Marketing Analysis Project

Project's description

We have 3 csv datasets from Yandex Afisha database (Yandex Afisha is the biggest russian prlatform created to aggregate tickets to different kind of actions like music festivals, cinemas, theaters and others).

Our data's describing period from ***june of 2017*** to ***may of 2018*** and contains:

1. Information about visits on the website;
2. Orders by the period;
3. Advertising costs statistics.

Monetary units in form of USD.

Our goal is to find the way how we can pull advertising costs down (refuse from non-benefitable traffic sources and redistribute the budget).

What should we define in our project:

1. How costumers use our platform;
2. When they usually make the first orders on the website;
3. How much each costumer brings to our company;
4. When costs for costumers attraction pay off.

Data loading and preparation to analysis process

In [3]:
```python
#setting up the libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import warnings
!pip install nbconvert

#making warnings hidden
warnings.filterwarnings('ignore')
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: nbconvert in /opt/conda/lib/python3.7/site-packa
ges (5.5.0)
Requirement already satisfied: traitlets>=4.2 in /opt/conda/lib/python3.7/site-
packages (from nbconvert) (5.0.5)
Requirement already satisfied: testpath in /opt/conda/lib/python3.7/site-packag
es (from nbconvert) (0.4.4)
Requirement already satisfied: jupyter-core in /opt/conda/lib/python3.7/site-pa
ckages (from nbconvert) (4.6.2)
Requirement already satisfied: defusedxml in /opt/conda/lib/python3.7/site-pack
ages (from nbconvert) (0.6.0)
Requirement already satisfied: entrypoints>=0.2.2 in /opt/conda/lib/python3.7/s
ite-packages (from nbconvert) (0.3)
Requirement already satisfied: nbformat>=4.4 in /opt/conda/lib/python3.7/site-p
ackages (from nbconvert) (4.4.0)
Requirement already satisfied: bleach in /opt/conda/lib/python3.7/site-packages
(from nbconvert) (3.1.5)
Requirement already satisfied: jinja2>=2.4 in /opt/conda/lib/python3.7/site-pac
kages (from nbconvert) (2.11.2)
Requirement already satisfied: pandocfilters>=1.4.1 in /opt/conda/lib/python3.
7/site-packages (from nbconvert) (1.4.2)
Requirement already satisfied: pygments in /opt/conda/lib/python3.7/site-packag
es (from nbconvert) (2.6.1)
Requirement already satisfied: mistune>=0.8.1 in /opt/conda/lib/python3.7/site-
packages (from nbconvert) (0.8.4)
Requirement already satisfied: ipython-genutils in /opt/conda/lib/python3.7/sit
e-packages (from traitlets>=4.2->nbconvert) (0.2.0)
Requirement already satisfied: jsonschema!=2.5.0,>=2.4 in /opt/conda/lib/python
3.7/site-packages (from nbformat>=4.4->nbconvert) (3.2.0)
Requirement already satisfied: six>=1.9.0 in /opt/conda/lib/python3.7/site-pack
ages (from bleach->nbconvert) (1.15.0)
Requirement already satisfied: webencodings in /opt/conda/lib/python3.7/site-pa
ckages (from bleach->nbconvert) (0.5.1)
Requirement already satisfied: packaging in /opt/conda/lib/python3.7/site-packa
ges (from bleach->nbconvert) (20.4)
Requirement already satisfied: MarkupSafe>=0.23 in /opt/conda/lib/python3.7/sit
e-packages (from jinja2>=2.4->nbconvert) (1.1.1)
Requirement already satisfied: setuptools in /opt/conda/lib/python3.7/site-pack
ages (from jsonschema!=2.5.0,>=2.4->nbformat>=4.4->nbconvert) (49.6.0.post20210
108)
Requirement already satisfied: pyrsistent>=0.14.0 in /opt/conda/lib/python3.7/s
ite-packages (from jsonschema!=2.5.0,>=2.4->nbformat>=4.4->nbconvert) (0.17.3)
Requirement already satisfied: attrs>=17.4.0 in /opt/conda/lib/python3.7/site-p
ackages (from jsonschema!=2.5.0,>=2.4->nbformat>=4.4->nbconvert) (19.1.0)
Requirement already satisfied: importlib-metadata; python_version < "3.8" in /o
```

```
pt/conda/lib/python3.7/site-packages (from jsonschema!=2.5.0,>=2.4->nbformat>=
4.4->nbconvert) (1.7.0)
Requirement already satisfied: pyparsing>=2.0.2 in /opt/conda/lib/python3.7/sit
e-packages (from packaging->bleach->nbconvert) (2.4.7)
Requirement already satisfied: zipp>=0.5 in /opt/conda/lib/python3.7/site-packa
ges (from importlib-metadata; python_version < "3.8"->jsonschema!=2.5.0,>=2.4->
nbformat>=4.4->nbconvert) (3.1.0)
```

In [4]:

```python
#uploading of datasets
visits = pd.read_csv('/datasets/visits_log.csv')
orders = pd.read_csv('/datasets/orders_log.csv')
costs = pd.read_csv('/datasets/costs.csv')
```

In [5]:
```
#making cycle to meet with data

list = [visits, orders, costs]
for i in list:
    display(i.head())
    i.info()
    display(i.duplicated().sum())
    display(i.isna().sum())
    print('     ')
    print('----- The End of information about table in order -----')
    print('     ')
```

	Device	End Ts	Source Id	Start Ts	Uid
0	touch	2017-12-20 17:38:00	4	2017-12-20 17:20:00	16879256277535980062
1	desktop	2018-02-19 17:21:00	2	2018-02-19 16:53:00	104060357244891740
2	touch	2017-07-01 01:54:00	5	2017-07-01 01:54:00	7459035603376831527
3	desktop	2018-05-20 11:23:00	9	2018-05-20 10:59:00	16174680259334210214
4	desktop	2017-12-27 14:06:00	3	2017-12-27 14:06:00	9969694820036681168

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 359400 entries, 0 to 359399
Data columns (total 5 columns):
Device       359400 non-null object
End Ts       359400 non-null object
Source Id    359400 non-null int64
Start Ts     359400 non-null object
Uid          359400 non-null uint64
dtypes: int64(1), object(3), uint64(1)
memory usage: 13.7+ MB
```

```
0
```

```
Device       0
End Ts       0
Source Id    0
Start Ts     0
Uid          0
dtype: int64
```

```
----- The End of information about table in order -----
```

	Buy Ts	Revenue	Uid
0	2017-06-01 00:10:00	17.00	10329302124590727494
1	2017-06-01 00:25:00	0.55	11627257723692907447
2	2017-06-01 00:27:00	0.37	17903680561304213844
3	2017-06-01 00:29:00	0.55	16109239769442553005
4	2017-06-01 07:58:00	0.37	14200605875248379450

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50415 entries, 0 to 50414
Data columns (total 3 columns):
Buy Ts     50415 non-null object
Revenue    50415 non-null float64
Uid        50415 non-null uint64
dtypes: float64(1), object(1), uint64(1)
memory usage: 1.2+ MB
```

```
0
```

```
Buy Ts     0
Revenue    0
Uid        0
dtype: int64


----- The End of information about table in order -----
```

	source_id	dt	costs
0	1	2017-06-01	75.20
1	1	2017-06-02	62.25
2	1	2017-06-03	36.53
3	1	2017-06-04	55.00
4	1	2017-06-05	57.08

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2542 entries, 0 to 2541
Data columns (total 3 columns):
source_id    2542 non-null int64
dt           2542 non-null object
costs        2542 non-null float64
dtypes: float64(1), int64(1), object(1)
memory usage: 59.7+ KB
```

```
0
```

```
source_id    0
dt           0
costs        0
dtype: int64


----- The End of information about table in order -----
```

Apparently quality of data is good. There's no gaps and duplicates. We should change format of date from "object" to "datetime", convert data to lowercase and we'll ready to start analysis.

In [6]:
```
visits.columns = ['device', 'end_ts', 'source_id', 'start_ts', 'uid']
display(visits.head())
```

	device	end_ts	source_id	start_ts	uid
0	touch	2017-12-20 17:38:00	4	2017-12-20 17:20:00	16879256277535980062
1	desktop	2018-02-19 17:21:00	2	2018-02-19 16:53:00	104060357244891740
2	touch	2017-07-01 01:54:00	5	2017-07-01 01:54:00	7459035603376831527
3	desktop	2018-05-20 11:23:00	9	2018-05-20 10:59:00	16174680259334210214
4	desktop	2017-12-27 14:06:00	3	2017-12-27 14:06:00	9969694820036681168

In [7]:
```
orders.columns = ['buy_ts', 'revenue', 'uid']
display(orders.head())
```

	buy_ts	revenue	uid
0	2017-06-01 00:10:00	17.00	10329302124590727494
1	2017-06-01 00:25:00	0.55	11627257723692907447
2	2017-06-01 00:27:00	0.37	17903680561304213844
3	2017-06-01 00:29:00	0.55	16109239769442553005
4	2017-06-01 07:58:00	0.37	14200605875248379450

In [8]:
```
costs.columns = ['source_id', 'date_marketing', 'costs']
display(costs.head())
```

	source_id	date_marketing	costs
0	1	2017-06-01	75.20
1	1	2017-06-02	62.25
2	1	2017-06-03	36.53
3	1	2017-06-04	55.00
4	1	2017-06-05	57.08

In [9]:
```
visits['end_ts'] = pd.to_datetime(visits['end_ts'], format='%Y.%m.%d %H:%M:%S')
visits['start_ts'] = pd.to_datetime(visits['start_ts'], format='%Y.%m.%d %H:%M:%S
orders['buy_ts'] = pd.to_datetime(orders['buy_ts'], format='%Y.%m.%d')
costs['date_marketing'] = pd.to_datetime(costs['date_marketing'], format='%Y.%m.%
```

Metrics estimation and graph's building

Product's metrics

Let's count DAU, MAU and WAU metrics (daily, monthly and weekly active users).

In [10]:
```
#select time periods
visits['ssn_date'] = visits['start_ts'].dt.date
visits['ssn_week'] = visits['start_ts'].dt.week
visits['ssn_month'] = visits['start_ts'].dt.month
visits['ssn_year'] = visits['start_ts'].dt.year
```

In [11]:
```
#count metrics
dau = visits.groupby('ssn_date').agg({'uid':'nunique'})
display(int(dau.mean()))
```

```
907
```

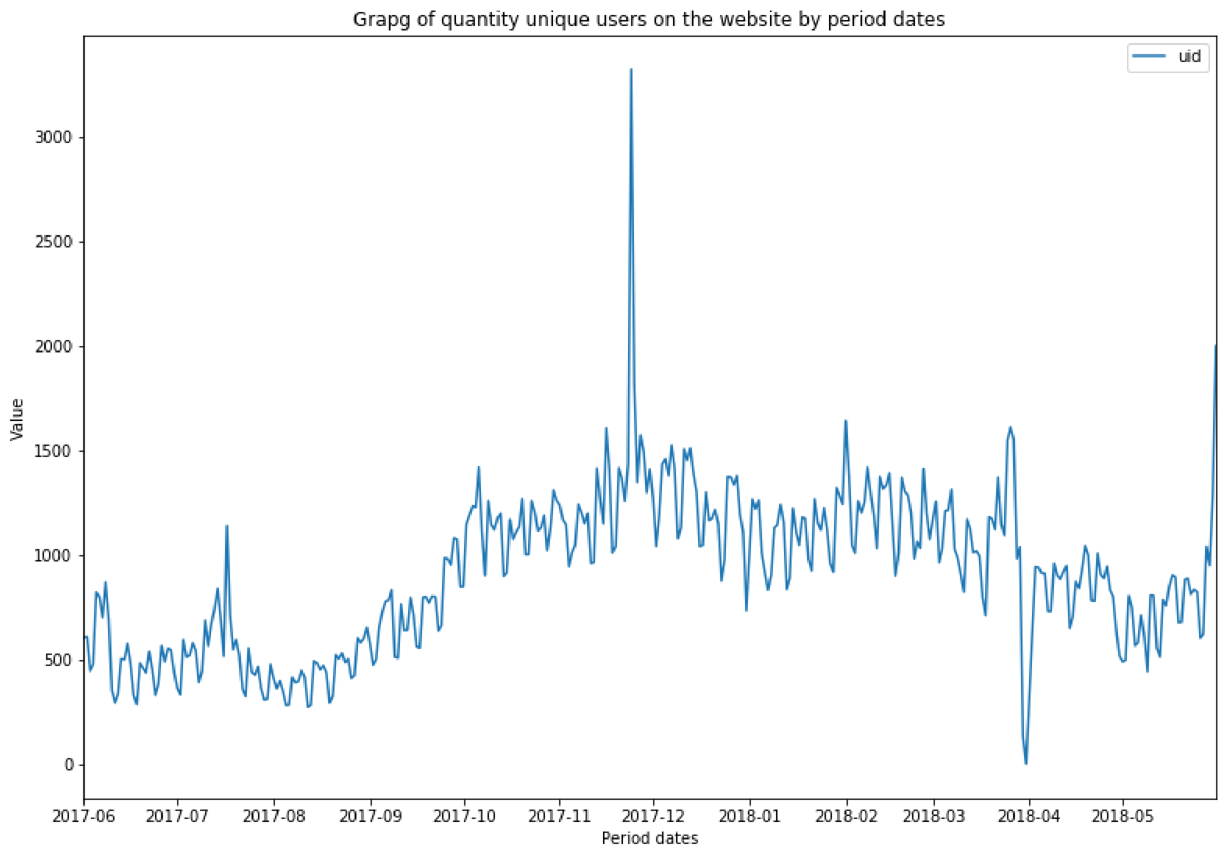
In [12]:
```
wau = visits.groupby(['ssn_year', 'ssn_week']).agg({'uid':'nunique'})
display(int(wau.mean()))
```

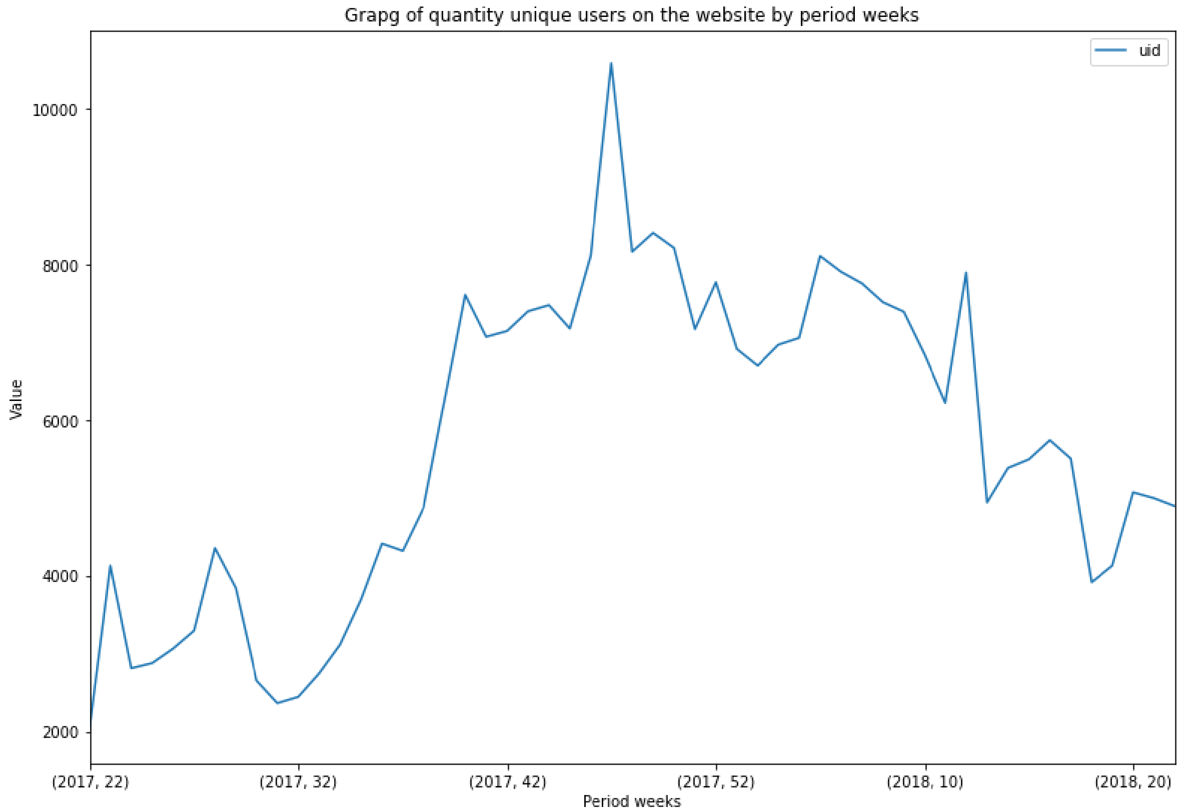
```
5716
```

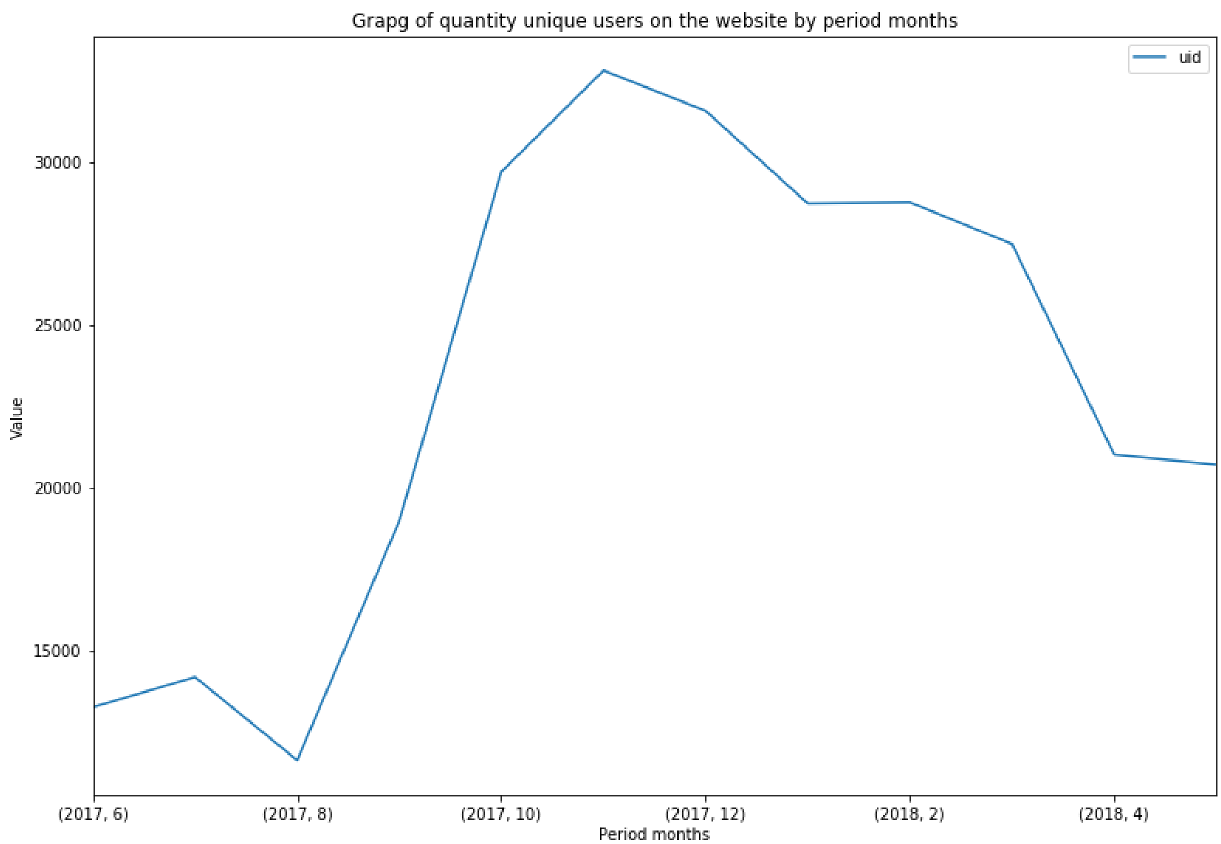
In [13]:
```
mau = visits.groupby(['ssn_year', 'ssn_month']).agg({'uid':'nunique'})
display(int(mau.mean()))
```

```
23228
```

In [14]:
```
#make cycle to draw graphs
list_user_metrics = {'dates':dau, 'weeks':wau, 'months':mau}
for name, value in list_user_metrics.items():
    value.plot(figsize=(13,9))
    plt.title('Grapg of quantity unique users on the website by period ' + name)
    plt.ylabel('Value')
    plt.xlabel('Period ' + name)
```







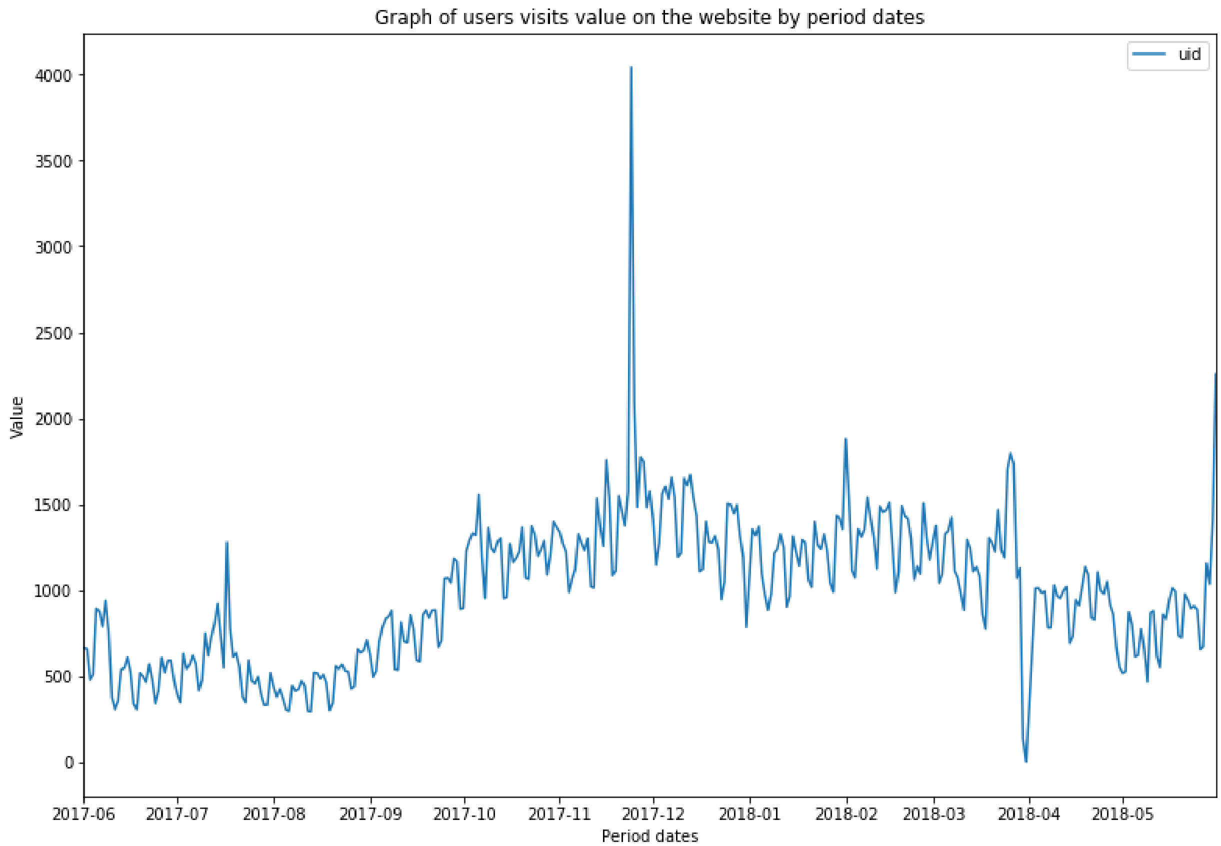
Metrics have periods of abnormal activity that most likely related to seasonal circumstances. For example in March 2018 we can see a rapid growth in user visits and then a rapid decline. Perhaps this is due to International Women's Day. As you know such purchases (tickets to movies, performances, etc.) have a clearly defined periodicity. If you take a user, you can find that he buys tickets (to the movies for example) with a certain frequency. Each client has its own strategy, but if an event occurs that forces the user to break their habits, their activity frequency is disrupted and we can see an uneven distribution of visits for this period on the graph. By analogy this works with other examples.

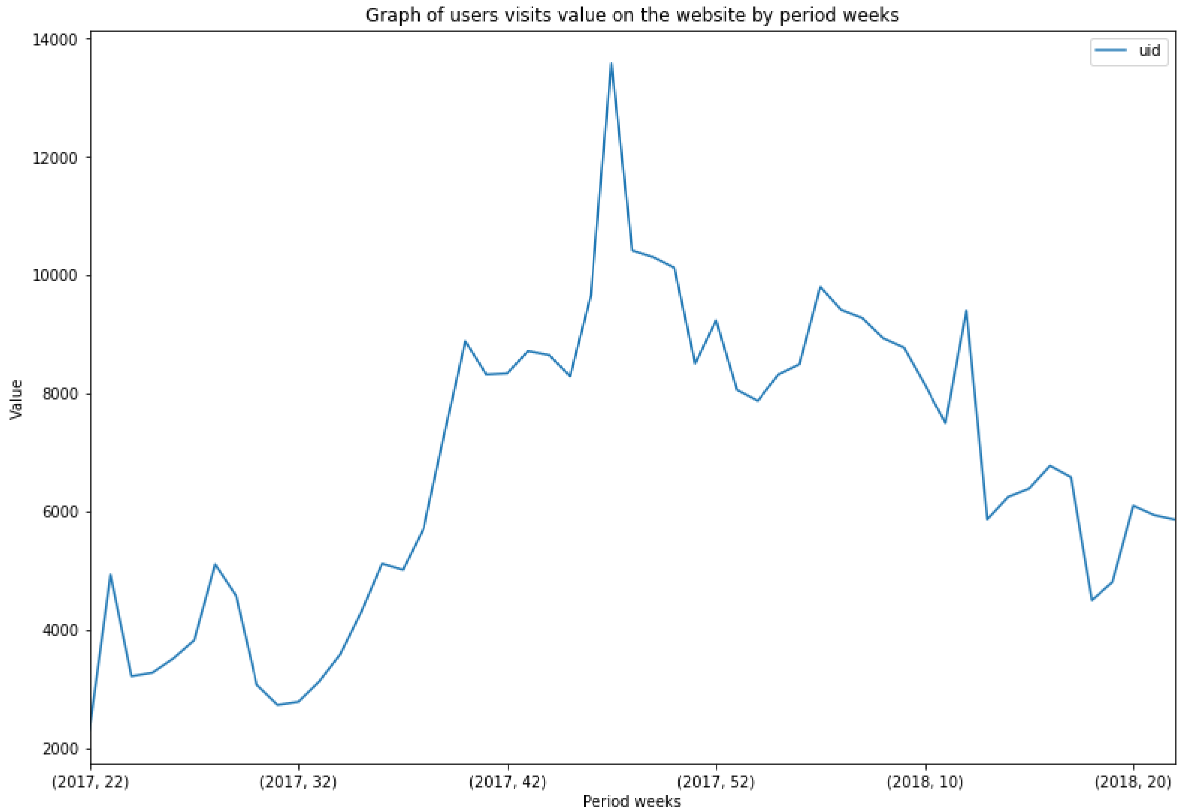
The general periods of the greatest user activity are similar - from the end of August to the end of December and then the decline begins. This is due to the return of users from holidays after August and the gradual change in seasonal activities.

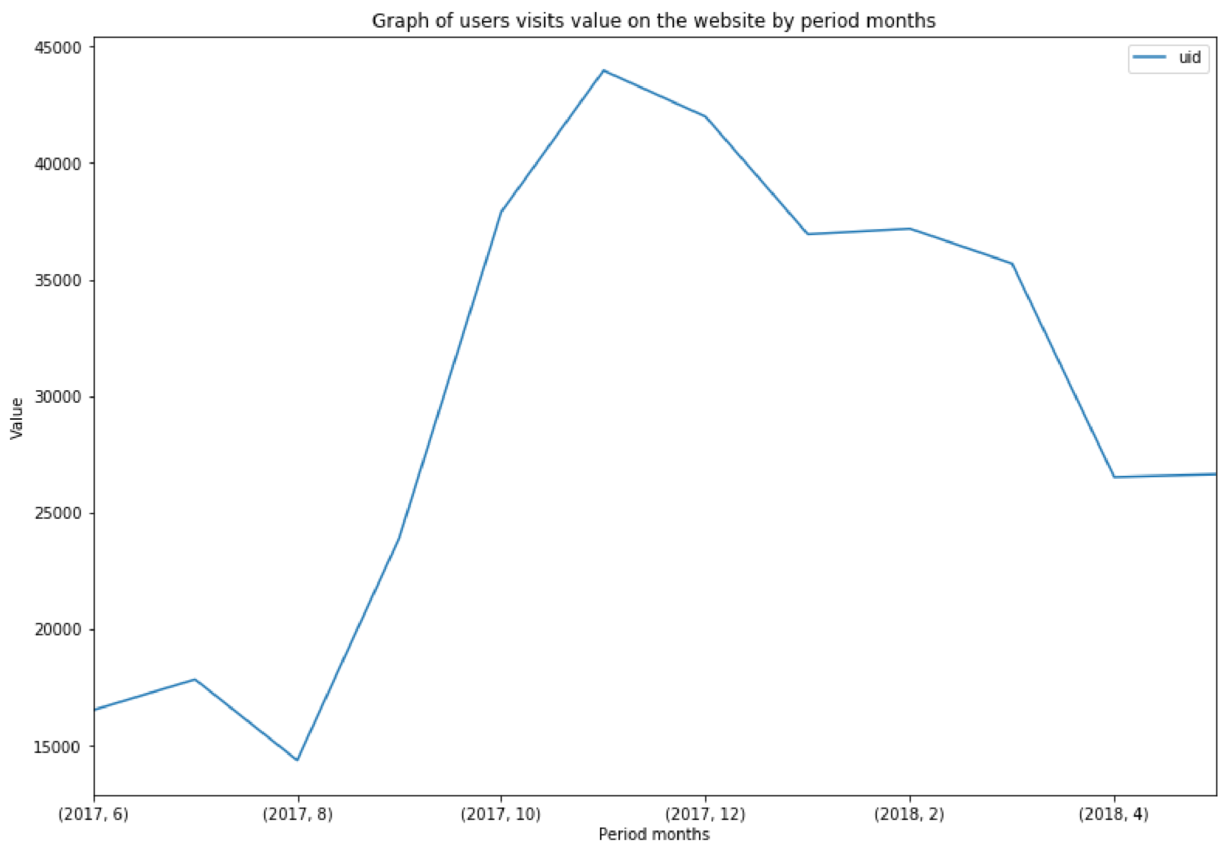
Let's estimate value of visits by period

In [15]:
```
#group our data by period and count uid
date_cnt = visits.groupby('ssn_date').agg({'uid':'count'})
week_cnt = visits.groupby(['ssn_year', 'ssn_week']).agg({'uid':'count'})
month_cnt = visits.groupby(['ssn_year', 'ssn_month']).agg({'uid':'count'})
```

In [16]:
```
#making cycle to draw graphs
list_user_metrics = {'dates':date_cnt, 'weeks':week_cnt, 'months':month_cnt}
for name, value in list_user_metrics.items():
    value.plot(figsize=(13,9))
    plt.title('Graph of users visits value on the website by period ' + name)
    plt.ylabel('Value')
    plt.xlabel('Period ' + name)
```



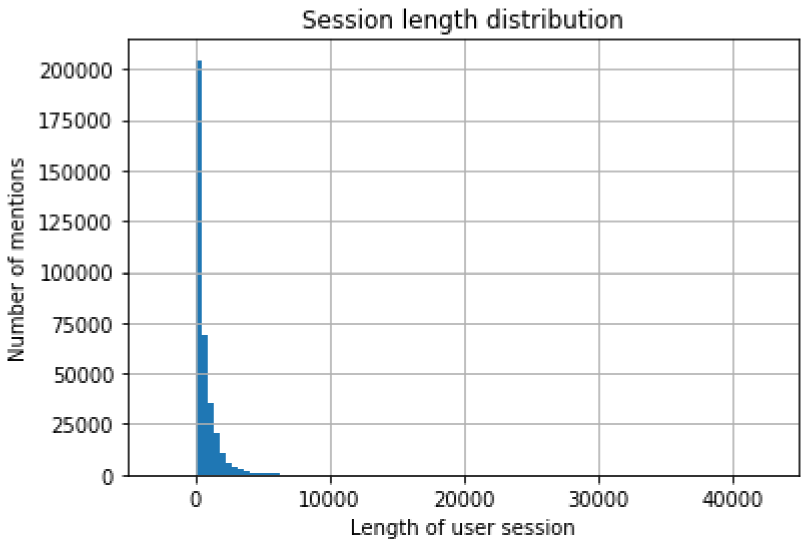




Let's count ASL (length of session activity) and build graph of middle length of user session.

In [17]:
```
visits['ssn_dur_sec'] = ( visits['end_ts'] - visits['start_ts'] ).dt.total_second
visits['ssn_dur_sec'].hist(bins=100)
plt.title('Session length distribution')
plt.ylabel('Number of mentions')
plt.xlabel('Length of user session')
```

Out[17]: Text(0.5, 0, 'Length of user session')



In [18]:
```
asl = visits['ssn_dur_sec'].mode()
display(asl)
```

```
0    60.0
dtype: float64
```

In [19]:
```
visits['ssn_dur_sec'].describe()
```

Out[19]:
```
count    359400.000000
mean        643.025687
std         997.127761
min       -2760.000000
25%         120.000000
50%         300.000000
75%         840.000000
max       42660.000000
Name: ssn_dur_sec, dtype: float64
```

In [20]:
```
first_activity_date = orders.groupby(['uid'])['buy_ts'].min()
first_activity_date.name = 'first_activity_date'
orders = orders.join(first_activity_date, on='uid')
orders.head()
```

Out[20]:

	buy_ts	revenue	uid	first_activity_date
0	2017-06-01 00:10:00	17.00	10329302124590727494	2017-06-01 00:10:00
1	2017-06-01 00:25:00	0.55	11627257723692907447	2017-06-01 00:25:00
2	2017-06-01 00:27:00	0.37	17903680561304213844	2017-06-01 00:27:00
3	2017-06-01 00:29:00	0.55	16109239769442553005	2017-06-01 00:29:00
4	2017-06-01 07:58:00	0.37	14200605875248379450	2017-06-01 07:58:00

Select user cohorts and estimate Retention Rate

In [21]:
```
orders['activity_month'] = orders['buy_ts'].astype('datetime64[M]')
orders['first_activity_month'] = orders['first_activity_date'].astype('datetime64

orders['cohort_lifetime'] =(
    orders['activity_month'] - orders['first_activity_month']
)
orders['cohort_lifetime'] = (
    orders['cohort_lifetime'] / np.timedelta64(1, 'M')
)
orders['cohort_lifetime'] = orders['cohort_lifetime'].round().astype('int')


cohorts = orders.groupby(['first_activity_month', 'cohort_lifetime']).agg({'uid':
display(cohorts)
```

	first_activity_month	cohort_lifetime	uid
0	2017-06-01	0	2023
1	2017-06-01	1	61
2	2017-06-01	2	50
3	2017-06-01	3	54
4	2017-06-01	4	88
...	...	...	...
74	2018-03-01	2	58
75	2018-04-01	0	2276
76	2018-04-01	1	69
77	2018-05-01	0	2988
78	2018-06-01	0	1

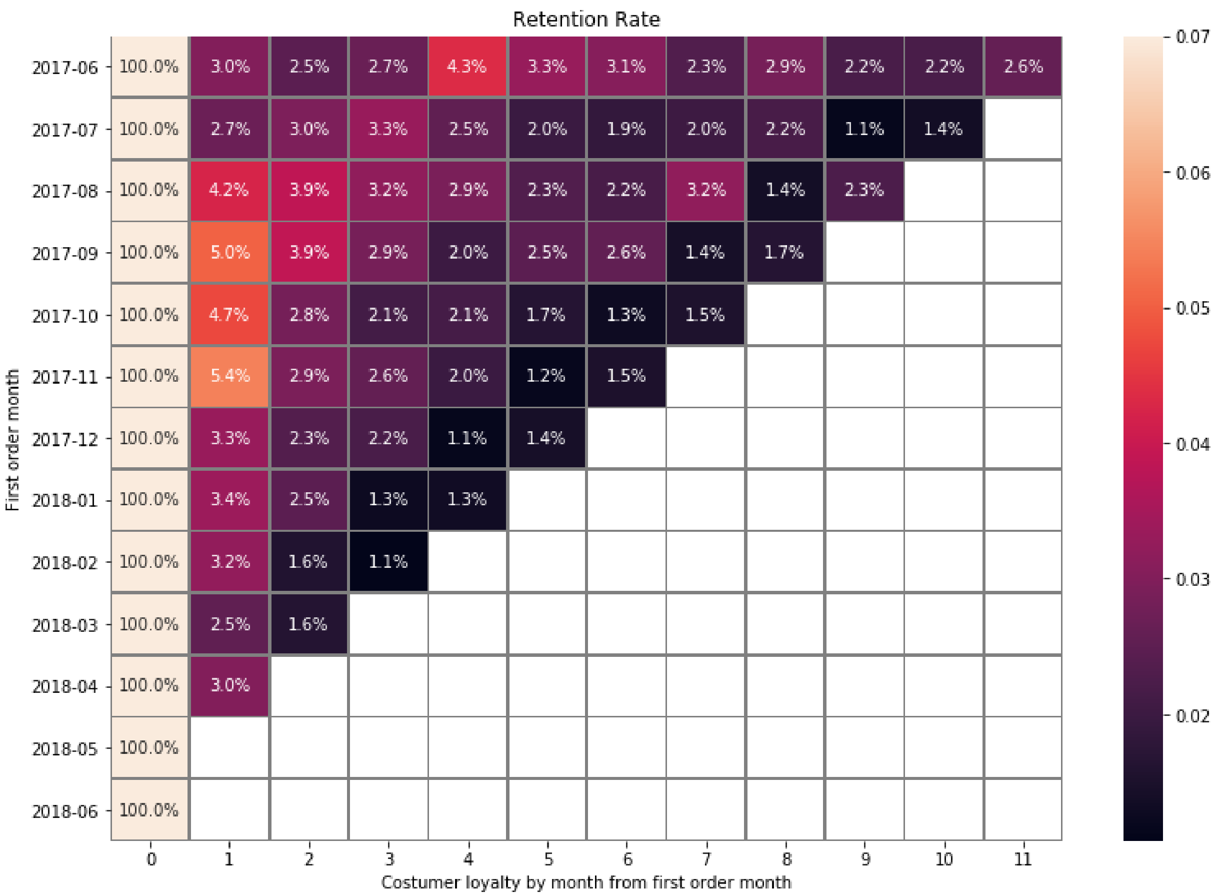
79 rows × 3 columns

In [22]:
```
initial_uid = cohorts[cohorts['cohort_lifetime'] == 0][
    ['first_activity_month', 'uid']
]
initial_uid = initial_uid.rename(columns={'uid':'cohort_uid'})

cohorts = cohorts.merge(initial_uid, on='first_activity_month')

display(cohorts)
```

	first_activity_month	cohort_lifetime	uid	cohort_uid
0	2017-06-01	0	2023	2023
1	2017-06-01	1	61	2023
2	2017-06-01	2	50	2023
3	2017-06-01	3	54	2023
4	2017-06-01	4	88	2023
...	...	...	...	...
74	2018-03-01	2	58	3533
75	2018-04-01	0	2276	2276
76	2018-04-01	1	69	2276
77	2018-05-01	0	2988	2988
78	2018-06-01	0	1	1

79 rows × 4 columns

In [23]:
```
cohorts['first_activity_month'] = cohorts['first_activity_month'].dt.strftime('%Y
cohorts['retention'] = cohorts['uid'] / cohorts['cohort_uid']

retention_pivot = cohorts.pivot_table(
    index='first_activity_month',
    columns='cohort_lifetime',
    values='retention',
    aggfunc='mean'
)

plt.figure(figsize=(13,9))
sns.heatmap(retention_pivot, fmt='.1%', annot=True, linewidth=0.7, linecolor='gre
plt.title('Retention Rate')
plt.ylabel('First order month')
plt.xlabel('Costumer loyalty by month from first order month')
plt.show()
```



Apparently the number of unique users as well as the number of visits to Yandex.Afisha increases from August to the end of the year and then decreases by the summer period.

Active peaks of visits and orders are August, September, November, and May.

I think that in August user activity may increase due to the end of the holiday season and by the end of the year and in May due to the approaching holidays

If you do not take into account the low percentage of repeat orders in general the Retention Rate is best preserved for users who made their first purchases from May to August.

In [24]:
```
print('The average RR for the period of 2 months of life was -', (retention_pivot
```

```
The average RR for the period of 2 months of life was - 3.7
```

E-commerce metrics

Let's see how much time passes from the first visit to the purchase

In [25]:
```
orders_grouped = orders.groupby('uid')['buy_ts'].min()
orders_grouped.name = 'orders_ts'
time_visit_order = visits.join(orders_grouped, how='left', on='uid')
```

In [26]:
```
time_visit_order = time_visit_order[time_visit_order['orders_ts'] > time_visit_or
time_visit_order.head()
```

Out[26]:

	device	end_ts	source_id	start_ts	uid	ssn_date	ssn_week	ssn_month
5	desktop	2017-09-03 21:36:00	5	2017-09-03 21:35:00	16007536194108375387	2017-09-03	35	9
15	touch	2018-02-12 20:30:00	2	2018-02-12 19:24:00	18188358787673499603	2018-02-12	7	2
27	desktop	2017-10-23 12:58:00	3	2017-10-23 12:49:00	4499746016005494365	2017-10-23	43	10
37	desktop	2018-03-12 23:25:00	3	2018-03-12 23:13:00	15857957287537270437	2018-03-12	11	3
38	touch	2018-03-01 08:45:00	4	2018-03-01 08:43:00	15763368622958393183	2018-03-01	9	3

In [27]:
```
time_visit_order['time_visit_order'] = (time_visit_order['orders_ts'] - time_visi
time_visit_order['time_visit_order'] = time_visit_order['time_visit_order'] / 60

display(time_visit_order['time_visit_order'].median())
display(time_visit_order['time_visit_order'].min())
display(time_visit_order['time_visit_order'].max())
```

```
32.0
```

```
0.0
```

```
1439.0
```

On average it takes 32 minutes from the start of the session to the purchase

---

Let's calculate the average number of purchases per customer for 6 months

In [28]:
```
orders.head()
```

Out[28]:

	buy_ts	revenue	uid	first_activity_date	activity_month	first_activity_month
0	2017-06-01 00:10:00	17.00	10329302124590727494	2017-06-01 00:10:00	2017-06-01	2017-06-01
1	2017-06-01 00:25:00	0.55	11627257723692907447	2017-06-01 00:25:00	2017-06-01	2017-06-01
2	2017-06-01 00:27:00	0.37	17903680561304213844	2017-06-01 00:27:00	2017-06-01	2017-06-01
3	2017-06-01 00:29:00	0.55	16109239769442553005	2017-06-01 00:29:00	2017-06-01	2017-06-01
4	2017-06-01 07:58:00	0.37	14200605875248379450	2017-06-01 07:58:00	2017-06-01	2017-06-01

In [29]:
```
orders['first_activity_month'] = orders['first_activity_month'].astype('datetime6
orders['activity_month'] = orders['activity_month'].astype('datetime64[M]')

cohort_sizes = (
    orders.groupby('activity_month')
    .agg({'uid': ['nunique', 'count']})
    .reset_index()
)

cohort_sizes.columns = ['activity_month', 'n_buyers', 'cnt_orders']
cohort_sizes['orders_per_user'] = cohort_sizes['cnt_orders'] / cohort_sizes['n_bu
cohort_sizes['orders_per_user'] = round(cohort_sizes['orders_per_user'], 2)

cohort_sizes = cohort_sizes.query('activity_month >= "2017-05-01" and activity_mo

display(cohort_sizes)
```

	activity_month	n_buyers	cnt_orders	orders_per_user
0	2017-06-01	2023	2354	1.16
1	2017-07-01	1984	2363	1.19
2	2017-08-01	1472	1807	1.23
3	2017-09-01	2750	3387	1.23
4	2017-10-01	4675	5679	1.21

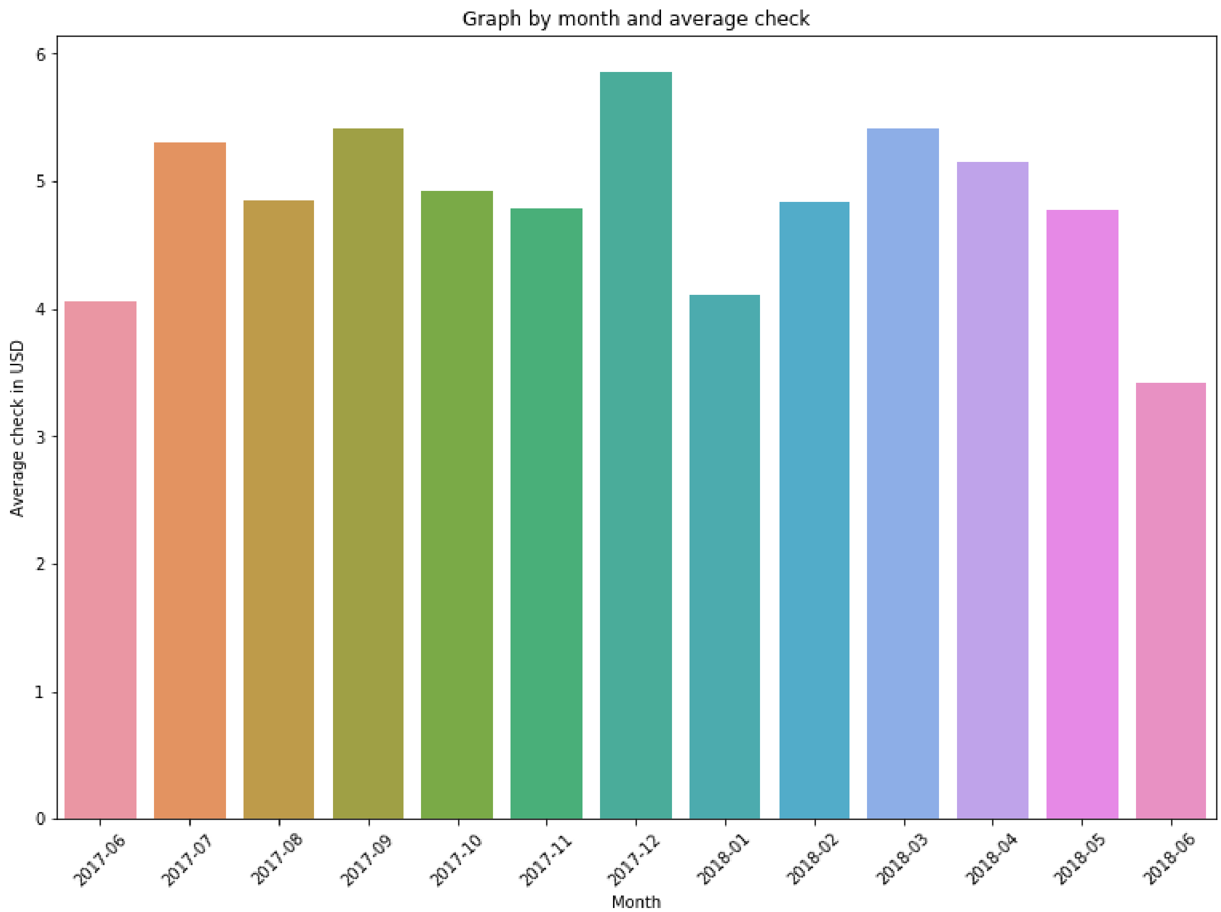
On average users make 1-2 orders per month. We have taken into account the period from May to October 2017. Despite the increase in orders by September the number of orders per user is almost unchanged which indicates a weak life expectancy of an individual client on the platform.

In [30]:
```
av_check = orders.groupby('activity_month').agg({'uid':'count', 'revenue':'sum'})

av_check['average_check'] = av_check['revenue'] / av_check['uid']
display(av_check)
```

	activity_month	uid	revenue	average_check
0	2017-06-01	2354	9557.49	4.060106
1	2017-07-01	2363	12539.47	5.306589
2	2017-08-01	1807	8758.78	4.847139
3	2017-09-01	3387	18345.51	5.416448
4	2017-10-01	5679	27987.70	4.928280
5	2017-11-01	5659	27069.93	4.783518
6	2017-12-01	6218	36388.60	5.852139
7	2018-01-01	4721	19417.13	4.112927
8	2018-02-01	5281	25560.54	4.840095
9	2018-03-01	5326	28834.59	5.413930
10	2018-04-01	3273	16858.06	5.150645
11	2018-05-01	4346	20735.98	4.771279
12	2018-06-01	1	3.42	3.420000

In [31]:
```
av_check['activity_month'] = av_check['activity_month'].dt.strftime('%Y-%m')


plt.figure(figsize=(13, 9))
sns.barplot(data=av_check ,x='activity_month', y='average_check')
plt.title('Graph by month and average check')
plt.xlabel('Month')
plt.ylabel('Average check in USD')
plt.xticks(rotation=45)
```

Out[31]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12]),
 <a list of 13 Text xticklabel objects>)
```



Let's calculate LTV (lifetime value) and build the graph

In [32]:
```
uids_sources = visits.groupby('uid', as_index=False).agg({'source_id':'first'})

orders = orders.merge(uids_sources, on='uid')
orders.head()
```

Out[32]:

	buy_ts	revenue	uid	first_activity_date	activity_month	first_activity_month
0	2017-06-01 00:10:00	17.00	10329302124590727494	2017-06-01 00:10:00	2017-06-01	2017-06-01
1	2017-06-01 00:25:00	0.55	11627257723692907447	2017-06-01 00:25:00	2017-06-01	2017-06-01
2	2017-06-01 00:27:00	0.37	17903680561304213844	2017-06-01 00:27:00	2017-06-01	2017-06-01
3	2017-06-01 00:29:00	0.55	16109239769442553005	2017-06-01 00:29:00	2017-06-01	2017-06-01
4	2017-06-01 07:58:00	0.37	14200605875248379450	2017-06-01 07:58:00	2017-06-01	2017-06-01

In [33]:

```
cohort_sizes = (
    orders.groupby(['source_id', 'first_activity_month'])
    .agg({'uid': 'nunique'})
    .reset_index()
)
cohort_sizes.columns = ['source_id', 'first_activity_month', 'n_buyers']

cohorts = (
    orders.groupby(['source_id', 'first_activity_month', 'activity_month'])
    .agg({'revenue': 'sum'})
    .reset_index()
)

report = pd.merge(cohorts, cohort_sizes, on=['source_id', 'first_activity_month']
report.head()
```

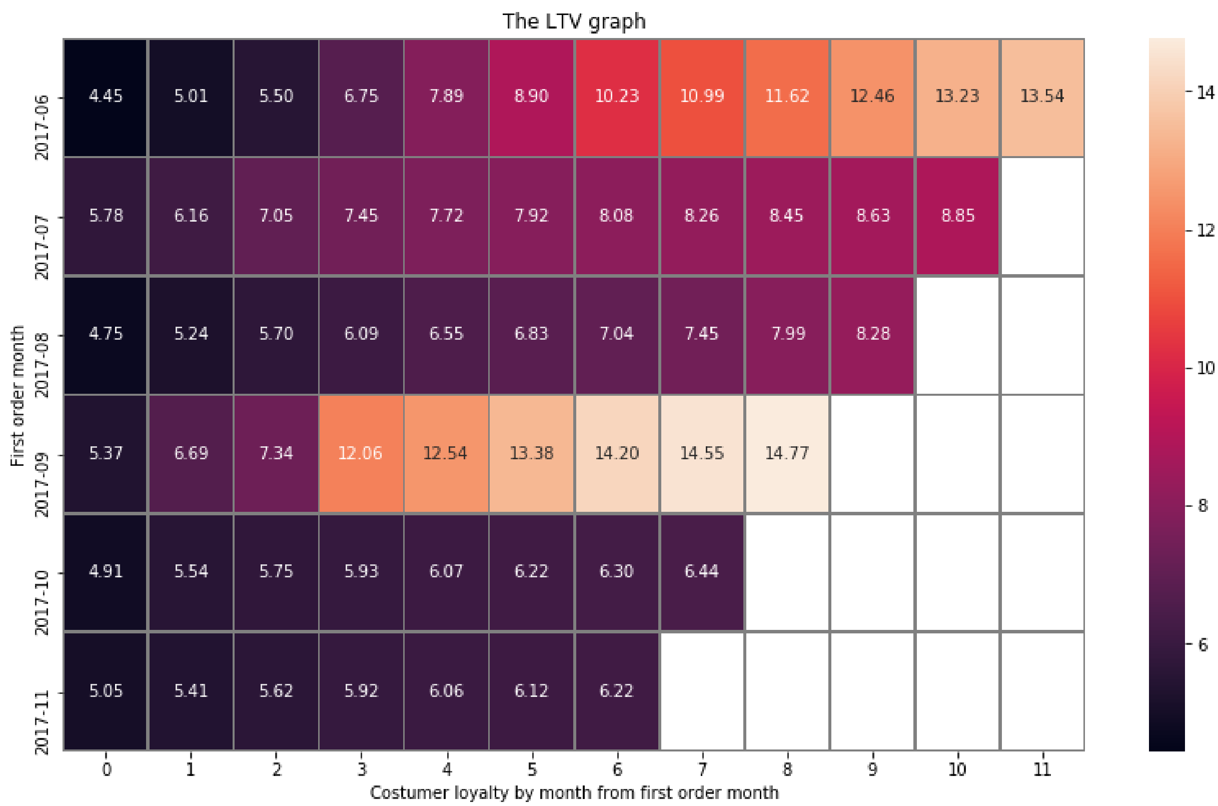
Out[33]:

	source_id	first_activity_month	activity_month	revenue	n_buyers
0	1	2017-06-01	2017-06-01	1168.45	203
1	1	2017-06-01	2017-07-01	362.94	203
2	1	2017-06-01	2017-08-01	153.72	203
3	1	2017-06-01	2017-09-01	695.88	203
4	1	2017-06-01	2017-10-01	760.74	203

In [34]:

```
margin_rate = 1

report['gp'] = report['revenue'] * margin_rate

report['age'] = (report['activity_month'] - report['first_activity_month']) / np.

report['age'] = report['age'].round().astype('int')

report['ltv'] = report['gp'] / report['n_buyers']

report['first_activity_month'] = report['first_activity_month'].dt.strftime('%Y-%
```

In [35]:
```
output = report.pivot_table(index='first_activity_month', columns='age', values='
output = output.query('index >= "2017-05" and index <= "2017-11"')

plt.figure(figsize=(14,8))
sns.heatmap(output, fmt='.2f', annot=True, linewidth=0.7, linecolor='grey')
plt.title('The LTV graph')
plt.ylabel('First order month')
plt.xlabel('Costumer loyalty by month from first order month')
plt.show()
```



In [36]:
```
m6_cum_ltv = output.mean(axis=0)[6]
print('Average LTV for 6 months after first order:', m6_cum_ltv)
```

```
Average LTV for 6 months after first order: 8.678333333333333
```

Marketing metrics

Let's calculate the total amount of marketing expenses and find out the expenses by source.

In [37]:
```
costs['month'] = costs['date_marketing'].astype('datetime64[M]')
costs.head()
```

Out[37]:

	source_id	date_marketing	costs	month
0	1	2017-06-01	75.20	2017-06-01
1	1	2017-06-02	62.25	2017-06-01
2	1	2017-06-03	36.53	2017-06-01
3	1	2017-06-04	55.00	2017-06-01
4	1	2017-06-05	57.08	2017-06-01

In [38]:
```
costs_all = costs['costs'].sum()
display(costs_all)
```

```
329131.62
```

The total amount of marketing expenses is 329131.62 USD.

In [39]:
```
costs_per_source = costs.groupby('source_id').agg({'costs':'sum'})
display(costs_per_source)
```

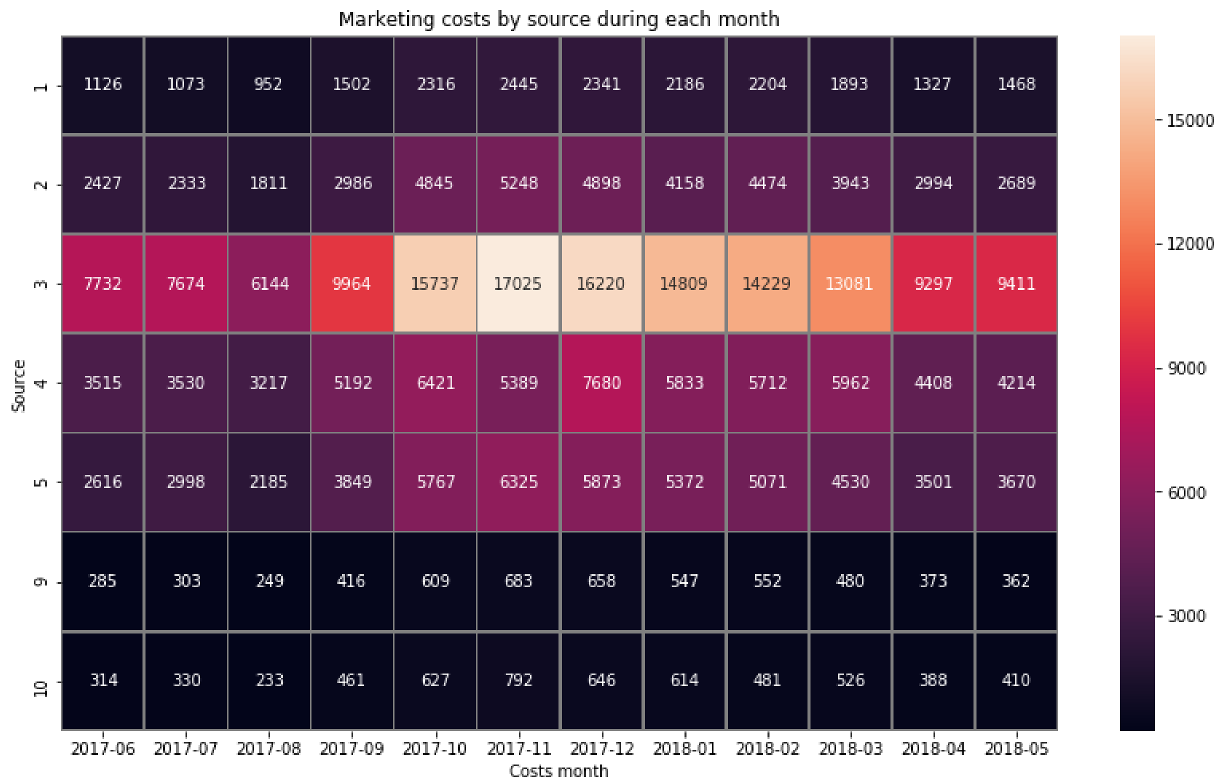
	costs
source_id	
1	20833.27
2	42806.04
3	141321.63
4	61073.60
5	51757.10
9	5517.49
10	5822.49

Source number 3 is the most expensive and source number 9 is the least expensive. The top three in terms of costs are the 3rd, 4th and 5th.

In [40]:
```
costs['month'] = costs['month'].dt.strftime('%Y-%m')
costs_pivot = costs.pivot_table(
    index='source_id',
    columns='month',
    values='costs',
    aggfunc='sum'
)

display(costs_pivot)
```

month	2017-06	2017-07	2017-08	2017-09	2017-10	2017-11	2017-12	2018-01	2018-02	
source_id										
1	1125.61	1072.88	951.81	1502.01	2315.75	2445.16	2341.20	2186.18	2204.48	
2	2427.38	2333.11	1811.05	2985.66	4845.00	5247.68	4897.80	4157.74	4474.34	
3	7731.65	7674.37	6143.54	9963.55	15737.24	17025.34	16219.52	14808.78	14228.56	1
4	3514.80	3529.73	3217.36	5192.26	6420.84	5388.82	7680.47	5832.79	5711.96	
5	2616.12	2998.14	2185.28	3849.14	5767.40	6325.34	5872.52	5371.52	5071.31	
9	285.22	302.54	248.93	415.62	609.41	683.18	657.98	547.16	551.50	
10	314.22	329.82	232.57	460.67	627.24	792.36	645.86	614.35	480.88	

In [41]:
```
plt.figure(figsize=(14,8))
sns.heatmap(costs_pivot, fmt='.0f', annot=True, linewidth=0.7, linecolor='grey')
plt.title('Marketing costs by source during each month')
plt.ylabel('Source')
plt.xlabel('Costs month')
plt.show()
```



In each source costs gradually decrease by the end of August and increase sharply from September to December. After the new year costs begin to gradually decrease.

Let's calculate in general CAC (costumer acquisition cost) for all over the project and for each source separately.

In [42]:
```
visits['month'] = visits['start_ts'].astype('datetime64[M]')
visits['month'] = visits['month'].dt.strftime('%Y-%m')

visits_source_month = visits.groupby(['month', 'source_id']).agg({'uid':'nunique'
```

In [43]:
```
visits.head()
```

Out[43]:

	device	end_ts	source_id	start_ts	uid	ssn_date	ssn_week	ssn_month
0	touch	2017-12-20 17:38:00	4	2017-12-20 17:20:00	16879256277535980062	2017-12-20	51	12
1	desktop	2018-02-19 17:21:00	2	2018-02-19 16:53:00	104060357244891740	2018-02-19	8	2
2	touch	2017-07-01 01:54:00	5	2017-07-01 01:54:00	7459035603376831527	2017-07-01	26	7
3	desktop	2018-05-20 11:23:00	9	2018-05-20 10:59:00	16174680259334210214	2018-05-20	20	5
4	desktop	2017-12-27 14:06:00	3	2017-12-27 14:06:00	9969694820036681168	2017-12-27	52	12

In [44]:
```
orders.head()
```

Out[44]:

	buy_ts	revenue	uid	first_activity_date	activity_month	first_activity_month
0	2017-06-01 00:10:00	17.00	10329302124590727494	2017-06-01 00:10:00	2017-06-01	2017-06-01
1	2017-06-01 00:25:00	0.55	11627257723692907447	2017-06-01 00:25:00	2017-06-01	2017-06-01
2	2017-06-01 00:27:00	0.37	17903680561304213844	2017-06-01 00:27:00	2017-06-01	2017-06-01
3	2017-06-01 00:29:00	0.55	16109239769442553005	2017-06-01 00:29:00	2017-06-01	2017-06-01
4	2017-06-01 07:58:00	0.37	14200605875248379450	2017-06-01 07:58:00	2017-06-01	2017-06-01

In [45]:
```
n_buyers_all = orders.groupby('activity_month').agg({'uid':'nunique'}).sum()
display(n_buyers_all)
```

```
uid    41019
dtype: int64
```

In [46]:
```
cac_all = costs_all / n_buyers_all
display(cac_all)
```

```
uid    8.023882
dtype: float64
```

In [47]:
```
monthly_costs = costs.groupby(['month', 'source_id'])['costs'].sum().reset_index(
monthly_costs.head()
```

Out[47]:

	month	source_id	costs
0	2017-06	1	1125.61
1	2017-06	2	2427.38
2	2017-06	3	7731.65
3	2017-06	4	3514.80
4	2017-06	5	2616.12

In [48]:
```
visits_source_month.head()
```

Out[48]:

	month	source_id	uid
0	2017-06	1	972
1	2017-06	2	1532
2	2017-06	3	4226
3	2017-06	4	3636
4	2017-06	5	2903

In [49]:
```
monthly_costs = monthly_costs.merge(
    visits_source_month, on=['month', 'source_id']
)
```

The CAC for the entire project is equal to 1.4 USD per attracted user.

In [50]:
```
report_new = pd.merge(
    monthly_costs, report, left_on=['month', 'source_id'], right_on=['first_activi
)

report_new['cac'] = report_new['costs'] / report_new['n_buyers']
display(report_new)
```

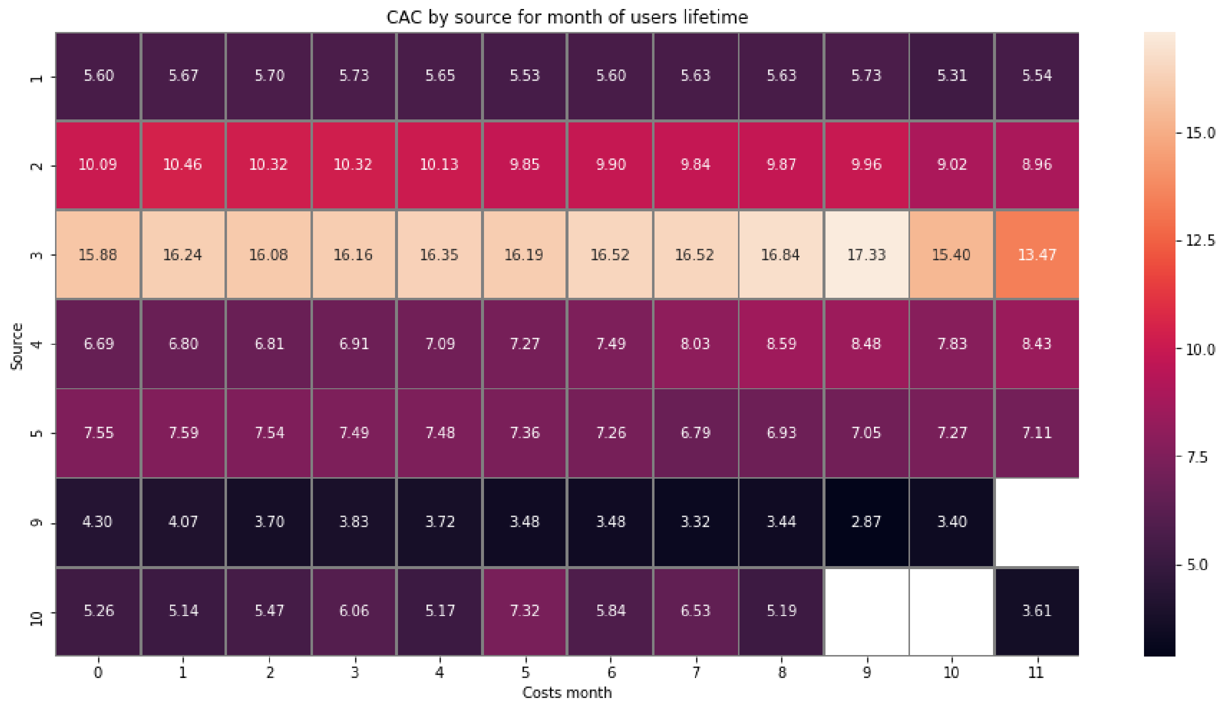
	month	source_id	costs	uid	first_activity_month	activity_month	revenue	n_buyers	
0	2017-06	1	1125.61	972	2017-06	2017-06-01	1168.45	203	116
1	2017-06	1	1125.61	972	2017-06	2017-07-01	362.94	203	36
2	2017-06	1	1125.61	972	2017-06	2017-08-01	153.72	203	15
3	2017-06	1	1125.61	972	2017-06	2017-09-01	695.88	203	69
4	2017-06	1	1125.61	972	2017-06	2017-10-01	760.74	203	76
...	...	...	...	...	...	...	...	...	
508	2018-05	3	9411.42	5343	2018-05	2018-05-01	2990.65	785	299
509	2018-05	4	4214.21	7275	2018-05	2018-05-01	3616.23	767	361
510	2018-05	5	3669.56	4038	2018-05	2018-05-01	2103.24	510	210
511	2018-05	9	362.17	753	2018-05	2018-05-01	200.38	53	20
512	2018-05	10	409.86	777	2018-05	2018-05-01	478.93	117	47

513 rows × 12 columns

In [51]:
```
source_cac = report_new.pivot_table(
    index='source_id',
    columns='age',
    values='cac',
    aggfunc='mean'
)

display(source_cac)
```

age source_id	0	1	2	3	4	5	6	7
1	5.600034	5.674538	5.700159	5.729075	5.646485	5.526385	5.595484	5.632652
2	10.087368	10.459870	10.317881	10.320378	10.134720	9.850867	9.895222	9.840281
3	15.884857	16.239020	16.078505	16.163100	16.353683	16.192237	16.523822	16.519482
4	6.687370	6.795822	6.809469	6.913436	7.091741	7.271592	7.490445	8.030521
5	7.553257	7.585806	7.541371	7.485356	7.477701	7.359916	7.262140	6.789045
9	4.301936	4.071803	3.700879	3.834142	3.719121	3.478545	3.479976	3.317200
10	5.259864	5.143985	5.472223	6.060143	5.170574	7.320679	5.839062	6.530117

In [52]:
```
plt.figure(figsize=(16,8))
sns.heatmap(source_cac, fmt='.2f', annot=True, linewidth=0.7, linecolor='grey')
plt.title('CAC by source for month of users lifetime')
plt.ylabel('Source')
plt.xlabel('Costs month')
plt.show()
```



The cost of attracting a single user for each of the sources increases by October and from November to January sharply decreases increasing again from February. Source number 3 is the most expensive to attract a single user.

Let's calculate the ROMI (return on marketing investment) for cohorts in the context of sources. So compare the payback for the same life periods of the cohorts.

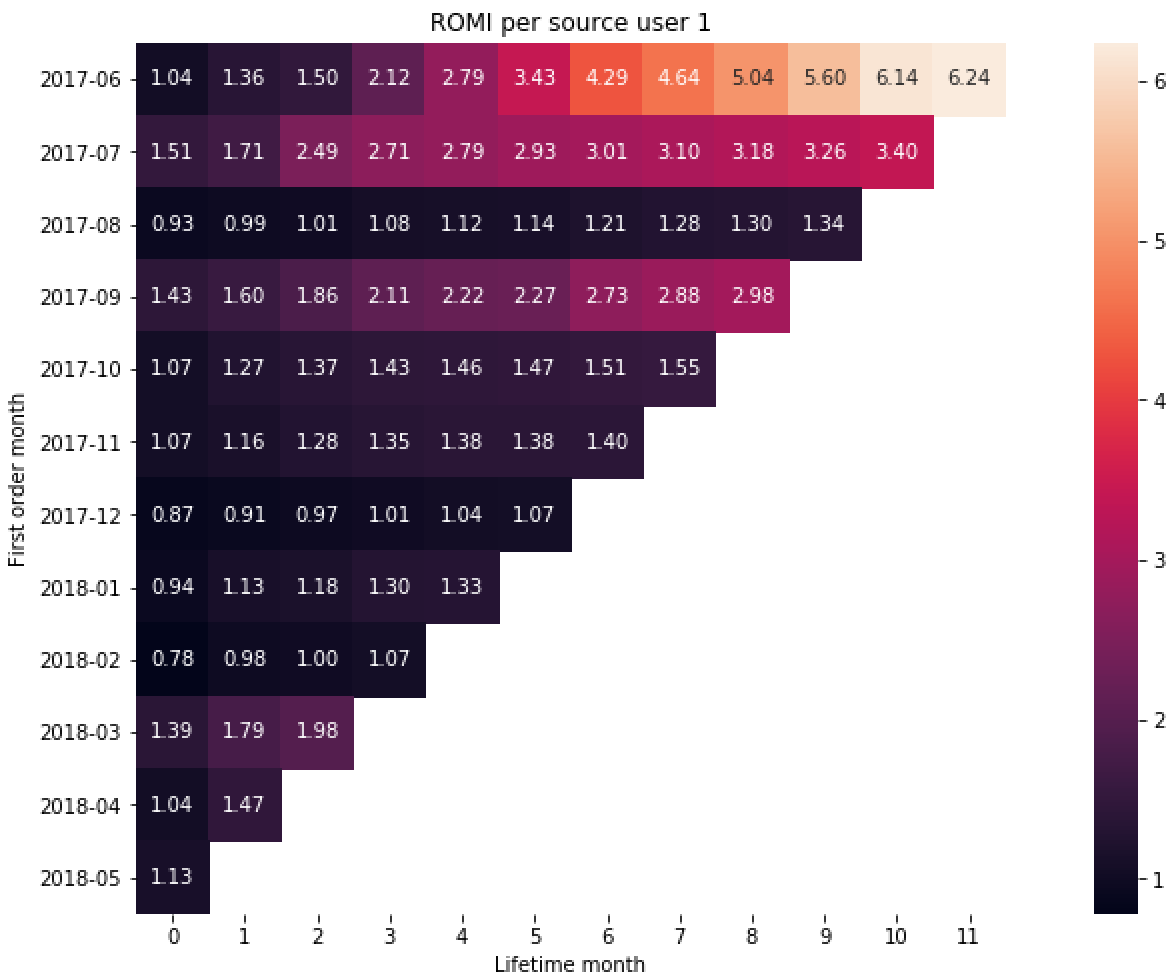
In [53]:
```
report_new['romi'] = report_new['ltv'] / report_new['cac']
report_new.head()
```

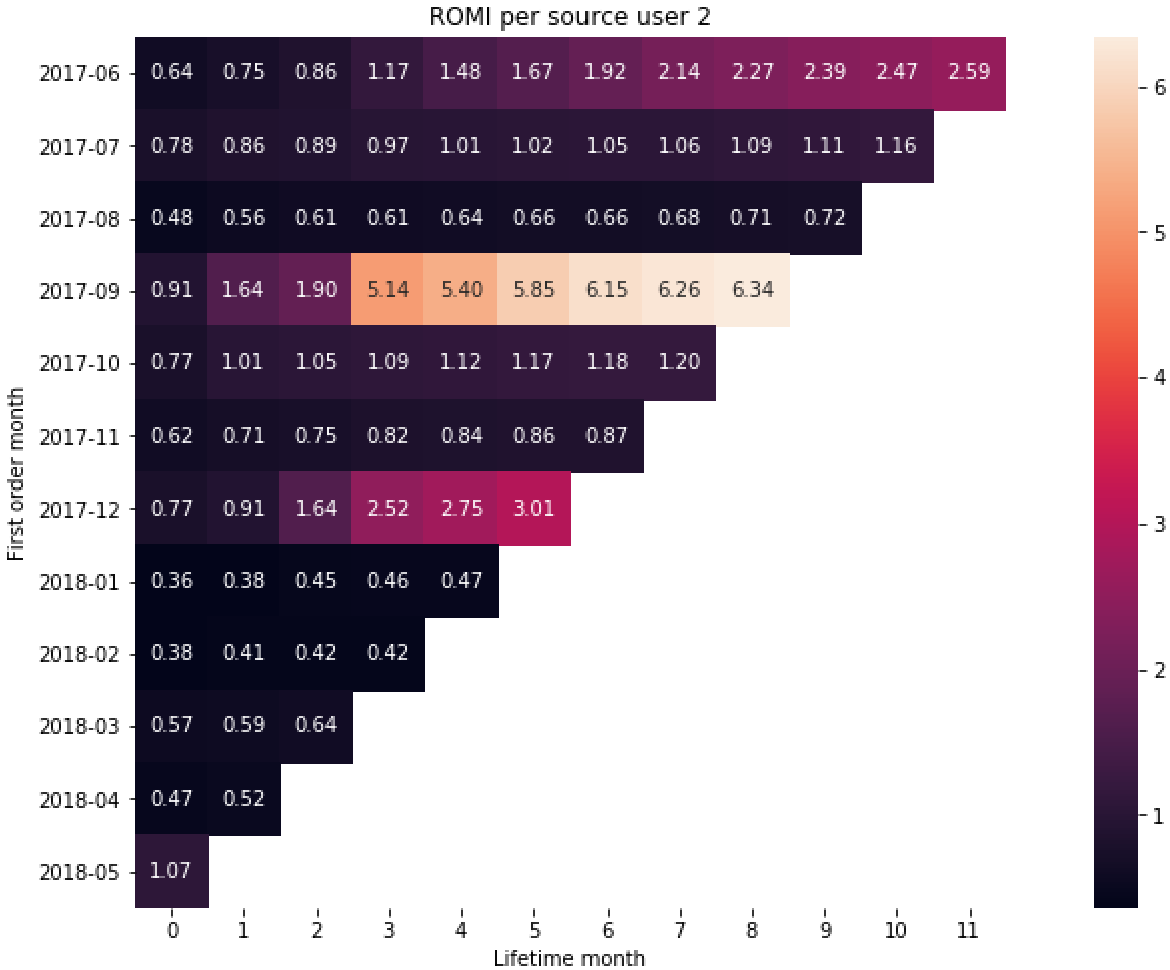
Out[53]:

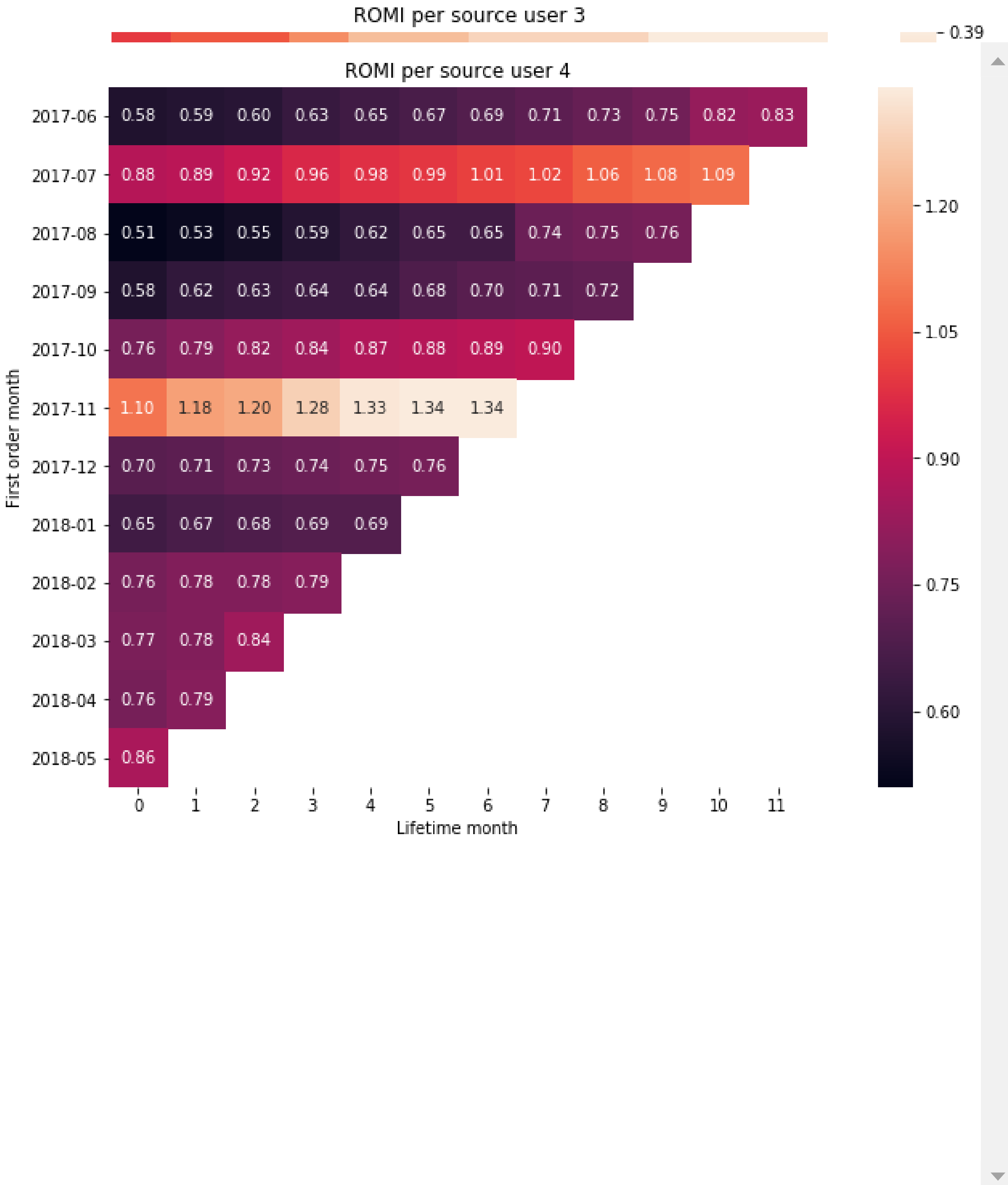
	month	source_id	costs	uid	first_activity_month	activity_month	revenue	n_buyers	gp
0	2017-06	1	1125.61	972	2017-06	2017-06-01	1168.45	203	1168.45
1	2017-06	1	1125.61	972	2017-06	2017-07-01	362.94	203	362.94
2	2017-06	1	1125.61	972	2017-06	2017-08-01	153.72	203	153.72
3	2017-06	1	1125.61	972	2017-06	2017-09-01	695.88	203	695.88
4	2017-06	1	1125.61	972	2017-06	2017-10-01	760.74	203	760.74

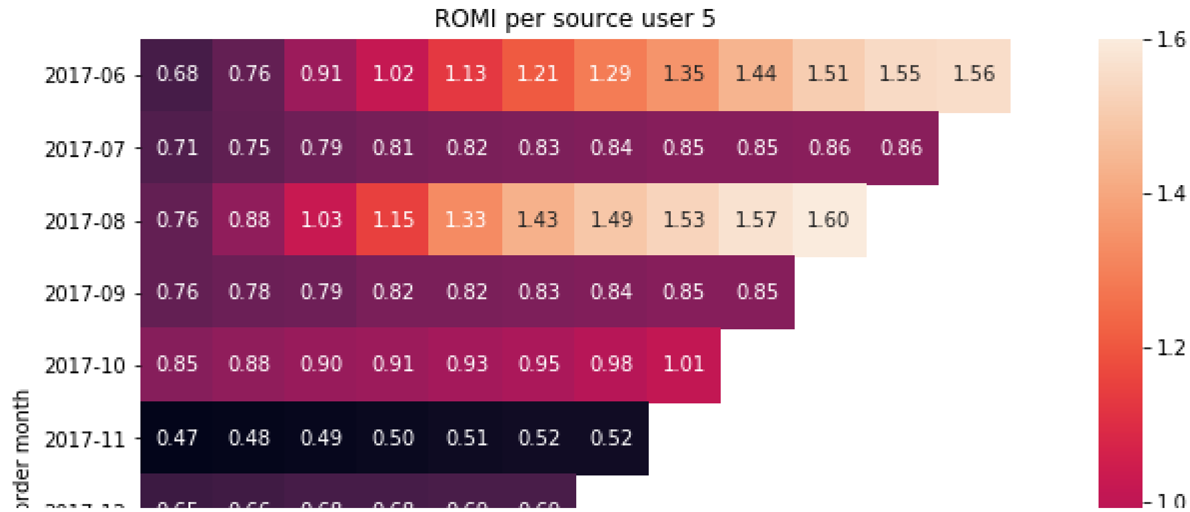
In [54]:
```
source_list = [ 1,  2,  3,  4,  5,  9, 10]
```

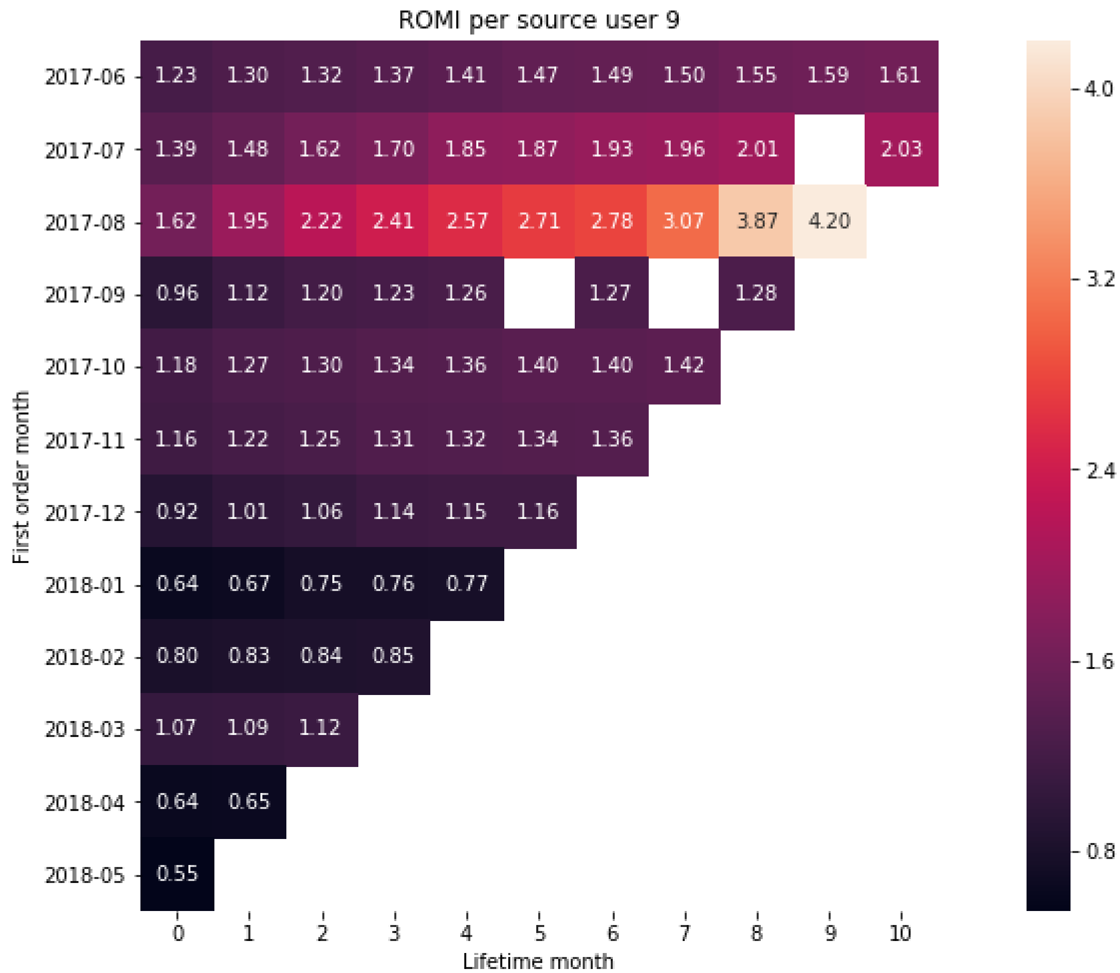
In [55]:
```
for i in source_list:
    plt.figure(figsize=(16,8))
    romi_pivot = report_new[report_new['source_id'] == i].pivot_table(
        index='first_activity_month',
        columns='age',
        values='romi',
        aggfunc='mean').cumsum(axis=1).round(2)
    sns.heatmap(romi_pivot, fmt='.2f', annot=True, linecolor='grey', square=True)
    plt.title('ROMI per source user {}' .format(i))
    plt.ylabel('First order month')
    plt.xlabel('Lifetime month')
    plt.show()
```

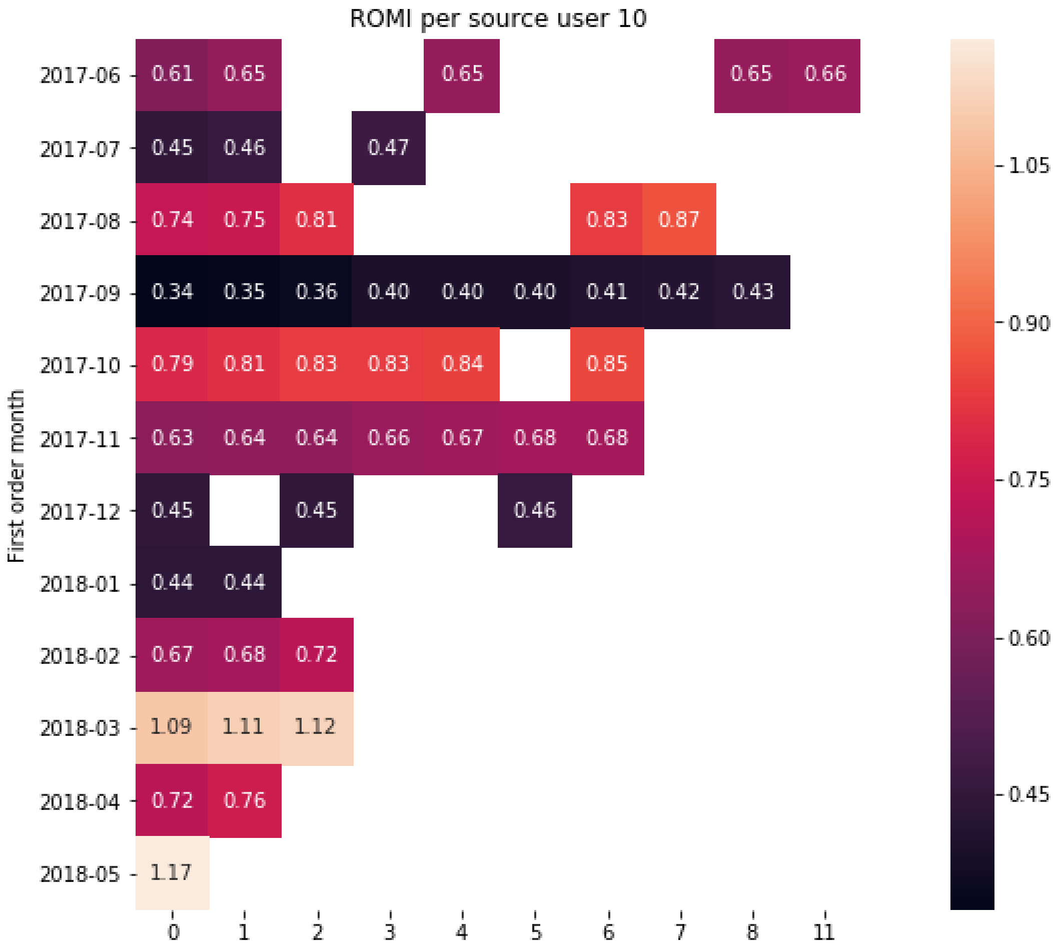



ROMI per source user 2
First order month
Lifetime month
2017-06
2017-07
2017-08
2017-09
2017-10
2017-11
2017-12
2018-01
2018-02
2018-03
2018-04
2018-05
0 1 2 3 4 5 6 7 8 9 10 11
0.64 0.75 0.86 1.17 1.48 1.67 1.92 2.14 2.27 2.39 2.47 2.59
0.78 0.86 0.89 0.97 1.01 1.02 1.05 1.06 1.09 1.11 1.16
0.48 0.56 0.61 0.61 0.64 0.66 0.66 0.68 0.71 0.72
0.91 1.64 1.90 5.14 5.40 5.85 6.15 6.26 6.34
0.77 1.01 1.05 1.09 1.12 1.17 1.18 1.20
0.62 0.71 0.75 0.82 0.84 0.86 0.87
0.77 0.91 1.64 2.52 2.75 3.01
0.36 0.38 0.45 0.46 0.47
0.38 0.41 0.42 0.42
0.57 0.59 0.64
0.47 0.52
1.07
1 2 3 4 5 6


ROMI per source user 3
0.39
ROMI per source user 4
2017-06 0.58 0.59 0.60 0.63 0.65 0.67 0.69 0.71 0.73 0.75 0.82 0.83
2017-07 0.88 0.89 0.92 0.96 0.98 0.99 1.01 1.02 1.06 1.08 1.09
2017-08 0.51 0.53 0.55 0.59 0.62 0.65 0.65 0.74 0.75 0.76
2017-09 0.58 0.62 0.63 0.64 0.64 0.68 0.70 0.71 0.72
2017-10 0.76 0.79 0.82 0.84 0.87 0.88 0.89 0.90
2017-11 1.10 1.18 1.20 1.28 1.33 1.34 1.34
2017-12 0.70 0.71 0.73 0.74 0.75 0.76
2018-01 0.65 0.67 0.68 0.69 0.69
2018-02 0.76 0.78 0.78 0.79
2018-03 0.77 0.78 0.84
2018-04 0.76 0.79
2018-05 0.86
First order month
0 1 2 3 4 5 6 7 8 9 10 11
Lifetime month
1.20
1.05
0.90
0.75
0.60


ROMI per source user 5
2017-06 0.68 0.76 0.91 1.02 1.13 1.21 1.29 1.35 1.44 1.51 1.55 1.56
2017-07 0.71 0.75 0.79 0.81 0.82 0.83 0.84 0.85 0.85 0.86 0.86
2017-08 0.76 0.88 1.03 1.15 1.33 1.43 1.49 1.53 1.57 1.60
2017-09 0.76 0.78 0.79 0.82 0.82 0.83 0.84 0.85 0.85
2017-10 0.85 0.88 0.90 0.91 0.93 0.95 0.98 1.01
2017-11 0.47 0.48 0.49 0.50 0.51 0.52 0.52
1.6
1.4
1.2
1.0


ROMI per source user 9
2017-06 1.23 1.30 1.32 1.37 1.41 1.47 1.49 1.50 1.55 1.59 1.61
2017-07 1.39 1.48 1.62 1.70 1.85 1.87 1.93 1.96 2.01 2.03
2017-08 1.62 1.95 2.22 2.41 2.57 2.71 2.78 3.07 3.87 4.20
2017-09 0.96 1.12 1.20 1.23 1.26 1.27 1.28
2017-10 1.18 1.27 1.30 1.34 1.36 1.40 1.40 1.42
2017-11 1.16 1.22 1.25 1.31 1.32 1.34 1.36
2017-12 0.92 1.01 1.06 1.14 1.15 1.16
2018-01 0.64 0.67 0.75 0.76 0.77
2018-02 0.80 0.83 0.84 0.85
2018-03 1.07 1.09 1.12
2018-04 0.64 0.65
2018-05 0.55
0 1 2 3 4 5 6 7 8 9 10
Lifetime month
First order month
4.0
3.2
2.4
1.6
0.8



The values are missing because they are too low to display in the 3-digit format.

Despite the huge spending on sources number 3, 4 and 5, they ROMI for them is the lowest among all the others. First of all this means that the effectiveness of these sources is extremely low and large cash injections into these sources are impractical. Based on the data on the chart the sources with the lowest cash infusions - 1, 9 and 10-were the most profitable in terms of the return of investments. Moreover sources 9 and 10 show simply cosmic results.

Conclusions and recomendations

The main sources to focus on are sources number 1 (CAC - 0.59 CU), 9 (CAC-0.16 CU) and 10 (CAC-0.16 CU). These sources are the most effective based on the cost per user and return on investment. The ROMI of sources 9 and 10 is between 2700 % and 9000 %.

Despite the fact that each of the sources pays off, it is worth focusing on the three sources described above. They use resources most efficiently.

The conclusions description of each type metrics: marketing, product,

and e-commerce metrics:

Product's metrics:

Apparently the number of unique users as well as the number of visits to Yandex.Afisha,increases from August to the end of the year and then decreases by the summer period.

Active peaks of visits and orders are August, September, November, and May.

I think that in August user activity may increase due to the end of the holiday season and by the end of the year and in May due to the approaching holidays.

If you do not take into account the low percentage of repeat orders in general the Retention Rate is best preserved for users who made their first purchases from May to August.

E-commerce metrics:

On average it takes 32 minutes from the start of the session to the purchase.

On average users make 1-2 orders per month. We have taken into account the period from May to October 2017. Despite the increase in orders by September the number of orders per user is almost unchanged which indicates a weak customer retention after the first purchase.

The average monthly check does not change significantly except in December. Before the new year the average check increases dramatically.

The average LTV for 6 months is 8.3 USD per user.

Marketing metrics:

Source number 3 is the most expensive, and source number 9 is the least expensive. The top three in terms of costs are the 3rd, 4th and 5th.

The total amount spent on marketing is 329 thousand USD.

Periods of marketing activity - In each source, costs gradually decrease by the end of August and increase sharply from September to December. After the new year, costs begin to gradually decrease.

The CAC for the entire project is 1.4 USD per attracted user, which is a good indicator when you consider that each user brings 8.3 USD in 6 months.

The cost of attracting a single user for each of the sources increases by October and from November to January sharply decreases, increasing again from February. Source number 3 is the most expensive to attract a single user.

Summarising the results of the cohort analysis:

The most promising cohorts are the cohorts of May, August, and September 2017, as their activity is maintained best.