Описание 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 отрицательно влияют на привлечение пользователей

Нам предстоит:

- изучить данные, которые имеются в нашем распоряжении.
- Выполнить предобработку
- Провести исследовательский анализ данных
- Посчитать затраты на рекламу
- Посчтиать сколько денег приносят пользователи
- Оценить окупаемость и дать рекомендации отделу маркетинга

Знакомство с данными

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from datetime import datetime, timedelta
```

In [2]:
```
pd.set_option('display.max_columns', None)
pd.options.display.float_format = '{:,.2f}'.format
```

In [3]:
```
try:
    visits = pd.read_csv('visits_info_short.csv')
    orders = pd.read_csv('orders_info_short.csv')
    costs = pd.read_csv('costs_info_short.csv')
except:
    visits = pd.read_csv('https://code.s3.yandex.net/datasets/visits_info_short.csv')
    orders = pd.read_csv('https://code.s3.yandex.net/datasets/orders_info_short.csv')
    costs = pd.read_csv('https://code.s3.yandex.net/datasets/costs_info_short.csv')
```

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.

С данными ознакомились. Следующим шагом выполним их предобработку

Предобработка данных

Предобработка VISITS

In [4]:
```
visits.head()
```

Out[4]:

	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

In [5]:
```
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 [6]:
```
#изменим наименование колонок
visits.columns = [col.lower().replace(' ','_') for col in visits.columns]
```

In [7]:
```
# Приведем даты к правильному типу
visits['session_start'] = pd.to_datetime(visits['session_start'])
visits['session_end'] = pd.to_datetime(visits['session_end'])
```

In [8]:
```
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
```

In [9]:
```
visits.duplicated().sum()
```

Out[9]:
```
0
```

Дубликатов нет.

Предобработка ORDERS

In [10]:
```
orders.head()
```

Out[10]:

	User Id	Event Dt	Revenue
0	188246423999	2019-05-01 23:09:52	4.99
1	174361394180	2019-05-01 12:24:04	4.99
2	529610067795	2019-05-01 11:34:04	4.99
3	319939546352	2019-05-01 15:34:40	4.99
4	366000285810	2019-05-01 13:59:51	4.99

In [11]:
```
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 [12]:
```
#изменим наименование колонок
orders.columns = [col.lower().replace(' ','_') for col in orders.columns]
```

In [13]:
```
# Приведем даты к правильному типу
orders['event_dt'] = pd.to_datetime(orders['event_dt'])
```

In [14]:
```
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  datetime64[ns]
 2   revenue   40212 non-null  float64       
dtypes: datetime64[ns](1), float64(1), int64(1)
memory usage: 942.6 KB
```

In [15]:
```
orders.duplicated().sum()
```

Out[15]:
```
0
```

Дубликатов нет.

In [16]:
```
costs.head()
```

Out[16]:

	dt	Channel	costs
0	2019-05-01	FaceBoom	113.30
1	2019-05-02	FaceBoom	78.10
2	2019-05-03	FaceBoom	85.80
3	2019-05-04	FaceBoom	136.40
4	2019-05-05	FaceBoom	122.10

In [17]:
```
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
```

In [18]:
```
#изменим наименование колонок
costs.columns = [col.lower() for col in costs.columns]
```

In [19]:
```
# Приведем даты к правильному типу
costs['dt'] = pd.to_datetime(costs['dt']).dt.date
```

In [20]:
```
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
```

In [21]:
```
costs.duplicated().sum()
```

Out[21]: 0

Дубликатов нет.

Выполнили предобработку данных. Заменили названия столбцов, тип некоторых данных.

Исследовательский анализ данных

Создание пользовательские профили. Определение минимальной и максимальной даты привлечения пользователей.

In [22]:
```
def get_profiles(visits):

    # сортируем сессии по ID пользователя и дате посещения
    # группируем по ID и находим первые значения session_start, channel, region, de
    # столбец с временем первого посещения назовём first_ts
    # от англ. first timestamp — первая временная отметка
    profiles = (
        visits.sort_values(by=['user_id', 'session_start'])
        .groupby('user_id')
        .agg({'session_start': 'first', 'channel': 'first', 'region': 'first', 'dev
        .rename(columns={'session_start': 'first_ts'})
        .reset_index()  # возвращаем user_id из индекса
    )

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

    # определим "платящий" ли пользователь
    profiles['payer'] =  profiles['user_id'].isin(orders['user_id'].unique())

    return profiles
```

In [23]:
```
profiles = get_profiles(visits)
```

In [24]:
```
profiles.head()
```

Out[24]:

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

In [25]:
```
min_date = profiles['dt'].min()
max_date = profiles['dt'].max()

print(f'Минимальная дата привлечения пользователей {min_date}')
print(f'Максимальная дата привлечения пользователей {max_date}')
```

```
Минимальная дата привлечения пользователей 2019-05-01
Максимальная дата привлечения пользователей 2019-10-27
```

Пользователи по странам/устройствам/каналам

Выясним:

- из каких стран пользователи приходят в приложение и на какую страну приходится больше всего платящих пользователей. Постройте таблицу, отражающую количество пользователей и долю платящих из каждой страны.
- какими устройствами пользуются клиенты и какие устройства предпочитают платящие пользователи. Постройте таблицу, отражающую количество пользователей и долю платящих для каждого устройства.

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

In [26]:
```
profiles['payer'].mean()
```

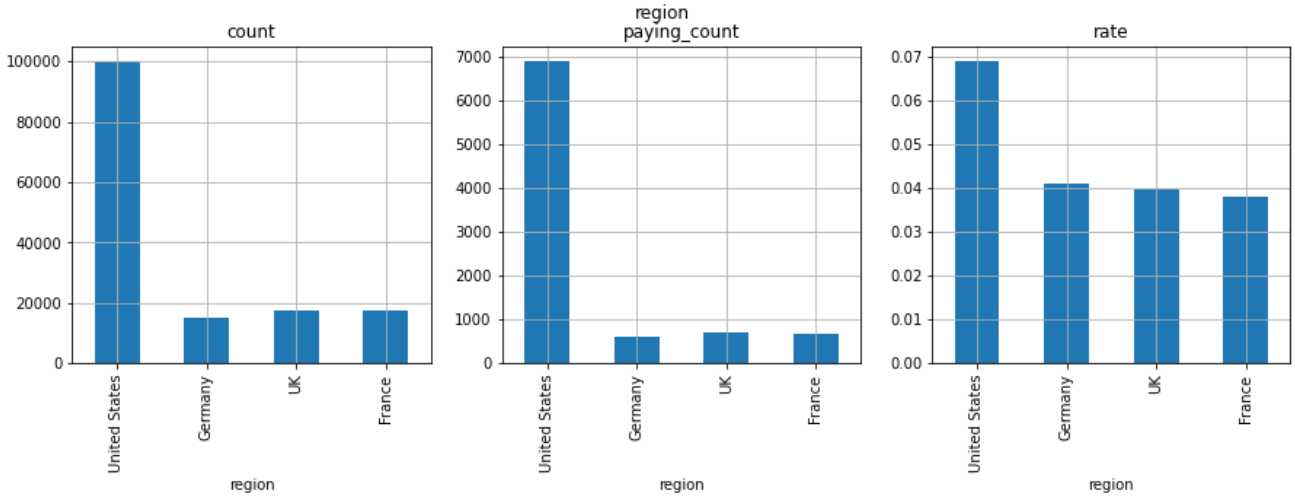
Out[26]:
```
0.05920350914617887
```

Платящих пользователей всего 5.9 процентов от всех пользователей сервиса

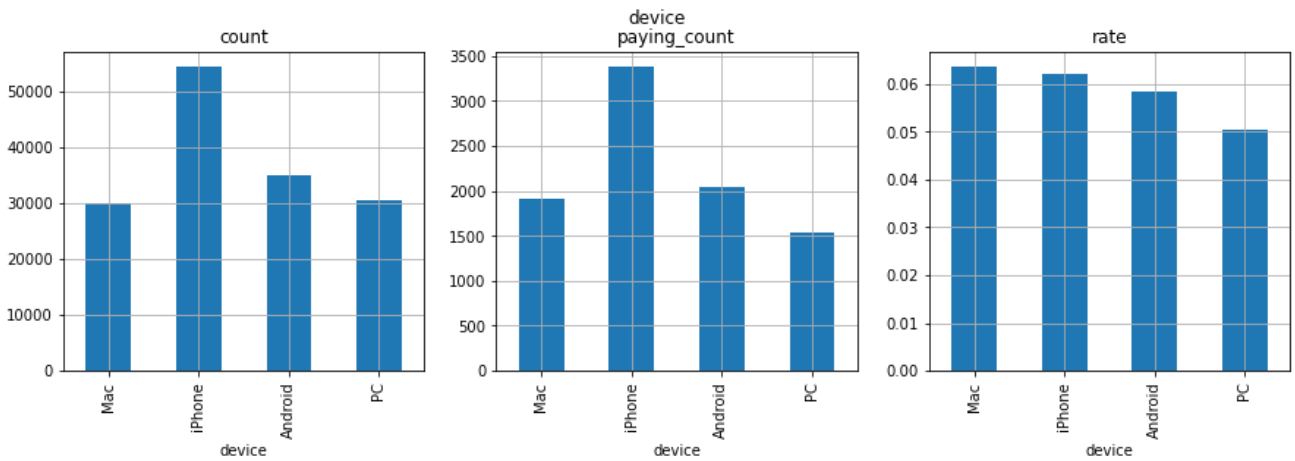
Поскольку нам предстоит произвести примерно одинаковые манипуляции, предлагаю сделать это при помощи цикла.

In [27]:
```
criteria_list = ['region', 'device', 'channel']
for i in criteria_list:
    #Цикл пробегается по критериям и для каждого строит сводник с долей, количество
    stat = (profiles
                    .groupby(i)['payer']
                    .agg(['count', 'sum', 'mean'])
                    .sort_values('mean', ascending=False)
                   )

    display(stat)
    # Визуализируем данные для наглядности
    fig, axs = plt.subplots(1,3)
    fig.suptitle(i)
    fig.set_figheight(4)
    fig.set_figwidth(15)
    stat['count'].plot(kind='bar', grid=True, title='count', ax=plt.subplot(1, 3, 1
    stat['sum'].plot(kind='bar', grid=True, title='paying_count', ax=plt.subplot(1,
    stat['mean'].plot(kind='bar', grid=True, title='rate', ax=plt.subplot(1, 3, 3))
    plt.show()
```

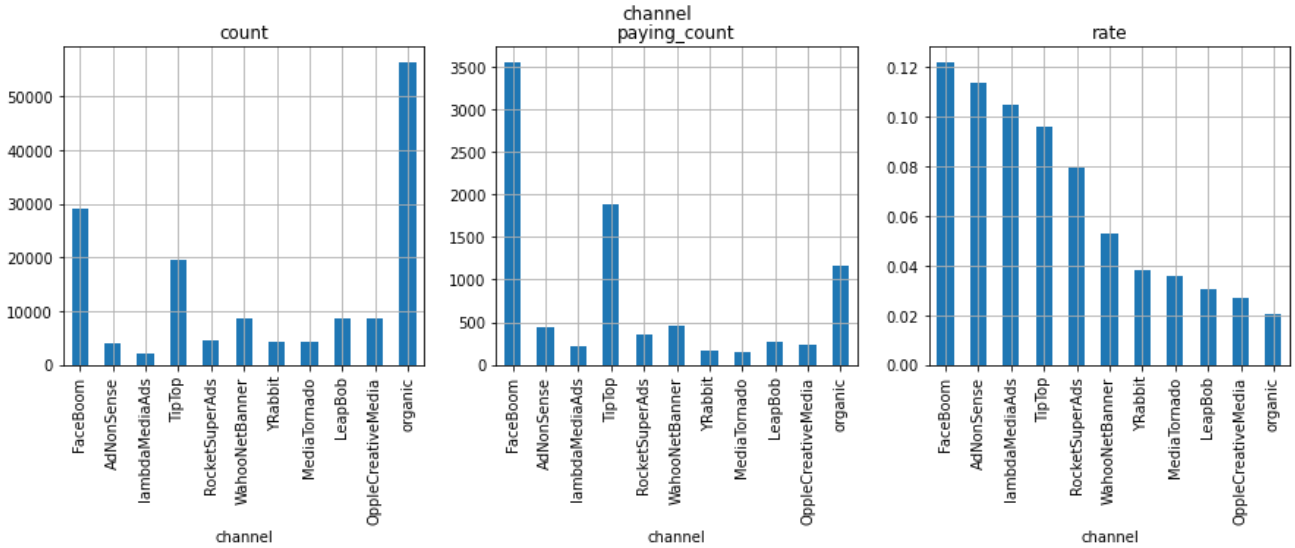
region	count	sum	mean
United States	100002	6902	0.07
Germany	14981	616	0.04
UK	17575	700	0.04
France	17450	663	0.04



device	count	sum	mean
Mac	30042	1912	0.06
iPhone	54479	3382	0.06
Android	35032	2050	0.06
PC	30455	1537	0.05



channel	count	sum	mean
FaceBoom	29144	3557	0.12
AdNonSense	3880	440	0.11
lambdaMediaAds	2149	225	0.10
TipTop	19561	1878	0.10
RocketSuperAds	4448	352	0.08
WahooNetBanner	8553	453	0.05
YRabbit	4312	165	0.04
MediaTornado	4364	156	0.04
LeapBob	8553	262	0.03
OppleCreativeMedia	8605	233	0.03
organic	56439	1160	0.02



Платящих пользователей больше всего в США, как и в целом всех пользователей сервиса. Доля также самая высокая у США, у остальных стран примерно одинаковая доля. Но в целом доля платящих пользователей достаточно мала.

Больше всего пользователей, которые пользуются айфонами. Пользователей остальных устройств примерно одинаковое количество. Доля платящих пользователей для всех платформ примерно равна, разве что немного "отстают" пользователи ПК.

Больше всего пользователей приходят в сервис органическим путем, однако доля платящих среди них всего 2%. Больше всего платящих пользователей приходит из источника FaceBoom. Учитывая, что по количеству всех пользователей FaceBoom занимает второе место, можно считать что это самый работающий канал привлечения клиентов. Также неплохие показатели у канала TipTop

---

Расходы на маркетинг

Посчитаем общую сумму расходов на маркетинг. Выясним, как траты распределены по источникам. Визуализируем изменения метрик во времени.

In [28]:
```
total_cost = costs['costs'].sum()
print(f'Общие затраты на рекламу: {total_cost}')
```

```
Общие затраты на рекламу: 105497.30000000002
```

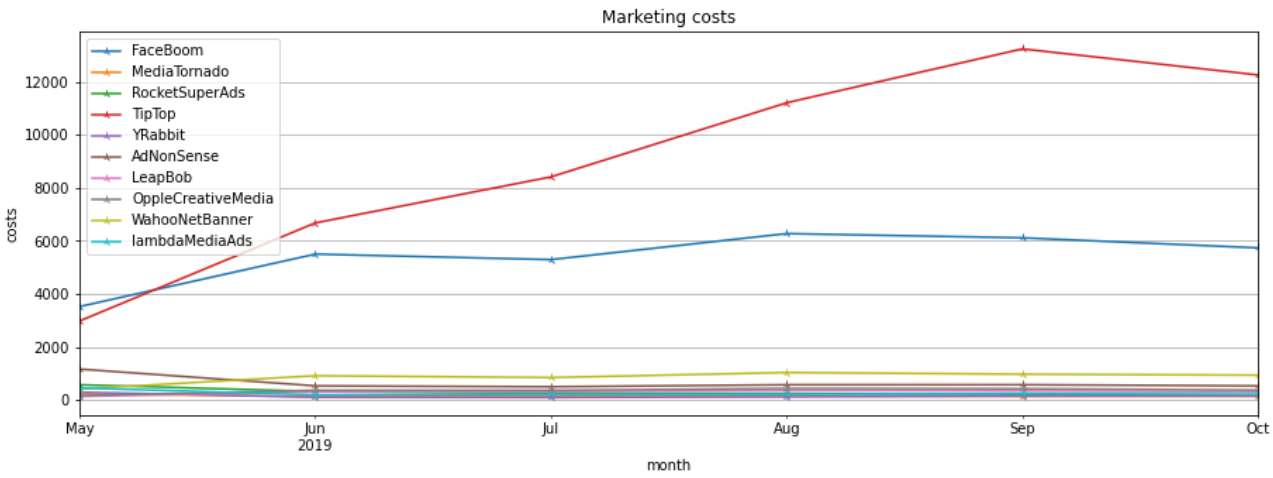
In [29]:
```
# Сводник по затратам на рекламу для каждого источника
costs.groupby('channel')['costs'].agg('sum').reset_index().sort_values('costs', a
```

Out[29]:

	channel	costs
6	TipTop	54,751.30
1	FaceBoom	32,445.60
7	WahooNetBanner	5,151.00
0	AdNonSense	3,911.25
4	OppleCreativeMedia	2,151.25
5	RocketSuperAds	1,833.00
2	LeapBob	1,797.60
9	lambdaMediaAds	1,557.60
3	MediaTornado	954.48
8	YRabbit	944.22

In [30]:
```
# Добавим колонку с мясецем
costs['month'] = costs['dt'].astype('datetime64[M]')
```

In [31]:
```
for i in list(costs['channel'].unique()):
    (costs[costs['channel'] == i]
     .groupby('month')['costs']
     .sum()
     .plot(grid=True, figsize=(15,5), marker='2')
    )
    plt.title('Marketing costs')
    plt.legend(costs['channel'].unique())
    plt.xlabel('month')
    plt.ylabel('costs')
```

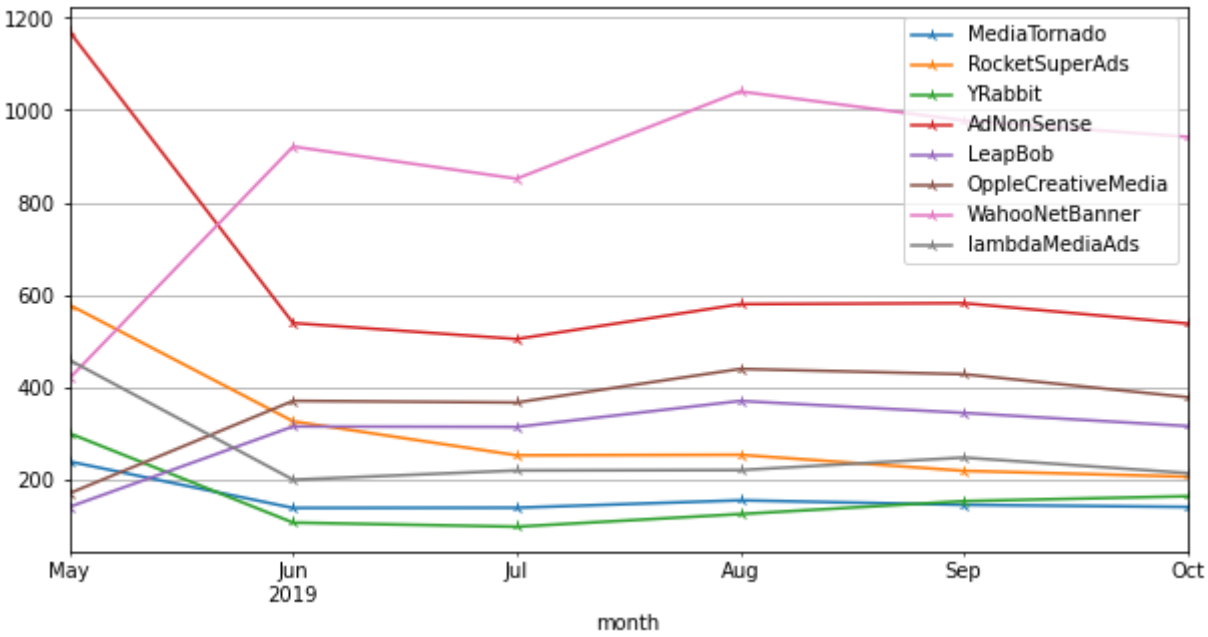


FaceBoom и TipTop сильно выделяются на фоне остальных источников. Построю отдельный график для остальных платформ. Это нужно для того, чтобы повнимательнее рассмотреть динамику по остальным источникам

In [32]:
```
# Платформы для 2го графика
```

```
li2 = ['MediaTornado',
 'RocketSuperAds',
 'YRabbit',
 'AdNonSense',
 'LeapBob',
 'OppleCreativeMedia',
 'WahooNetBanner',
 'lambdaMediaAds']
```

In [33]:
```
for i in li2:
    (costs[costs['channel'] == i]
     .groupby('month')['costs']
     .sum()
     .plot(grid=True, figsize=(10,5), marker='2')
    )
    plt.legend(li2)
```



Ну и табличку для наглядности :)

In [34]:
```
costs.groupby(['channel', 'month'])['costs'].sum().reset_index()
```

Out[34]:

	channel	month	costs
0	AdNonSense	2019-05-01	1,169.70
1	AdNonSense	2019-06-01	538.65
2	AdNonSense	2019-07-01	504.00
3	AdNonSense	2019-08-01	579.60
4	AdNonSense	2019-09-01	581.70
5	AdNonSense	2019-10-01	537.60
6	FaceBoom	2019-05-01	3,524.40
7	FaceBoom	2019-06-01	5,501.10
8	FaceBoom	2019-07-01	5,294.30
9	FaceBoom	2019-08-01	6,274.40
10	FaceBoom	2019-09-01	6,114.90
11	FaceBoom	2019-10-01	5,736.50
12	LeapBob	2019-05-01	140.28
13	LeapBob	2019-06-01	314.58
14	LeapBob	2019-07-01	313.53
15	LeapBob	2019-08-01	369.81
16	LeapBob	2019-09-01	343.98
17	LeapBob	2019-10-01	315.42
18	MediaTornado	2019-05-01	238.56
19	MediaTornado	2019-06-01	138.00
20	MediaTornado	2019-07-01	138.48
21	MediaTornado	2019-08-01	154.56
22	MediaTornado	2019-09-01	144.72
23	MediaTornado	2019-10-01	140.16
24	OppleCreativeMedia	2019-05-01	169.75
25	OppleCreativeMedia	2019-06-01	370.00
26	OppleCreativeMedia	2019-07-01	366.50
27	OppleCreativeMedia	2019-08-01	439.25
28	OppleCreativeMedia	2019-09-01	427.75
29	OppleCreativeMedia	2019-10-01	378.00
30	RocketSuperAds	2019-05-01	577.98
31	RocketSuperAds	2019-06-01	325.72
32	RocketSuperAds	2019-07-01	252.07
33	RocketSuperAds	2019-08-01	253.11
34	RocketSuperAds	2019-09-01	218.40
35	RocketSuperAds	2019-10-01	205.72
36	TipTop	2019-05-01	2,981.00
37	TipTop	2019-06-01	6,675.60
38	TipTop	2019-07-01	8,410.20
39	TipTop	2019-08-01	11,202.00

	channel	month	costs
40	TipTop	2019-09-01	13,232.50
41	TipTop	2019-10-01	12,250.00
42	WahooNetBanner	2019-05-01	418.80
43	WahooNetBanner	2019-06-01	921.00
44	WahooNetBanner	2019-07-01	851.40
45	WahooNetBanner	2019-08-01	1,040.40
46	WahooNetBanner	2019-09-01	977.40
47	WahooNetBanner	2019-10-01	942.00
48	YRabbit	2019-05-01	299.70
49	YRabbit	2019-06-01	106.20
50	YRabbit	2019-07-01	97.38
51	YRabbit	2019-08-01	124.74
52	YRabbit	2019-09-01	152.79
53	YRabbit	2019-10-01	163.41
54	lambdaMediaAds	2019-05-01	458.40
55	lambdaMediaAds	2019-06-01	199.20
56	lambdaMediaAds	2019-07-01	219.20
57	lambdaMediaAds	2019-08-01	220.00
58	lambdaMediaAds	2019-09-01	247.20
59	lambdaMediaAds	2019-10-01	213.60

Теперь можно сделать выводы:

Сумма затрат на все источники составляет 105497.3

На FaceBoom и TipTop компания тратит намного больше, чем на другие платформы. Что касается FaceBoom, то тут еще можно согласиться, ведь доля платящих клиентов для этого источника самая высокая, а вот TipTop занимает лишь 4е место по доле платящих клиентов. Возможно, компании стоит пересмотреть маркетинговую компанию. Будем разбираться с этим

Далее посмотрим на "низшую лигу": затраты на источник "AdNonSense" в июне уменьшили более чем в 2 раза, хотя доля платящих клиентов по этому источнику на втором месте. А вот для источника WahooNetBanner бюджет наоборот увеличили в 2 раза. Очень странное решение, учитывая то, что данный источник даже не входит в топ-5 по доле платящих клиентов.

Затраты на остальные источники в целом выглядят ровно

Стоимость привлечения пользователей

Узнаем, сколько в среднем стоило привлечение одного пользователя из каждого

источника. Рассчитаем средний CAC на одного пользователя для всего проекта и для каждого источника трафика.

In [35]:
```
profiles.head()
```

Out[35]:

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

In [36]:
```
# Посчитаем число пользователей привлеченных в определенный день определнной пла
new_users = (
    profiles.groupby(['dt', 'channel'])
    .agg({'user_id': 'nunique'})
    .rename(columns={'user_id': 'unique_users'})
    .reset_index()
)
new_users.head()
```

Out[36]:

	dt	channel	unique_users
0	2019-05-01	AdNonSense	39
1	2019-05-01	FaceBoom	104
2	2019-05-01	LeapBob	12
3	2019-05-01	MediaTornado	26
4	2019-05-01	OppleCreativeMedia	18

In [37]:
```
# Объединим траты на рекламу и число пользоваталей
costs = costs.merge(new_users, on=['dt', 'channel'], how='left')

# Считаем CAC
costs['acquisition_cost'] = costs['costs'] / costs['unique_users']

costs.head()
```

Out[37]:

	dt	channel	costs	month	unique_users	acquisition_cost
0	2019-05-01	FaceBoom	113.30	2019-05-01	104	1.09
1	2019-05-02	FaceBoom	78.10	2019-05-01	72	1.08
2	2019-05-03	FaceBoom	85.80	2019-05-01	76	1.13
3	2019-05-04	FaceBoom	136.40	2019-05-01	123	1.11
4	2019-05-05	FaceBoom	122.10	2019-05-01	113	1.08

In [38]:
```
# Добавим стоимость привлечения в профили
profiles = profiles.merge(
    costs[['dt', 'channel', 'acquisition_cost']],
    on=['dt', 'channel'],
    how='left',
```

```
)
# Заменим пропуски на ноль для CAC органических пользователей
profiles['acquisition_cost'] = profiles['acquisition_cost'].fillna(0)
```

In [39]:
```
profiles.head()
```

Out[39]:

	user_id	first_ts	channel	region	device	dt	month	payer	acquisition_cost
0	599326	2019-05-07 20:58:57	FaceBoom	United States	Mac	2019-05-07	2019-05-01	True	1.09
1	4919697	2019-07-09 12:46:07	FaceBoom	United States	iPhone	2019-07-09	2019-07-01	False	1.11
2	6085896	2019-10-01 09:58:33	organic	France	iPhone	2019-10-01	2019-10-01	False	0.00
3	22593348	2019-08-22 21:35:48	AdNonSense	Germany	PC	2019-08-22	2019-08-01	False	0.99
4	31989216	2019-10-02 00:07:44	YRabbit	United States	iPhone	2019-10-02	2019-10-01	False	0.23

Теперь посмотрим сколько в среднем стоило привлечение одного пользователя из каждого источника

In [40]:
```
display(
    profiles
    .groupby('channel')['acquisition_cost']
    .mean()
    .reset_index()
    .sort_values('acquisition_cost', ascending=False)
)

avg_cost = round(profiles[profiles['channel'] != 'organic']['acquisition_cost'].
print(f'Средняя стоимость привлечения одного клиента по всему проекту: {avg_cost
```

	channel	acquisition_cost
6	TipTop	2.80
1	FaceBoom	1.11
0	AdNonSense	1.01
9	lambdaMediaAds	0.72
7	WahooNetBanner	0.60
5	RocketSuperAds	0.41
4	OppleCreativeMedia	0.25
8	YRabbit	0.22
3	MediaTornado	0.22
2	LeapBob	0.21
10	organic	0.00

```
Средняя стоимость привлечения одного клиента по всему проекту: 1.13
```

Средняя стоимость привлечения одного клиента по всему проекту: 0.7. Дороже всех получились клиенты из TipTop, причем они опережают второе по данному

показателю более чем в 2,5 раза. Пока источник TipTop выглядит очень сомнительно: Самые дорогие бюджеты на рекламу, самые дорогие пользователи и при этом далеко не самая большая доля платящих клиентов.

Окупаемость рекламы

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

Общая окупаемость рекламы

Проанализируем общую окупаемость рекламы. Построим графики LTV и ROI, а также графики динамики LTV, CAC и ROI.

Зададим функцию для расчета LTV и ROI

In [41]:
```
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
    )
```

Также зададим функцию для визуализации

In [42]:
```
# Функция для сглаживания
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
```

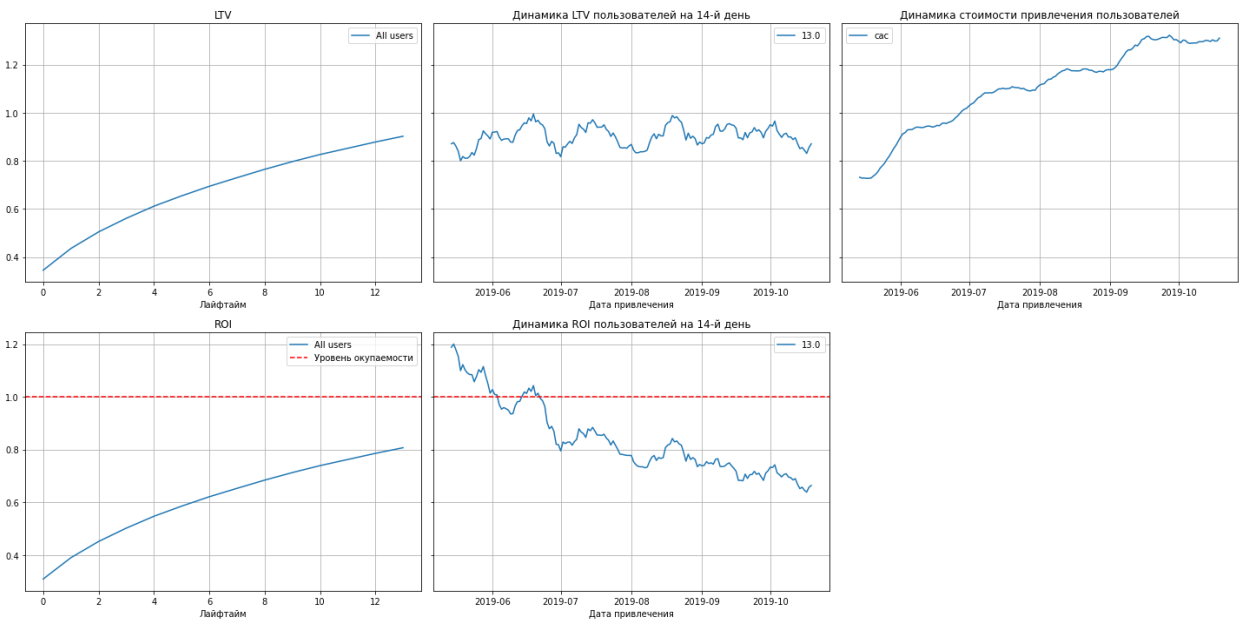
In [43]:
```
# Функуия для визуализации
def plot_ltv_roi(ltv, ltv_history, roi, roi_history, horizon, window=7):

    # 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()
```

In [44]:
```
# Установим момент и горизонт анализа (из условий)
observation_date = datetime(2019, 11, 1).date()
horizon_days = 14
```

Проверим окупаемость рекламы.

In [45]:
```
# Уберем оранических пользователей
profiles_without_organic = profiles[profiles['channel'] != 'organic'].copy()

# LTV и ROI
ltv_raw, ltv_grouped, ltv_history, roi_grouped, roi_history = get_ltv(
    profiles_without_organic, orders, observation_date, horizon_days
)

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



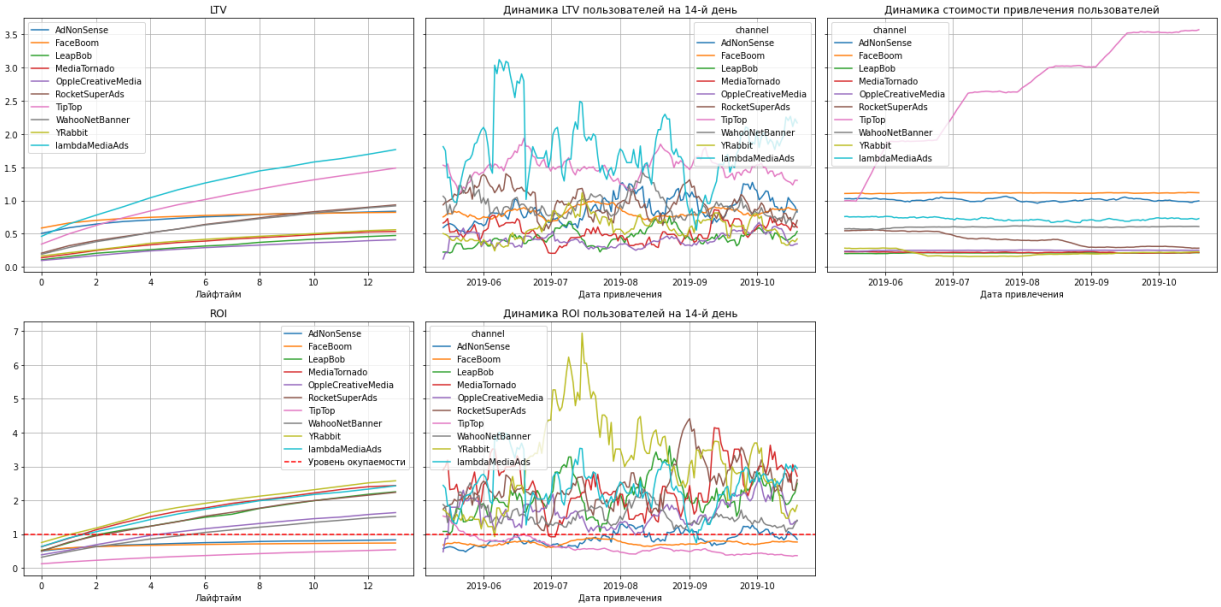
Выводы по общей окупаемости рекламы:

- Кривая LTV плавно растет от нуля, значит LTV рассчитан верно.
- LTV в целом достаточно стабилен.
- Реклама не окупается к 14 дню жизни пользователей
- Стоимсоть привлечения пользователей резко возрастает начиная с июня.
- Вместе с этим можно заметить, что начиная с июля ROI всегда меньше единицы (100%)

Окупаемость по каналам

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

In [46]:
```
# Окупаемость с разбивкой по источникам
dimensions = ['channel']

ltv_raw, ltv_grouped, ltv_history, roi_grouped, roi_history = get_ltv(
    profiles_without_organic, orders, observation_date, horizon_days, dimensio
)

plot_ltv_roi(
    ltv_grouped, ltv_history, roi_grouped, roi_history, horizon_days, window=1
)
```



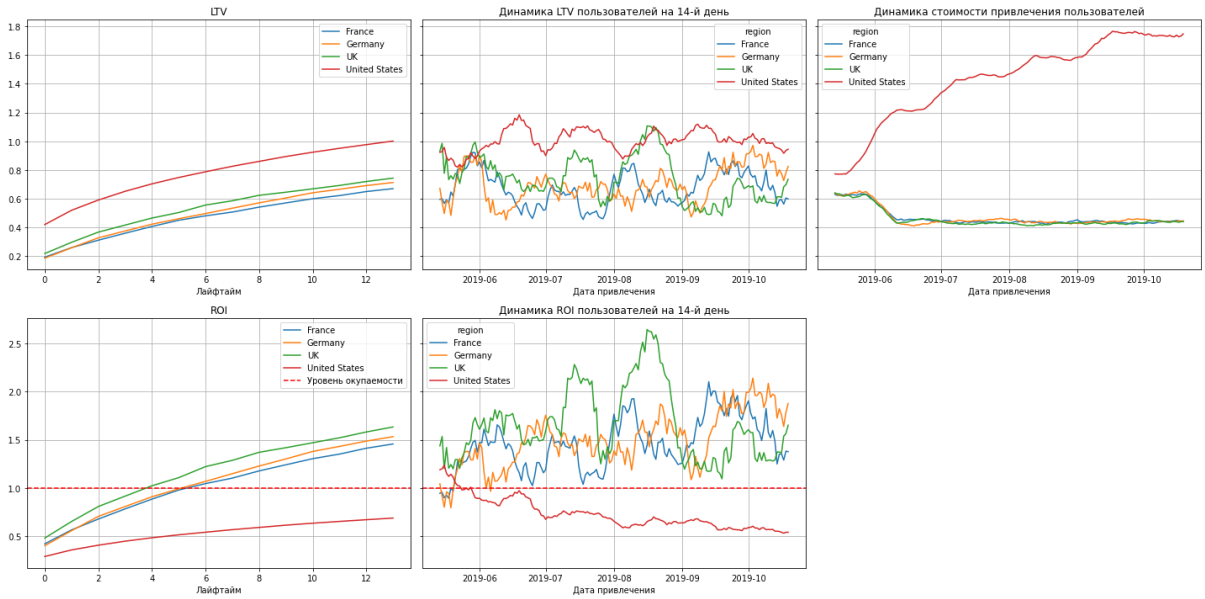
Здесь картина уже более наглядная для анализа

- По динамике LTV можно выделить лидера - lambdaMediaAds, на втором месте TipTop.
- На прошлом шаге мы отмечали увеличение стоимости привлечения пользователей начина с июня. Теперь можно назвать и причину такого увеличения (хотя мы уже подозревали неладное на шаге с расчетом CAC) - начиная с июня на TipTop компания начинала тратить сильно больше.
- Что же касается ROI, тут так же появилось больше ясности: AdNonSense, FaceBoom и TipTop стабильно не окупаются. Напомню, что данные три источника являются самыми дорогими по стоимости привлечения пользоваителей.
- Остальные источники в целом стабильны и показывают результат ROI около 100-125%. Здесь немного выделяется в лучшую сторону YRabbit, который показал результат почти в 170% в середине июля.

Окупаемость по странам

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

In [47]:
```
# Окупаемость с разбивкой по странам
dimensions = ['region']

ltv_raw, ltv_grouped, ltv_history, roi_grouped, roi_history = get_ltv(
    profiles_without_organic, orders, observation_date, horizon_days, dimens
)

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



Очень интересно....
Начиная с июня окупаемость рекламы в Америке резко упала. А мы помним, что в США самая большая доля платящих клиентов. Да и в целом клиентов из США намного больше, чем из других стран.

Пользователи из Англии, Германии и Франции хорошо окупаются.

Напомню правильо расчета ROI (LTV/CAC). Значит, чтобы так резко уменьшилось ROI должно либо резко увеличиться CAC, либо резко уменьшиться LTV. По графику LTV мы видим, что ценность клиентов в целом стабильна, даже показывает рост. Получается все дело в увеличении CAC для клиентов из Америки

Увеличение CAC связано с ростом расходов на рекламу и/или уменьшением когорты

Посмотрим на кол-во пользователей и стоимость привлечения по месяцу для США

In [48]:
```
display(
    profiles
    .query('region == "United States"')
    .groupby('month')
    .agg({'user_id': 'nunique', 'acquisition_cost': 'mean'})
```

```
    .reset_index()
)
```

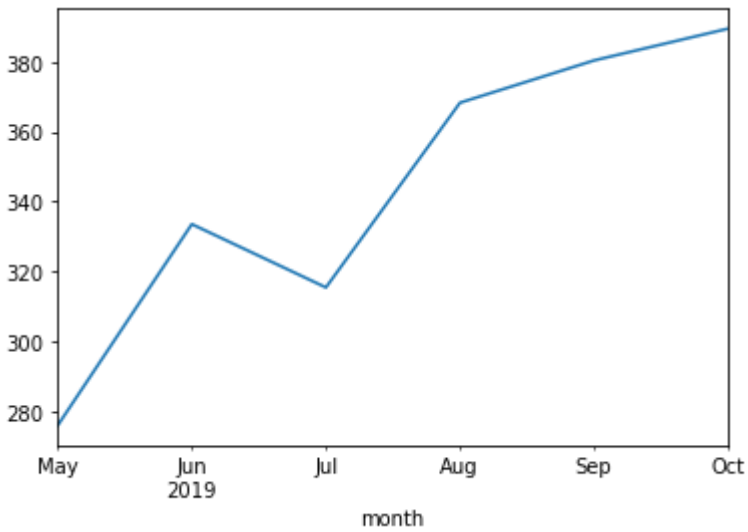
	month	user_id	acquisition_cost
0	2019-05-01	17974	0.42
1	2019-06-01	15379	0.83
2	2019-07-01	15077	0.94
3	2019-08-01	17581	1.02
4	2019-09-01	17663	1.12
5	2019-10-01	16328	1.13

Также посмотрим как менялся размер когорты для США от месяца к месяцу

In [49]:
```
# Избавимся от многоэтажных индексов
usa_cohort_size = roi_history.reset_index()

# Добавим колонку с месяцем
usa_cohort_size['month'] = usa_cohort_size['dt'].astype('datetime64[M]')

# Сводник
usa_cohort_size = (
    usa_cohort_size
    .query('region == "United States"')
    .groupby('month')['cohort_size']
    .mean()
)
display(usa_cohort_size)

# и выведем график для наглядности
usa_cohort_size.plot()
```

```
month
2019-05-01   275.90
2019-06-01   333.50
2019-07-01   315.39
2019-08-01   368.26
2019-09-01   380.37
2019-10-01   389.47
Name: cohort_size, dtype: float64
```

Out[49]:
```
<AxesSubplot:xlabel='month'>
```

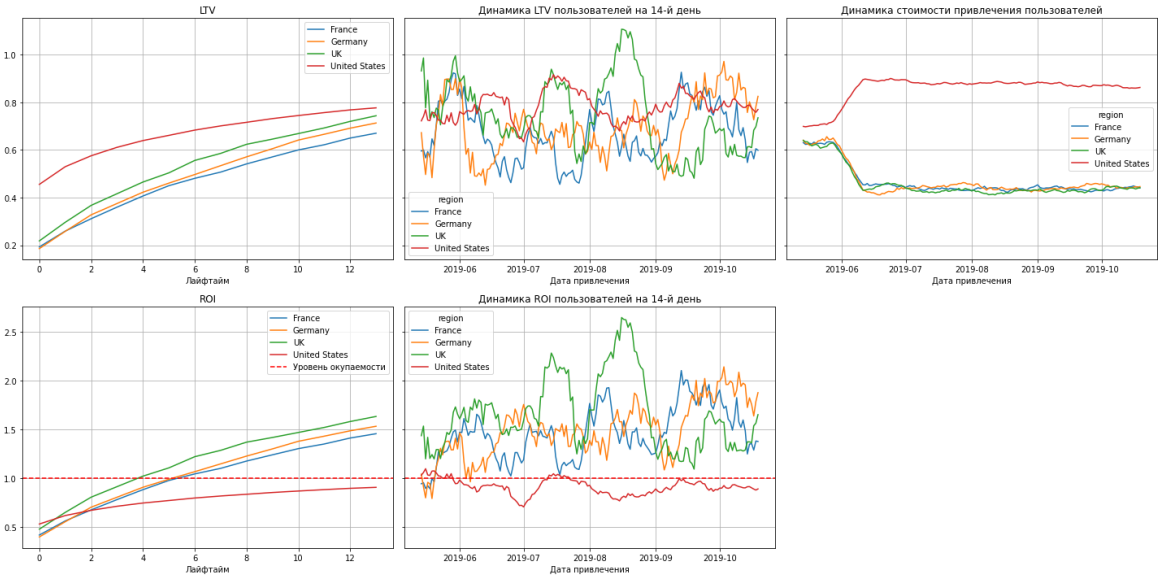


Размер когорты в целом показывал рост. Значит основная причина в увеличении затрат на рекламу

Ну что же, вывода достаточно очевидны. Начиная с лета показатель ROI для Америки резко упал. Это связано с резким увеличением затрат на привлечение пользователей. На мой взгляд, это связано с резким "вкачиванием" средств на источник TipTop. Напомню, что с июня месяца по этому источнику мы заметили аномальный рост затрат на рекламу. Думаю, это и послужило основной причиной для снижения окупаемости.

Предлагаю построить графики без учета пользователей из данной платформы и посмотреть на результаты.

In [50]:
```
# убираем TipTop из профилей
profiles_without_tiptop = profiles_without_organic.query('channel != "TipT

# Получаем результаты
ltv_raw, ltv_grouped, ltv_history, roi_grouped, roi_history = get_ltv(
    profiles_without_tiptop, orders, observation_date, horizon_days, dimer
)

# Визуализируем
dimensions = ['region']
plot_ltv_roi(
    ltv_grouped, ltv_history, roi_grouped, roi_history, horizon_days, winc
)
```



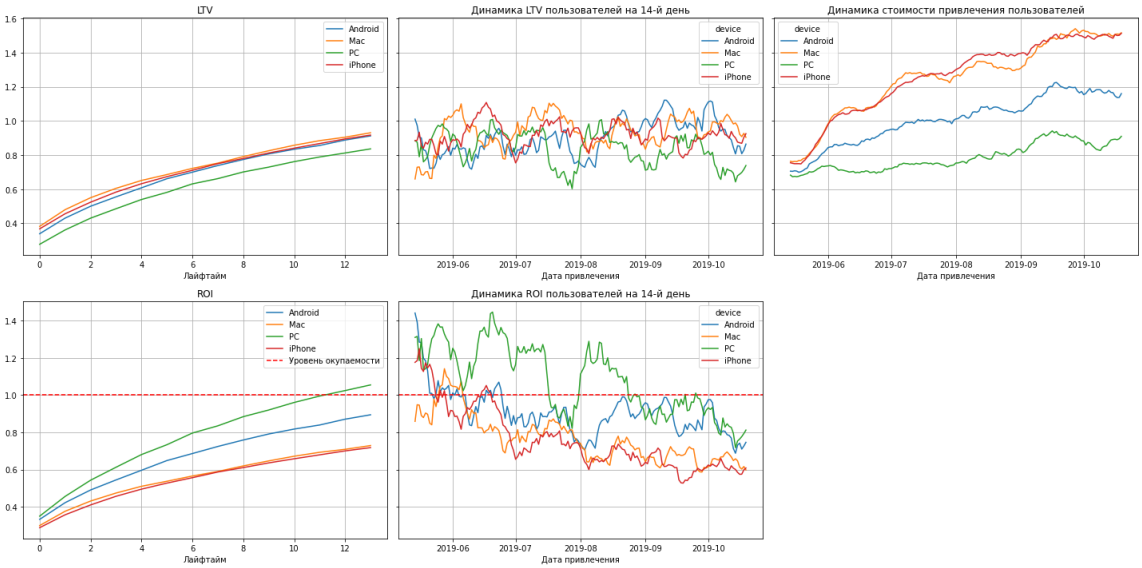
Картина заметно улучшилась. На предыдущем графике ROI постоянно шел вниз, а без учета TipTop затраты на рекламу несколько раз окупались. График выглядит заметно лучше

Окупаемость по устройствам

Также я бы проверил окупаемость с разбивкой по устройствам. Может быть мы узнаем еще что-нибудь полезное

In [51]:
```
# Окупаемость с разбивкой по устройствам
dimensions = ['device']

ltv_raw, ltv_grouped, ltv_history, roi_grouped, roi_history = get_ltv(
    profiles_without_organic, orders, observation_date, horizon_days, di
```

```
)

plot_ltv_roi(
    ltv_grouped, ltv_history, roi_grouped, roi_history, horizon_days, wi
)
```



К концу 2й недели окупаются лишь пользователи PC. От них немного отстают пользователи Android. Также можно заметить, что стоимость привлечения пользователей айфонов и маков значительно выше, чем для пользователей других устройств. Возможно, это связано и со страной этих пользователей. Ведь в Америке значительно больше пользователей яблочной продукции.

Конверсия и удержание

Построим и изучите графики конверсии и удержания с разбивкой по устройствам, странам, рекламным каналам. Ответим на такие вопросы:

- Окупается ли реклама в целом?
- Какие устройства, страны и каналы могут снижать окупаемость рекламы?
- Чем могут быть вызваны проблемы окупаемости?

Зададим функции для подсчета удержания и конверсии

In [52]:
```
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='nu
        )
        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_

    # 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 [53]:

```
# 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='lef
    )

    # 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='nu
        )
        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_

    # 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 [54]:

```
# 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'.forma
            horizon
        )
    )

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

In [55]:

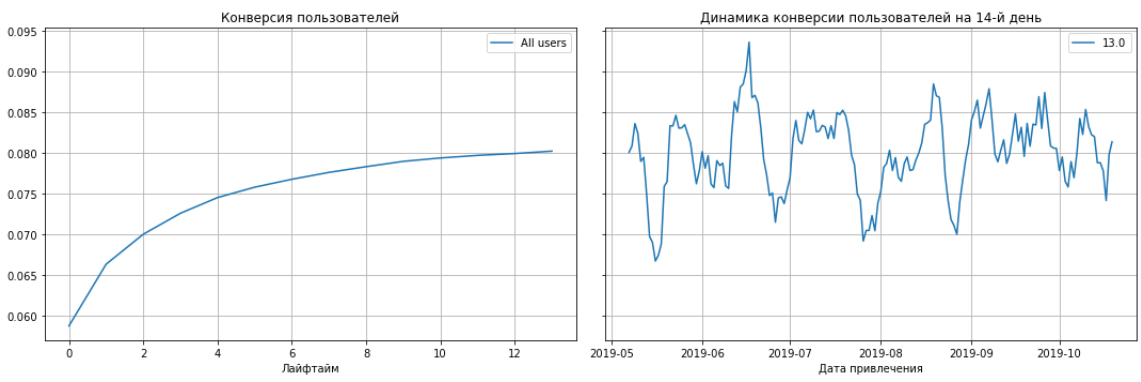
```
# функция для визуализации конверсии

def plot_conversion(conversion, conversion_history, horizon, window=7):

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

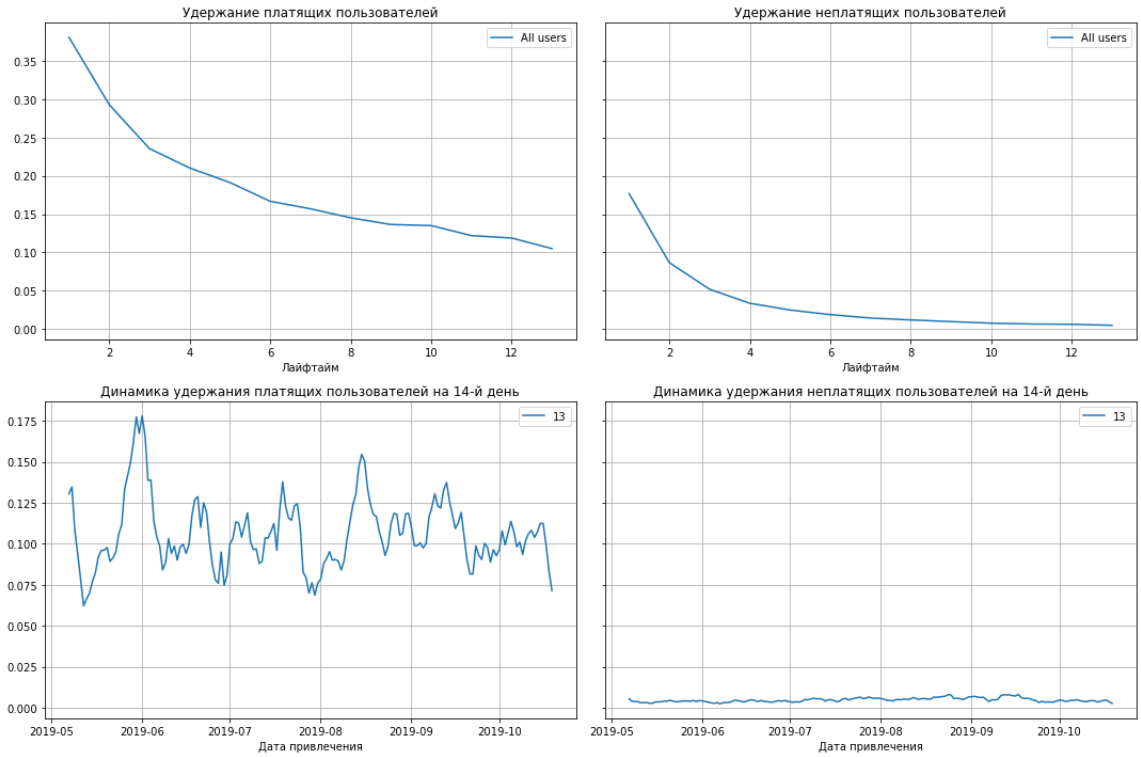
```
    # исключаем размеры когорт
    conversion = conversion.drop(columns=['cohort_size'])
    # в таблице динамики оставляем только нужный лайфтайм
    conversion_history = conversion_history.drop(columns=['cohort_size']
        [horizon - 1]
    ]

    # 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 да
        name for name in conversion_history.index.names if name not in [
    ]
    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(hor

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

Теперь посчитаем конверсию и удержание с разбивкой по параметрам

Конверсия и удержание по устройствам

In [65]:
```
# смотрим конверсию и удержание с разбивкой по устройствам
dimensions = ['device']

# Конверсия
conversion_raw, conversion_grouped, conversion_history = get_conversion(
    profiles_without_organic, orders, observation_date, horizon_days, di
)

plot_conversion(conversion_grouped, conversion_history, horizon_days)


# Удержание
retention_raw, retention_grouped, retention_history = get_retention(
    profiles_without_organic, visits, observation_date, horizon_days, di
)

plot_retention(retention_grouped, retention_history, horizon_days)
```





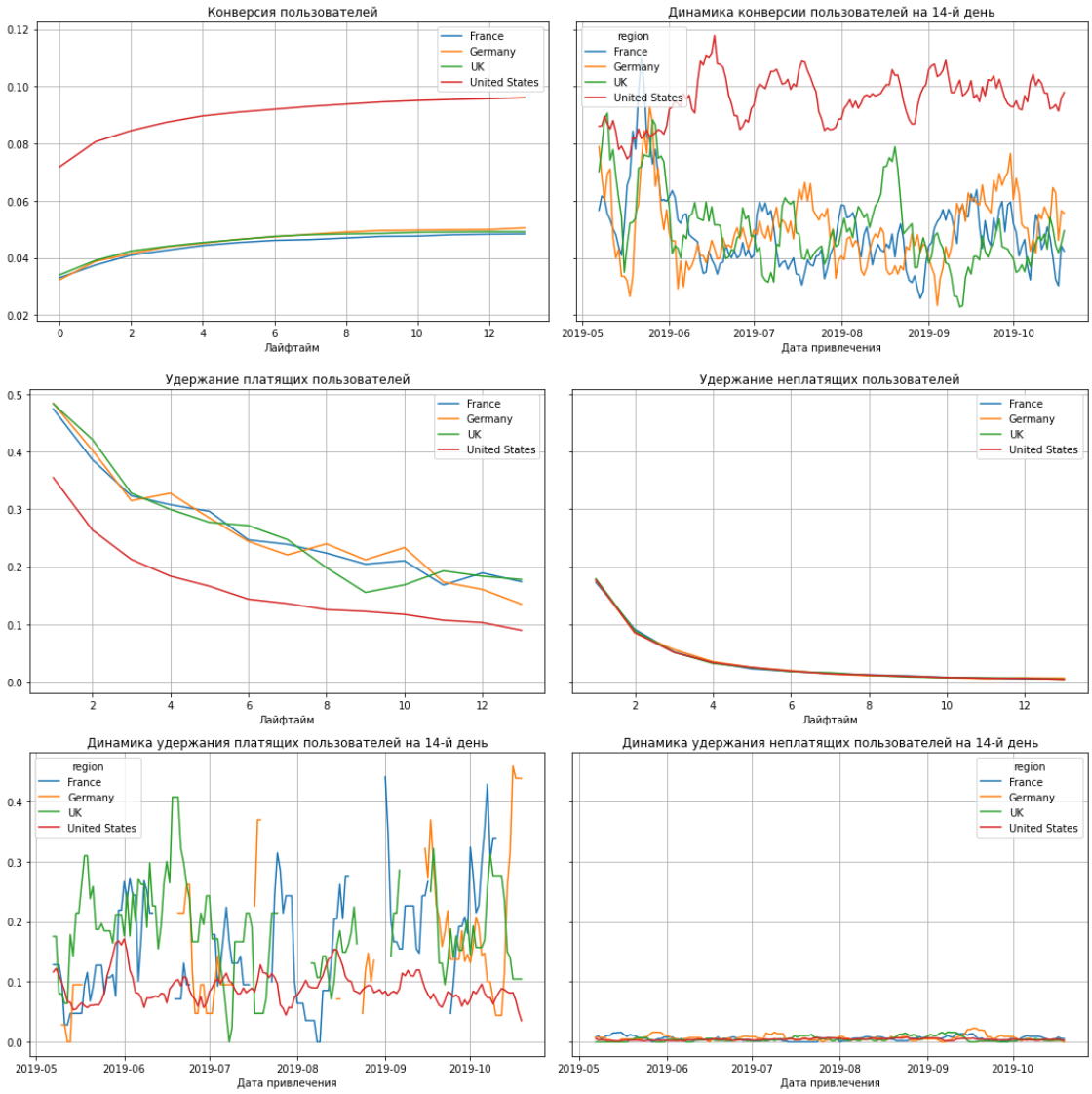
Результаты достаточно любопытны

- Пользователи стабильно неплохо конвертируются. Немного отстают пользователи PC.
- Диамика удержания платящих пользоватей также достаточно стабильна.
- Пользователи Android из когорты середины июня показали нулевой уровень удержания на 14й день. Возможно, были технические проблемы

Конверсия и удержание по странам

In [58]:

```
# смотрим конверсию и удержание с разбивкой по странам
dimensions = ['region']

# Конверсия
conversion_raw, conversion_grouped, conversion_history = get_conversion
    profiles_without_organic, orders, observation_date, horizon_days, d
)

plot_conversion(conversion_grouped, conversion_history, horizon_days)


# Удержание
retention_raw, retention_grouped, retention_history = get_retention(
    profiles_without_organic, visits, observation_date, horizon_days, d
)

plot_retention(retention_grouped, retention_history, horizon_days)
```

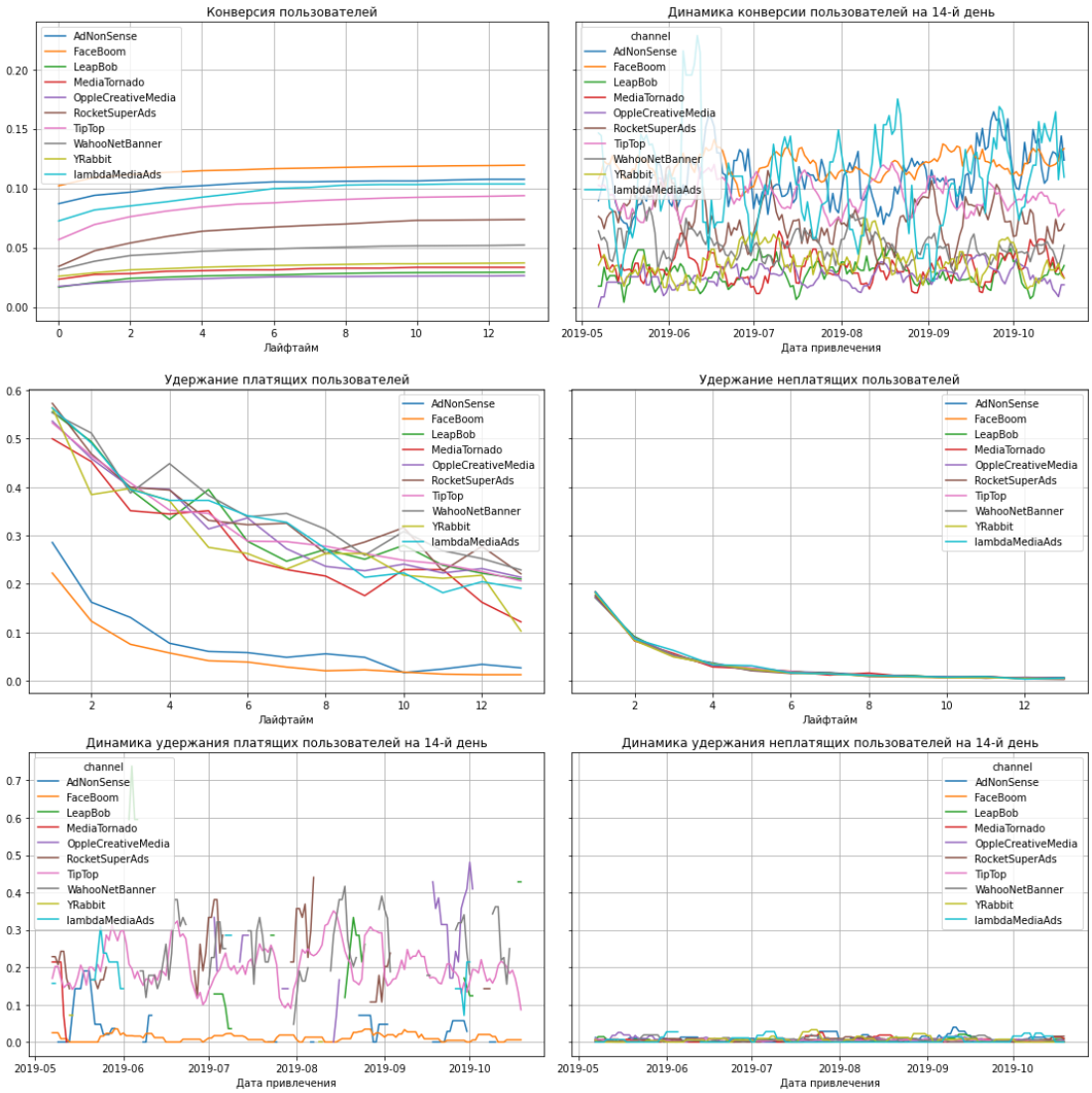


Пользователи из США конвертируются лучше остальных, а вот удерживать пользователей из США получается хуже
Лидеры по удержанию Англия, Германия и Франция

Можно заметить, что Французы показали нулевое удержание для когорты в начале августа

Конверсия и удержание по каналам

In [59]:
```
# смотрим конверсию и удержание с разбивкой по каналам
dimensions = ['channel']

# Конверсия
conversion_raw, conversion_grouped, conversion_history = get_conversio
    profiles_without_organic, orders, observation_date, horizon_days,
)

plot_conversion(conversion_grouped, conversion_history, horizon_days)


# Удержание
retention_raw, retention_grouped, retention_history = get_retention(
    profiles_without_organic, visits, observation_date, horizon_days,
)

plot_retention(retention_grouped, retention_history, horizon_days)
```



Лучше всего конвертируются пользователи FaceBoom и AdNonSense. Однако удерживаются эти пользователи хуже всех. Пользователи из остальных каналов привлечения показывают стабильную динамику удержания.

Общие выводы

Была проделана большая аналитическая работа.

В нашем распоряжении были данные:

- Логи с сервера
- Выгрузка покупок
- Рекламные расходы

Данные были изучены и предобработаны.

В ходе работы мы выяснили:

- Откуда приходят и какими устройствами пользуются пользователи сервиса.
- Долю платящих клиентов в разбивке по странам, устройствам и каналам привлечения

- Стоимость привлечения пользователей
- Когда окупаются затраты на рекламу

Также был проведен EDA.

Посчитали сумму расходов и оценили окупаемость рекламы.

Сейчас я готов дать свои рекомендации отделу маркетинга

Рекомендации

При определении рекомендаций будем учитывать то, что в нашей организации принято считать, что окупаемость должна наступать не позднее, чем через 2 недели после привлечения пользователей.
В целом окупаемость не наступает к 14му дню после привлечения пользователей. Но есть нюансы, исправив которые можно увеличить показатель ROI:

- Особое внимание нужно уделить пользователям из США. Именно этих пользователей больше всего и именно они хуже всего окупаются.
- Лидерами по окупаемости являются пользователи UK. Германия и Франция несильно отстают от них.
- В связи с большим увеличением средств на рекламу в TipTop, окупаемость сильно упала. Компания не получила большой прибыли, при этом потратив много средств. Необходимо скорректировать рекламную политику в отношении этого источника привлчения.
- Также стабильно плохо окупаются пользователи, которые пришли из FaceBoom и AdNonSense. Необходимо обратить на это внимание
- Необходимо также помнить, что пользователи этих каналов самые дорогие по стоимости привлечения одного пользователя.

В целом, я считаю, что для начала будет верным сильно скорректировать затраты на рекламу в TipTop, тк результаты без пользователей из этого канала привлечения выглядят намного лучше. В таком сценарии пользователи окупаются в среднем на 2-6 день.