видимо, размещение рекламы на этих каналах очень дорого. Учитывая, что большинство пользователей пришли с этих каналов,возможно здесь кроется проблема в окупаемости рекламы. Видно, что затраты на рекламу у FaceBoom превышают 6000 , а у TipTop -12000,в то время, как у других каналов - не более 1000.

Средний CAC на одного пользователя для каждого источника трафика.

In [38]:

```
cquisition_cost_mean = profiles.pivot_table(index='month', columns='channel', values='acquisiti
                                    ).plot(grid=True, figsize=(20, 11))
plt.title('Средний CAC на одного пользователя для каждого источника трафика')
plt.xlabel('Месяцы')
plt.ylabel('Средний CAC')
plt.show()
```



Средняя стоимость привлечения пользователей по каналам различается. У TipTop она выше, чем у других пользователей и непрерывно растет. Размещение рекламы на этом канале очень дорого. Стоимость привлечения «органических» пользователей во всех когортах равна нулю, потому что они пришли самостоятельно, а не благодаря рекламе.

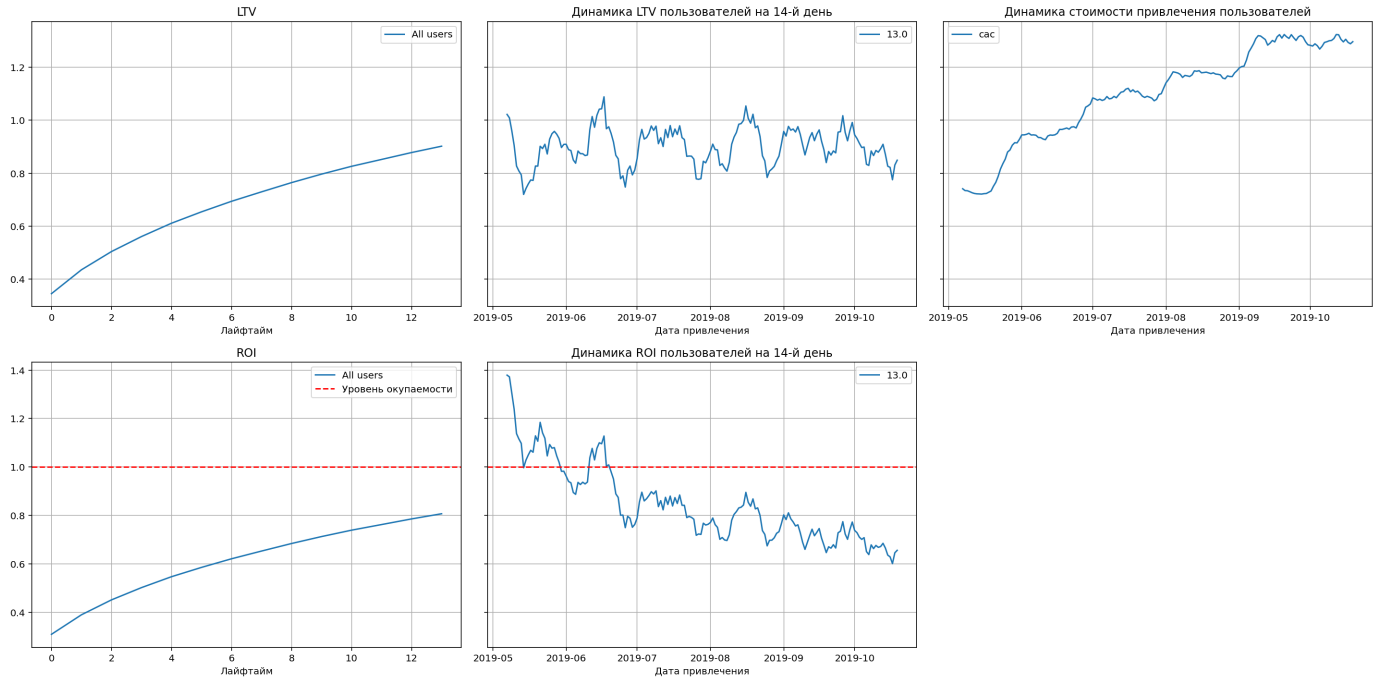
Бизнес-показатели

Для начала оценим общую ситуацию — посмотрим на окупаемость рекламы.

Исключим пользователей с органическим трафиком

In [39]:
```
profiles = profiles.query('channel != "organic"')
```

In [40]:
```
# считаем LTV и ROI
ltv_raw, ltv_grouped, ltv_history, roi_grouped, roi_history = get_ltv(
    profiles, orders, observation_date, horizon_days
)
# строим графики
plot_ltv_roi(ltv_grouped, ltv_history, roi_grouped, roi_history, horizon_days)
```



По графикам можно сделать такие выводы: Реклама не окупается с июня- июля. На LTV влияет временной фактор, но этот показатель достаточно стабилен.Значит, дело не в ухудшении качества пользователей. Стоимость привлечения пользователей растет, причем в начале скачкообразно. Видимо, была рекламная компания. ROI в конце двух недель приблизился к 80%. Чтобы разобраться в причинах, пройдём по всем доступным характеристикам пользователей — стране, источнику и устройству первого посещения.

Смотрим окупаемость с разбивкой по странам

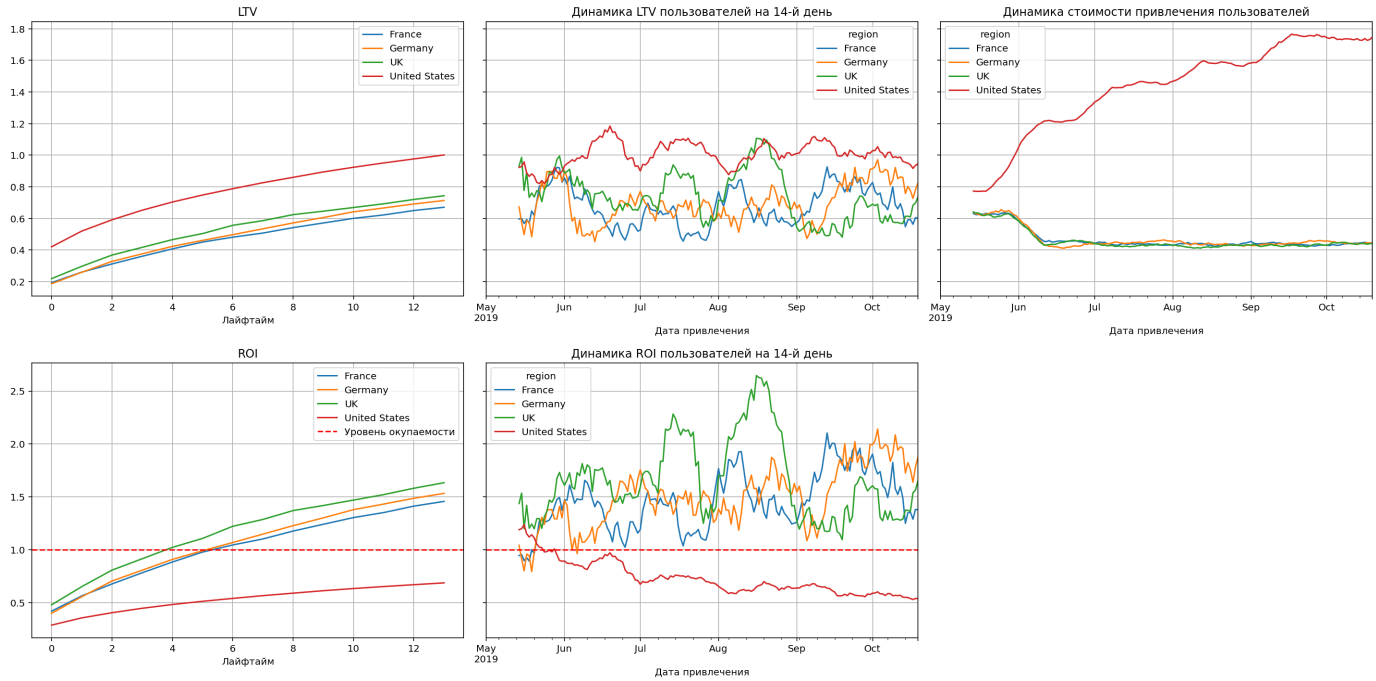
In [41]:

```
# смотрим окупаемость с разбивкой по странам

dimensions = ['region']

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

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



С разбивкой по странам всё неоднозначно. Из графиков видно: LTV в США заметно выше, чем в

Европе.Он подвержен сезонности, но стабилен. Стоимость привлечения после снижения в мае-июне стабильна для всех стран,кроме США.В США она постоянно растет. Реклама не окупается только в США.Лучше всего окупается Великобритания,явный аутсайдер - США. Учитывая, что боьшинство пользователей из США, возможно дело в стране — США этот показатель тянет назад

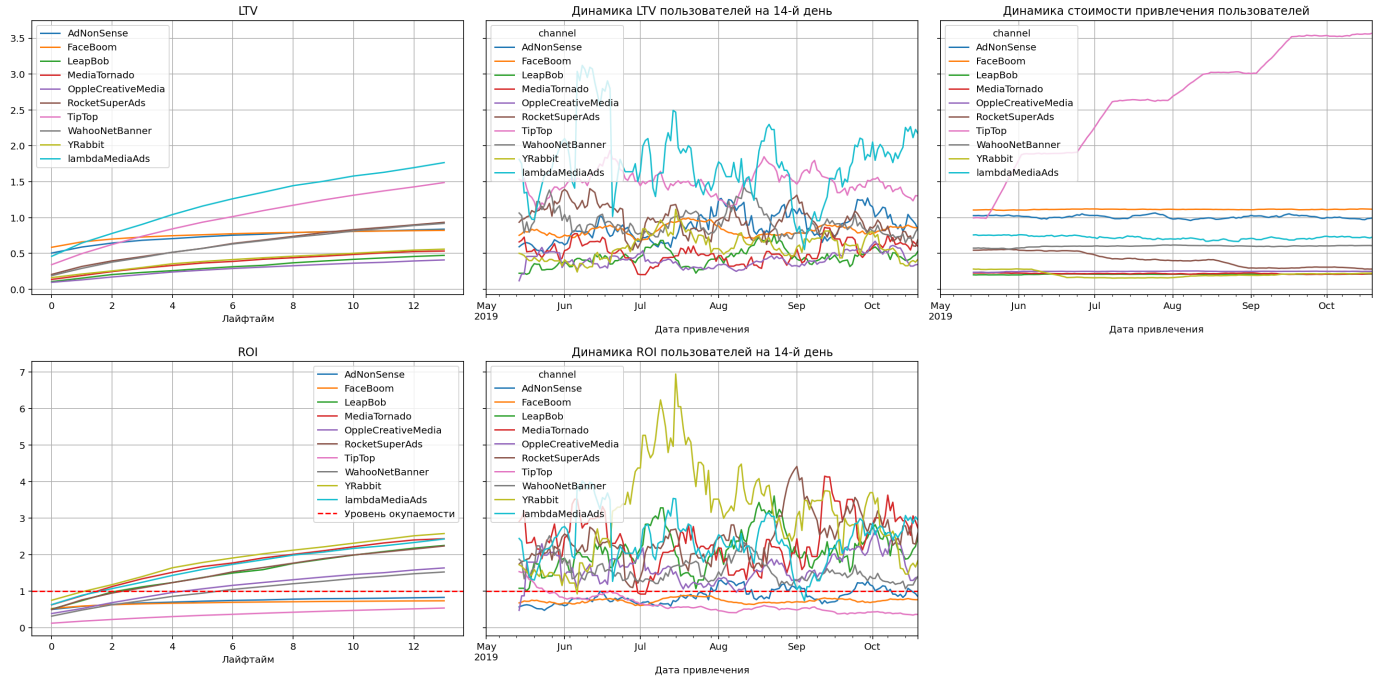
Построим графики, отражающие количество пользователей и долю платящих для каждого канала привлечения.

In [42]:
```
#смотрим окупаемость с разбивкой по источникам привлечения

dimensions = ['channel']

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

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



ltv самая лучшая у lambdaMediaAds и TipTop,самая худшая OppleCreativeMedia. Стоимость привлечения пользователей у всех каналов, кроме Tip Top стабильная. У Tip Top она резко растет и намного выше. Стоимость привлечения пользователей у RocketSuperAds даже снижается. Окупаются не все каналы.FaceBoom,TipTop и AdNonSense не окупаются. Лучше всех YRabbit,MediaTornado и lambdaMediaAds. По динамике roi выделяется YRabbit. Возможно влияют аномалии.

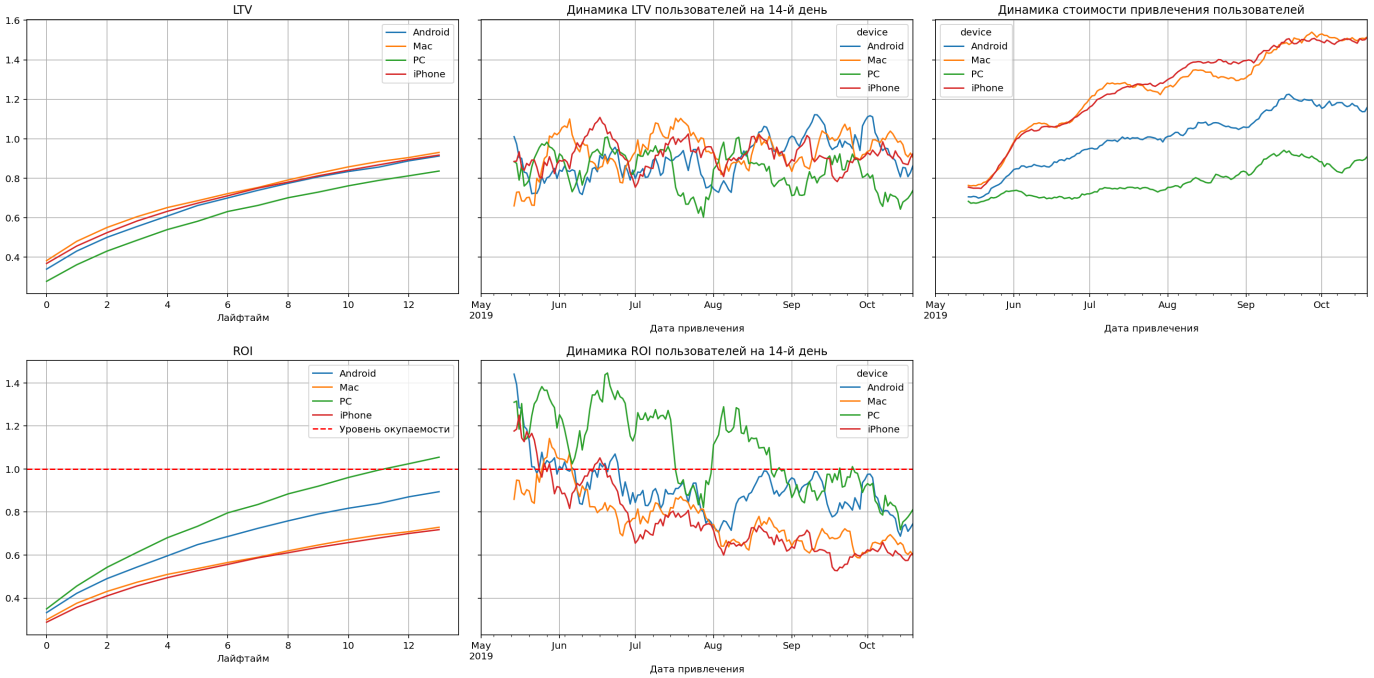
Построим графики, отражающие количество пользователей и долю платящих для каждого устройства.

In [43]:
```
#смотрим окупаемость с разбивкой по устройствам

dimensions = ['device']

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

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



ltv по устройствам почти не различается. Исключение PC, здесь он ниже. Динамика ltv стабильна. Стоимость привлечения владельцев растет, у iPhone и Mac она заметно выше. Пользователи PC стабильно окупаются, а вот владельцы мобильных устройств — нет. Динамика окупаемости показывает снижение у всех устройств.

Узнаем, в чём причина: в низкой конверсии или низком удержании.

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

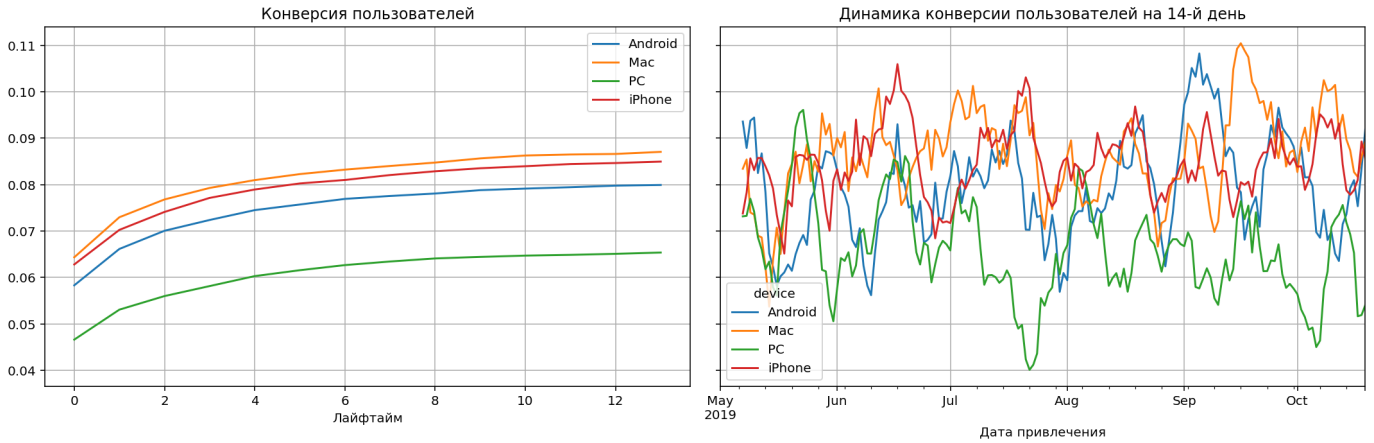
конверсия с разбивкой по устройствам

In [44]:

```
# смотрим конверсию с разбивкой по устройствам

conversion_raw, conversion_grouped, conversion_history = get_conversion(
    profiles, orders, observation_date, horizon_days, dimensions=dimensions
)

plot_conversion(conversion_grouped, conversion_history, horizon_days)
```



Судя по графикам, пользователи iPhone и Mac конвертируются очень хорошо, причём постоянно.Заметно хуже у пользователей PC.

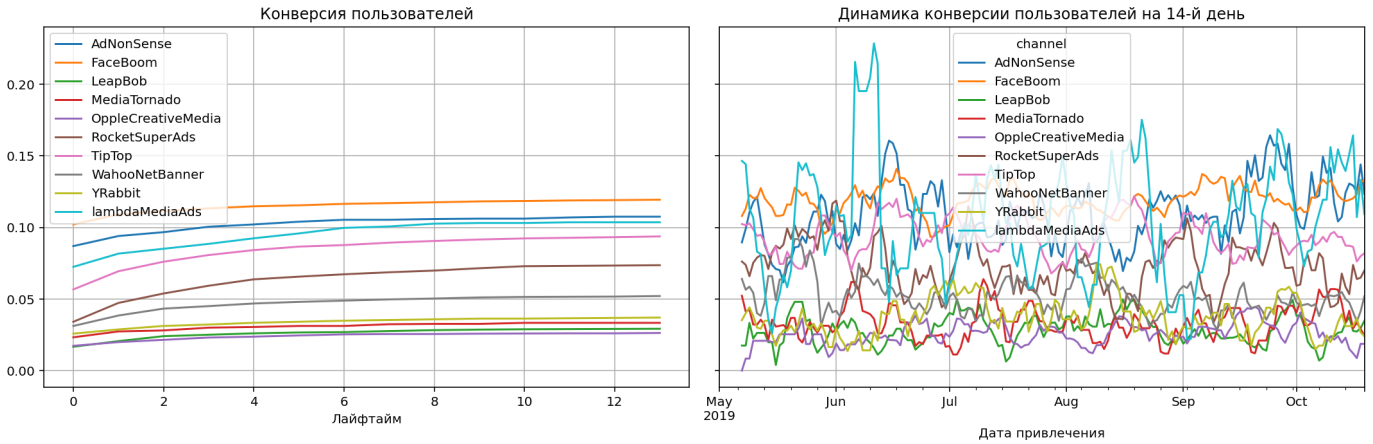
конверсия с разбивкой по по каналам

In [45]:

```
# смотрим конверсию с разбивкой по каналам
dimensions = ['channel']
conversion_raw, conversion_grouped, conversion_history = get_conversion(
```

```
    profiles, orders, observation_date, horizon_days, dimensions=dimensions
)

plot_conversion(conversion_grouped, conversion_history, horizon_days)
```



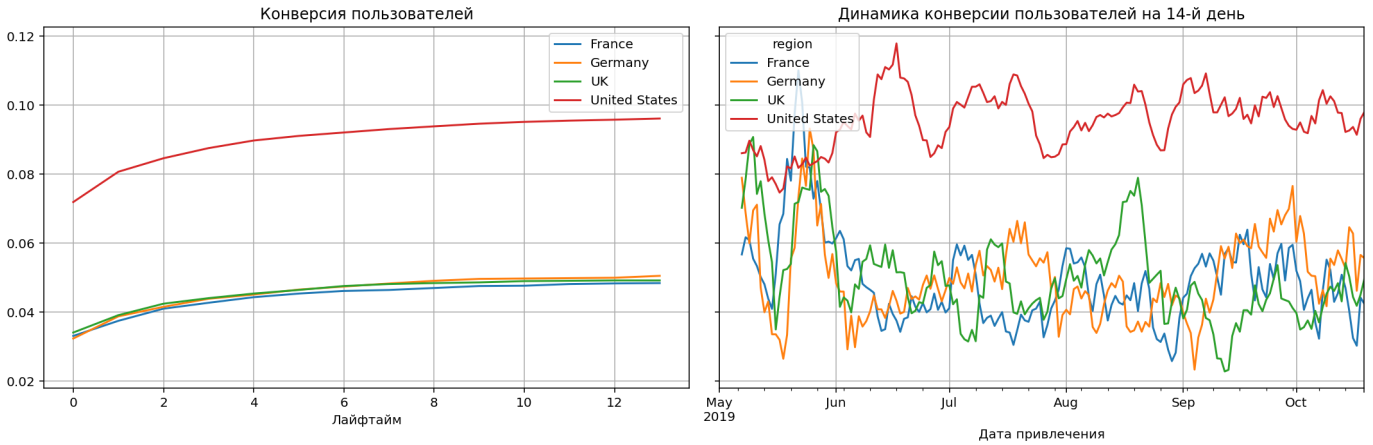
Судя по графикам,лучше всего конвертируются пользователи через FaceBoom, AdNonSense, lambdaMediaAds и Tip Top. Хуже всего - OppleCreativeMedia. Динамика конверсии стабильна у всех каналов,кроме lambdaMediaAds-здесь наблюдаются большие пики,аномальные. Видимо,есть выбросы в данных

конверсия с разбивкой по по странам:

In [46]:

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

plot_conversion(conversion_grouped, conversion_history, horizon_days)
```



Пользователи, пришедшие из США конвертируются очень хорошо, в 2 раза лучше, чем европейские. Динамика конверсии пользователей на 14 день стабильна и подвержена сезонности.

Посчитаем и визуализируем удержание.

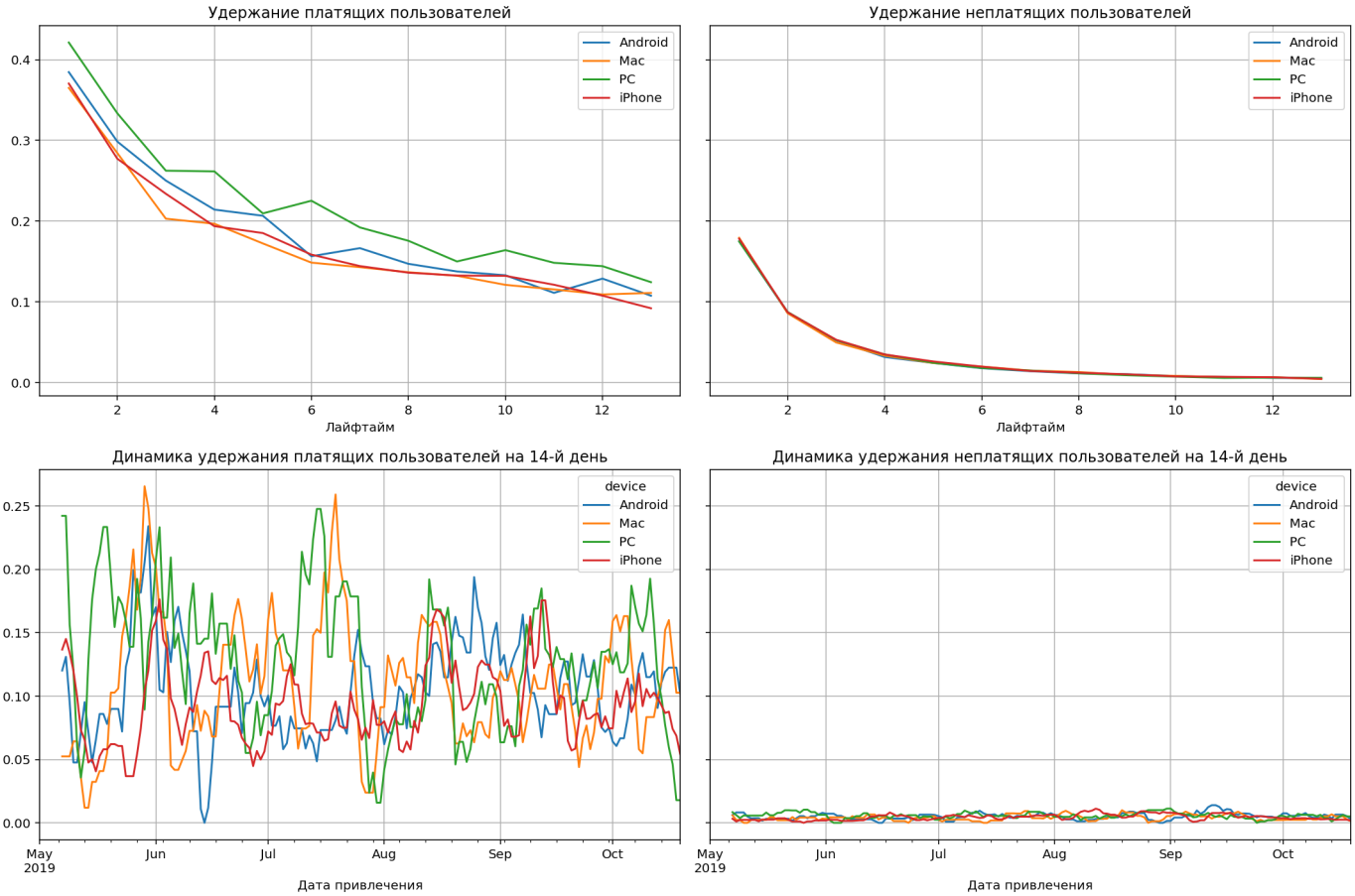
Вызовем функции get_retention() и plot_retention(), чтобы рассчитать и отразить на графиках этот показатель.

удержание с разбивкой по устройствам

In [47]:

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

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



Удержание платящих пользователей по устройствам на 14 день не сильно различается. Удержание неплатящих пользователей показывает что в какой бы день ни привлекались неплательщики, у них стабильно удержание около нуля. Значит,с устройствами все в порядке, явно не они причина плохой маркетинговой кампании.

удержание с разбивкой по каналам

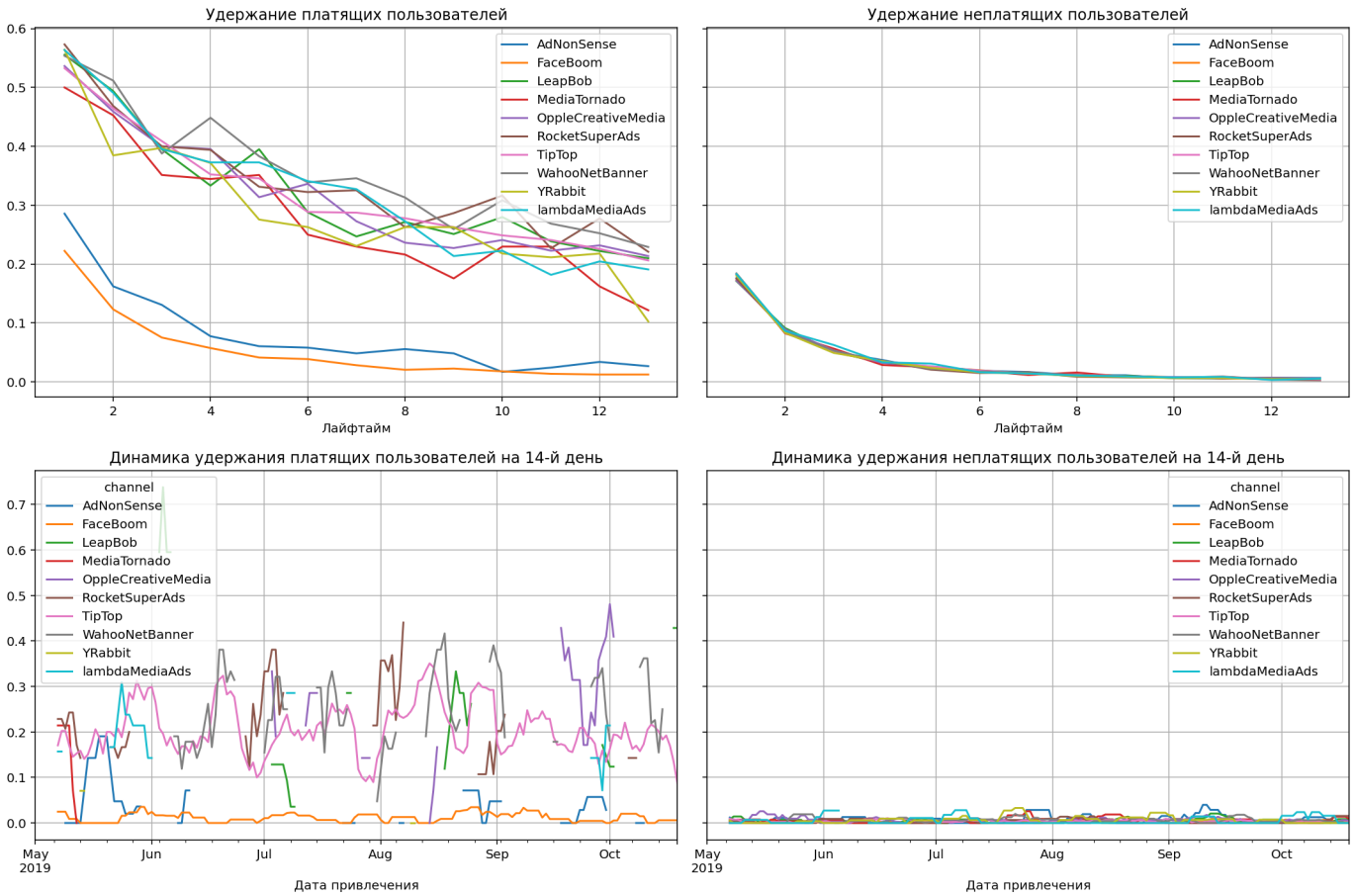
In [48]:

```
# смотрим удержание с разбивкой по по каналам:

dimensions = ['channel']

retention_raw, retention_grouped, retention_history = get_retention(
    profiles, visits, observation_date, horizon_days, dimensions=dimensions
)

plot_retention(retention_grouped, retention_history, horizon_days)
```



Удержание платящих пользователей по каналам на 14 день не сильно различается, за исключением FaceBoom и AdNonSense. Удержание неплатящих пользователей стабильно около нуля.

удержание с разбивкой по странам.

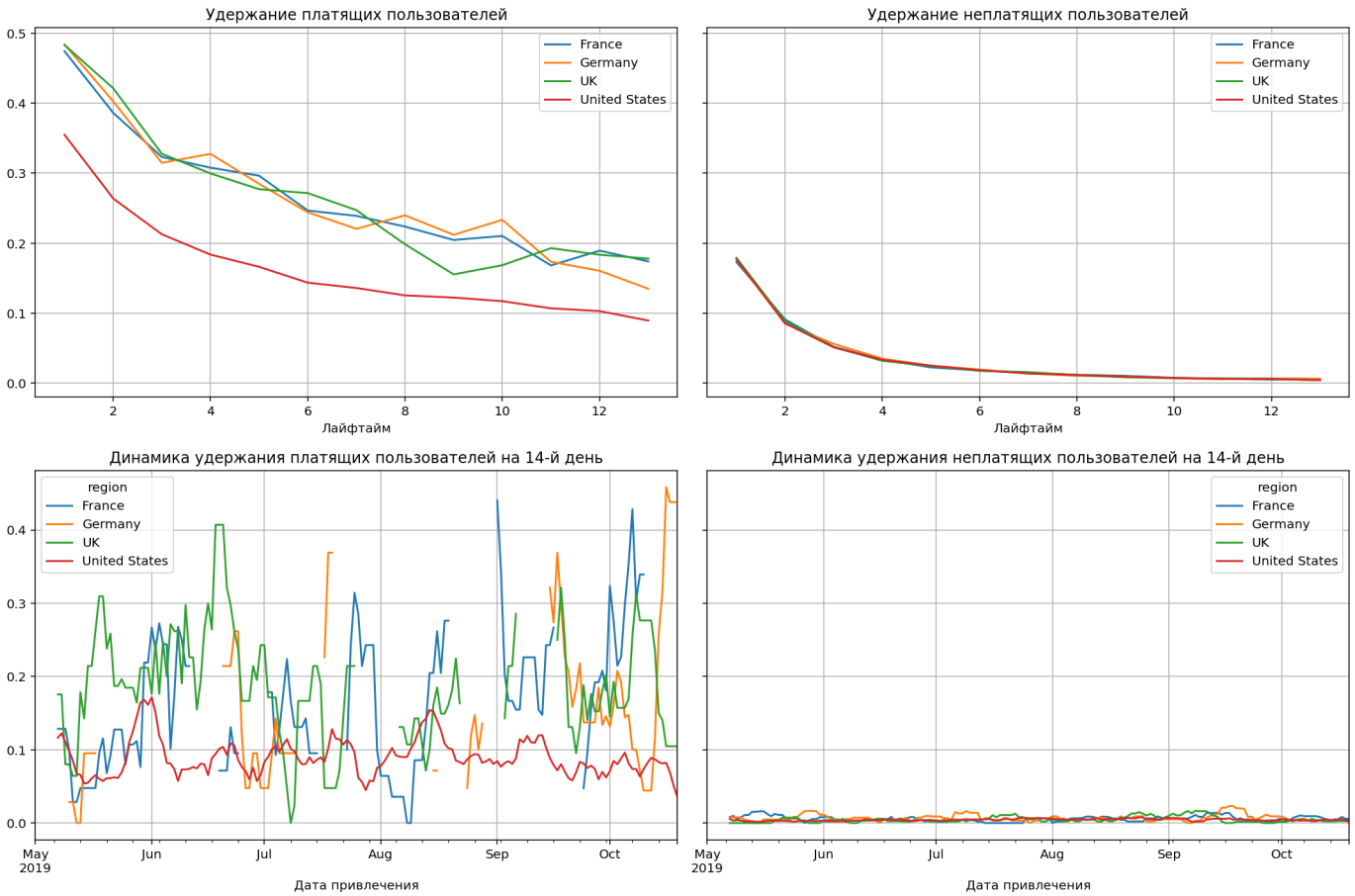
In [49]:

```
# смотрим удержание с разбивкой по по странам:

dimensions = ['region']

retention_raw, retention_grouped, retention_history = get_retention(
    profiles, visits, observation_date, horizon_days, dimensions=dimensions
)

plot_retention(retention_grouped, retention_history, horizon_days)
```



Удержание платящих пользователей по странам на 14 день не сильно различается, за исключением США. Здесь удержание значительно хуже.

Выводы

В целом, реклама перестала окупаться с июня - июля. На окупаемость рекламы негативно влияют: страны - США. Здесь расходы на рекламу намного выше. А ведь большинство пользователей оттуда. каналы - FaceBoom.Расходы на рекламу здесь вторые по затратам.А удержание у FaceBoom самое низкое.

Рекомендации: Больше внимания уделить другим странам.Процент платящих клиентов в Европе намного ниже, чем в США. Есть куда расти. Уделить внимание FaceBoom - очень низкое удержание. Надо искать причины. Подумать, как больше привлечь пользователей со стационарных устройств - стоимость привлечения у PC низкая, удержание хорошее. Но пользователей пока немного. Также можно предложить для американского рынка вкладывать средства в RocketSuperAds, который показывает неплохую конверсию, очень хороший ROI и высокое удержание платящих пользователей.