АНАЛИЗ НОВОЙ ФУНКЦИОНАЛЬНОСТИ ИГРЫ.СРАВНЕНЕЕ ПОКАЗАТЕЛЕЙ ИГРЫ ДО И ПОСЛЕ ВВЕДЕНИЯ НОВОЙ ФУНКЦИОНАЛЬНОСТИ.

Задание:

В игру введена новуая функциональность. Необходимо проанализировать ее и определить полезна ли она игре, сравнить поведение игры после введения новой функциональности. Определить есть ли рост основных игровых метрик (retention,ARPU), и посмотреть удерживает игроков новый функционал лучше.

In [1]:
```
%matplotlib inline
import pandas as pd
import numpy as np
```

Шаг 1. Получение, предобработка данных об активных и зарегистрированных пользователях игры

In [2]:
```
registered_users_count=pd.read_csv("registered_users.csv")
registered_users_count.head()
```

Out[2]:

	Unnamed: 0	registration_date	registered_users_count
0	0	2019-06-01	4833
1	1	2019-06-02	5255
2	2	2019-06-03	4193
3	3	2019-06-04	4194
4	4	2019-06-05	3998

In [3]:
```
#исключим первый лишний столбец
registered_users_count=registered_users_count.iloc[:, 1:4]
```

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

Out[4]:

	registration_date	registered_users_count
0	2019-06-01	4833
1	2019-06-02	5255
2	2019-06-03	4193
3	2019-06-04	4194
4	2019-06-05	3998

In [5]:
```
active_users_count_with_cohorts=pd.read_csv("active_users.csv")
active_users_count_with_cohorts.head()
```

Out[5]:

	Unnamed: 0	activity_date	registration_date	active_users_count
0	0	2019-06-01	2019-06-01	1651
1	1	2019-06-02	2019-06-01	1429
2	2	2019-06-02	2019-06-02	1933
3	3	2019-06-03	2019-06-01	1021
4	4	2019-06-03	2019-06-02	1550

In [6]:
```
active_users_count_with_cohorts=active_users_count_with_cohorts.iloc[:, 1:4]
active_users_count_with_cohorts.head()
```

Out[6]:

	activity_date	registration_date	active_users_count
0	2019-06-01	2019-06-01	1651
1	2019-06-02	2019-06-01	1429
2	2019-06-02	2019-06-02	1933
3	2019-06-03	2019-06-01	1021
4	2019-06-03	2019-06-02	1550

Посмотрим на информацию о датафрейме `registered_users`:

In [7]:
```
registered_users_count.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 2 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   registration_date       30 non-null     object
 1   registered_users_count  30 non-null     int64
dtypes: int64(1), object(1)
memory usage: 608.0+ bytes
```

In [8]:
```
registered_users_count.head(4)
```

Out[8]:

	registration_date	registered_users_count
0	2019-06-01	4833
1	2019-06-02	5255
2	2019-06-03	4193
3	2019-06-04	4194

Поменяем тип колонки с датой - `object`, на `datetime`:

In [9]:
```
registered_users_count['registration_date'] = pd.to_datetime(registered_users_count['registr
ation_date'])
```

Посмотрим на информацию о датафрейме `active_users_count_with_cohorts`:

In [10]:
```
active_users_count_with_cohorts.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 930 entries, 0 to 929
Data columns (total 3 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   activity_date       930 non-null    object
 1   registration_date   930 non-null    object
 2   active_users_count  930 non-null    int64
dtypes: int64(1), object(2)
memory usage: 21.9+ KB
```

Типы колонок с датой - `object`, поменяем их на `datetime`:

In [11]:
```
for col in ['activity_date','registration_date']:
    active_users_count_with_cohorts[col] = pd.to_datetime(active_users_count_with_cohorts[co
l])
```

1. Расчет Retention Rate Объединим таблицы два датафрейма: `active_users_count_with_cohorts` и `registered_users_count` по колонке `registration_date` и запишем объединенные данные в датафрейм `retention_table`.

Посчитаем `retention_rate` - отношение числа активных пользователей `active_users_count` на конкретную неделю к общему числу пользователей в когорте `registered_users_count`.

In [12]:
```
retention_table = active_users_count_with_cohorts.merge(registered_users_count,on=['registra
tion_date'],how='left')

retention_table['retention_rate'] = retention_table['active_users_count'] / retention_table[
'registered_users_count']
retention_table.head()
```

Out[12]:

	activity_date	registration_date	active_users_count	registered_users_count	retention_rate
0	2019-06-01	2019-06-01	1651	4833	0.341610
1	2019-06-02	2019-06-01	1429	4833	0.295676
2	2019-06-02	2019-06-02	1933	5255	0.367840
3	2019-06-03	2019-06-01	1021	4833	0.211256
4	2019-06-03	2019-06-02	1550	5255	0.294957

Посчитаем показатель `lifetime` -разницу между датой активности `activity_date` и датой регистрации `registration_date`:

In [13]:
```
retention_table['lifetime'] = retention_table['activity_date'] - retention_table['registrati
on_date']
retention_table['lifetime'] = retention_table['lifetime']/np.timedelta64(1,'D')
retention_table['lifetime'] = retention_table['lifetime'].round().astype(int)
retention_pivot = retention_table.pivot_table(index='registration_date',columns='lifetime',v
alues='retention_rate',aggfunc='sum')
```

Посчитаем усредненный Retention Rate по различным lifetime:

In [14]:
```
retention_mean_by_lifetime = retention_pivot.mean()
print (retention_mean_by_lifetime)
```

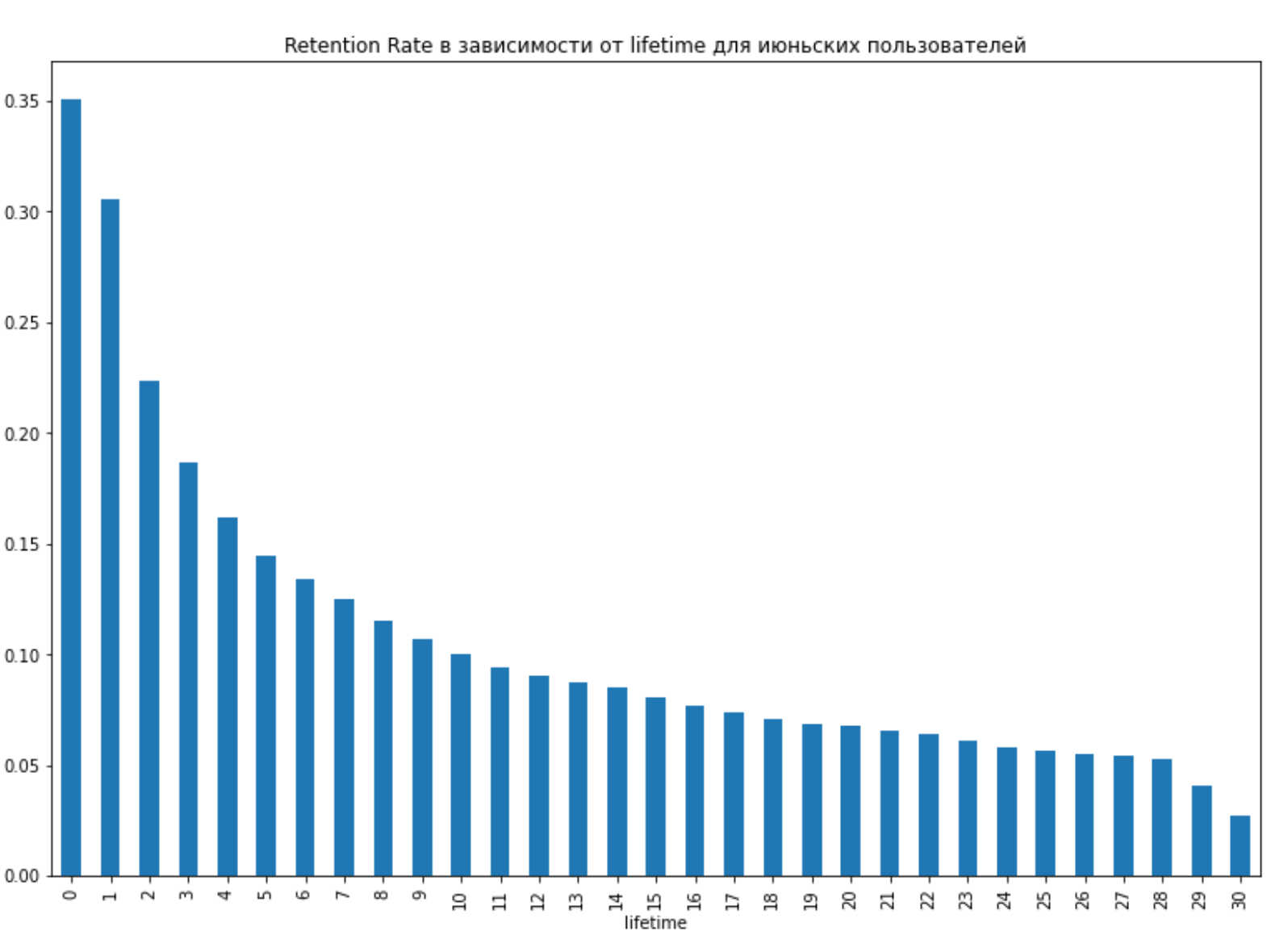
```
lifetime
0     0.350607
1     0.305707
2     0.223304
3     0.186753
4     0.161584
5     0.144832
6     0.134225
7     0.125151
8     0.115505
9     0.106549
10    0.100147
11    0.094230
12    0.090701
13    0.087670
14    0.085370
15    0.080326
16    0.076764
17    0.073749
18    0.070988
19    0.068546
20    0.067471
21    0.065852
22    0.063643
23    0.060973
24    0.057966
25    0.056677
26    0.054686
27    0.053935
28    0.052781
29    0.040779
30    0.027130
dtype: float64
```

Шаг 2. Анализируем Retention Rate

Построим график изменения Retention Rate в зависимости от lifetime:

In [15]:
```
from matplotlib import pyplot as plt
plt.figure(figsize=(13, 9))
plt.title('Retention Rate в зависимости от lifetime для июньских пользователей')
retention_mean_by_lifetime.plot.bar()
```

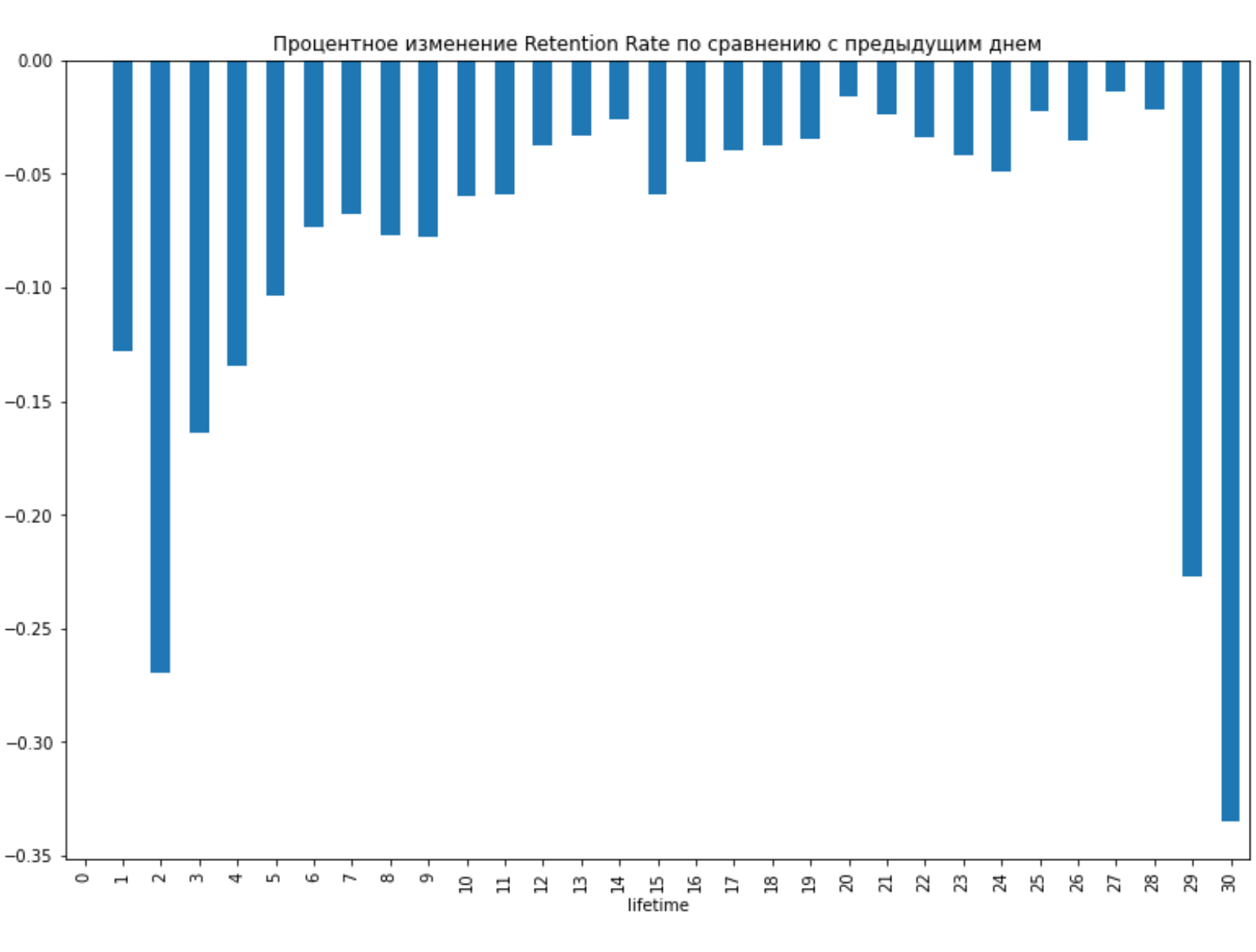
Out[15]:
```
<AxesSubplot:title={'center':'Retention Rate в зависимости от lifetime для июньских пользоват
елей'}, xlabel='lifetime'>
```



Как видно, с течением времени Retention Rate убывает, при этом наблюдается существенное процентное падение на 2, 29 и 30 день. Это будет особенно заметно, если построить график процентного изменения Retention Rate относительно предыдущего дня:

In [16]:
```
plt.figure(figsize=(13, 9))
plt.title('Процентное изменение Retention Rate по сравнению с предыдущим днем')
retention_mean_by_lifetime.pct_change().plot.bar()
```

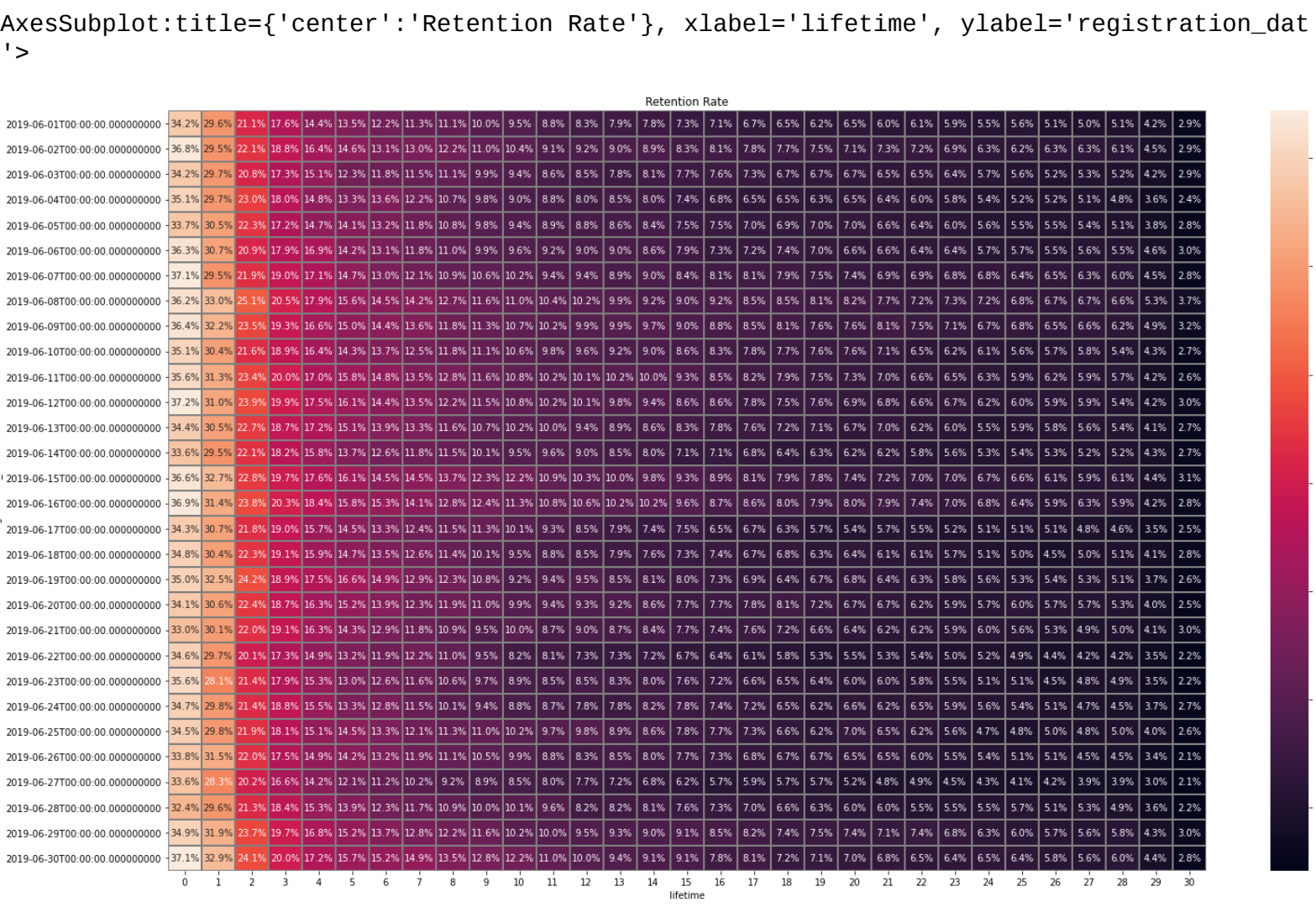
Out[16]:
```
<AxesSubplot:title={'center':'Процентное изменение Retention Rate по сравнению с предыдущим д
нем'}, xlabel='lifetime'>
```



Построим тепловую карту изменения Retention Rate:

In [17]:
```
import seaborn as sns
from matplotlib import pyplot as plt
plt.figure(figsize=(25, 15))
plt.title('Retention Rate')
sns.heatmap(retention_pivot, annot=True, fmt='.1%', linewidths=1, linecolor='gray')
```

Out[17]:
```
<AxesSubplot:title={'center':'Retention Rate'}, xlabel='lifetime', ylabel='registration_dat
e'>
```



Шаг 3. Получение,предобработка данных о выручке

In [18]:
```
revenue=pd.read_csv("revenue_users.csv")
revenue=revenue.iloc[:,1:5]
revenue.head()
```

Out[18]:

	date	registration_date	users_count_with_revenue	revenue
0	2019-06-01	2019-06-01	1082	1611.860
1	2019-06-02	2019-06-01	928	317.738
2	2019-06-02	2019-06-02	1257	1305.220
3	2019-06-03	2019-06-01	623	192.273
4	2019-06-03	2019-06-02	1008	554.859

Посмотрим на информацию о датафрейме `revenue`:

In [19]:
```
revenue.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 833 entries, 0 to 832
Data columns (total 4 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   date                      833 non-null    object
 1   registration_date         833 non-null    object
 2   users_count_with_revenue  833 non-null    int64
 3   revenue                   833 non-null    float64
dtypes: float64(1), int64(1), object(2)
memory usage: 26.2+ KB
```

Типы колонок с датой - `object`, поменяем их на `datetime`:

In [20]:
```
for col in ['date','registration_date']:
    revenue[col] = pd.to_datetime(revenue[col])
```

Шаг 4. Подсчет выручки по когортам

Объединим данные датафрейма `revenue` с данными датафрейма `retention_table`. Но прежде нам нужно измненить название колонки `date` на `activity_date` в датафрейме `revenue`. Это нужно для того, фчтобы в дальнейшем объединять датафреймы по одинаковому названию колонок.

In [21]:
```
revenue = revenue.rename(columns={'date':'activity_date'})
```

После переименования, можно объединить датафреймы в новый датафрейм `retention_table_with_revenue`:

In [22]:
```
retention_table_with_revenue = retention_table.merge(revenue,on=['registration_date','activi
ty_date'],how='left')
```

Посмотрим на получившийся объединенный датафрейм:

In [23]:
```
retention_table_with_revenue.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 930 entries, 0 to 929
Data columns (total 8 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   activity_date             930 non-null    datetime64[ns]
 1   registration_date         930 non-null    datetime64[ns]
 2   active_users_count        930 non-null    int64
 3   registered_users_count    930 non-null    int64
 4   retention_rate            930 non-null    float64
 5   lifetime                  930 non-null    int32
 6   users_count_with_revenue  833 non-null    float64
 7   revenue                   833 non-null    float64
dtypes: datetime64[ns](2), float64(3), int32(1), int64(2)
memory usage: 61.8 KB
```

Видим, что после объединения у нас получились пропущенные значения. Это говорит о том, что не во все дни была выручка с пользователей.

Произведем замену пропущенных значений на 0 с помощью функции `fillna()`:

In [24]:
```
for col in ['revenue','users_count_with_revenue']:
    retention_table_with_revenue[col] = retention_table_with_revenue[col].fillna(0)
```

Преобразуем тип в колонке `users_count_with_revenue` в целочисленный:

In [25]:
```
retention_table_with_revenue['users_count_with_revenue'] = retention_table_with_revenue['use
rs_count_with_revenue'].astype(int)
```

Рассчитаем ARPU в колонку `arpu`:

In [26]:
```
retention_table_with_revenue['arpu'] = retention_table_with_revenue['revenue'] / retention_t
able_with_revenue['active_users_count']
```

Шаг 5. Анализируем ARPU

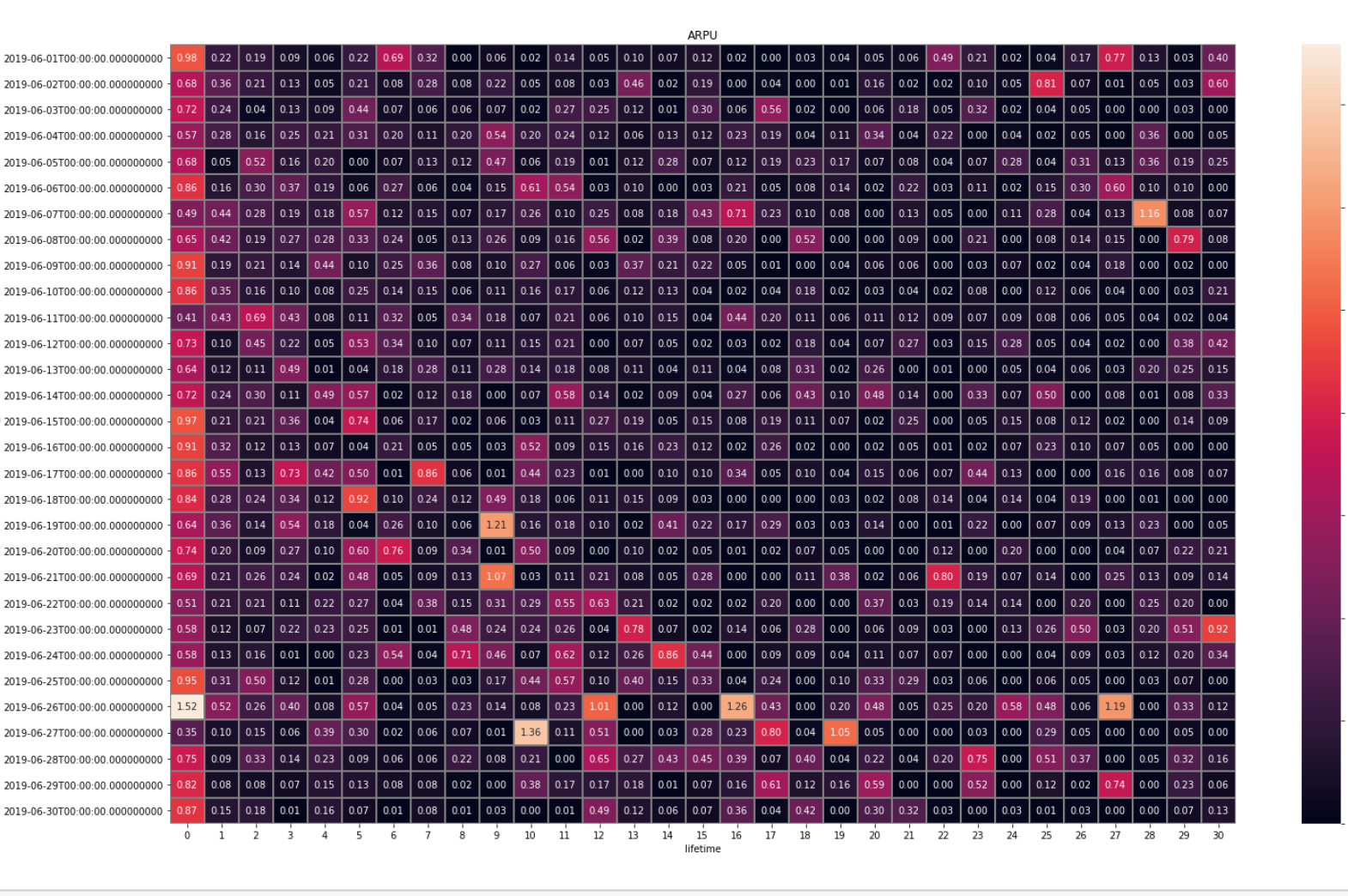
Создадим сводную таблицу `arpu_pivot`, где в строках будут даты регистрации пользователей, в колонках lifetime, а в показателе arpu:

In [27]:
```
arpu_pivot = retention_table_with_revenue.pivot_table(index='registration_date',columns='lif
etime',values='arpu',aggfunc='sum')
```

Построим тепловую карту по таблице `arpu_pivot`:

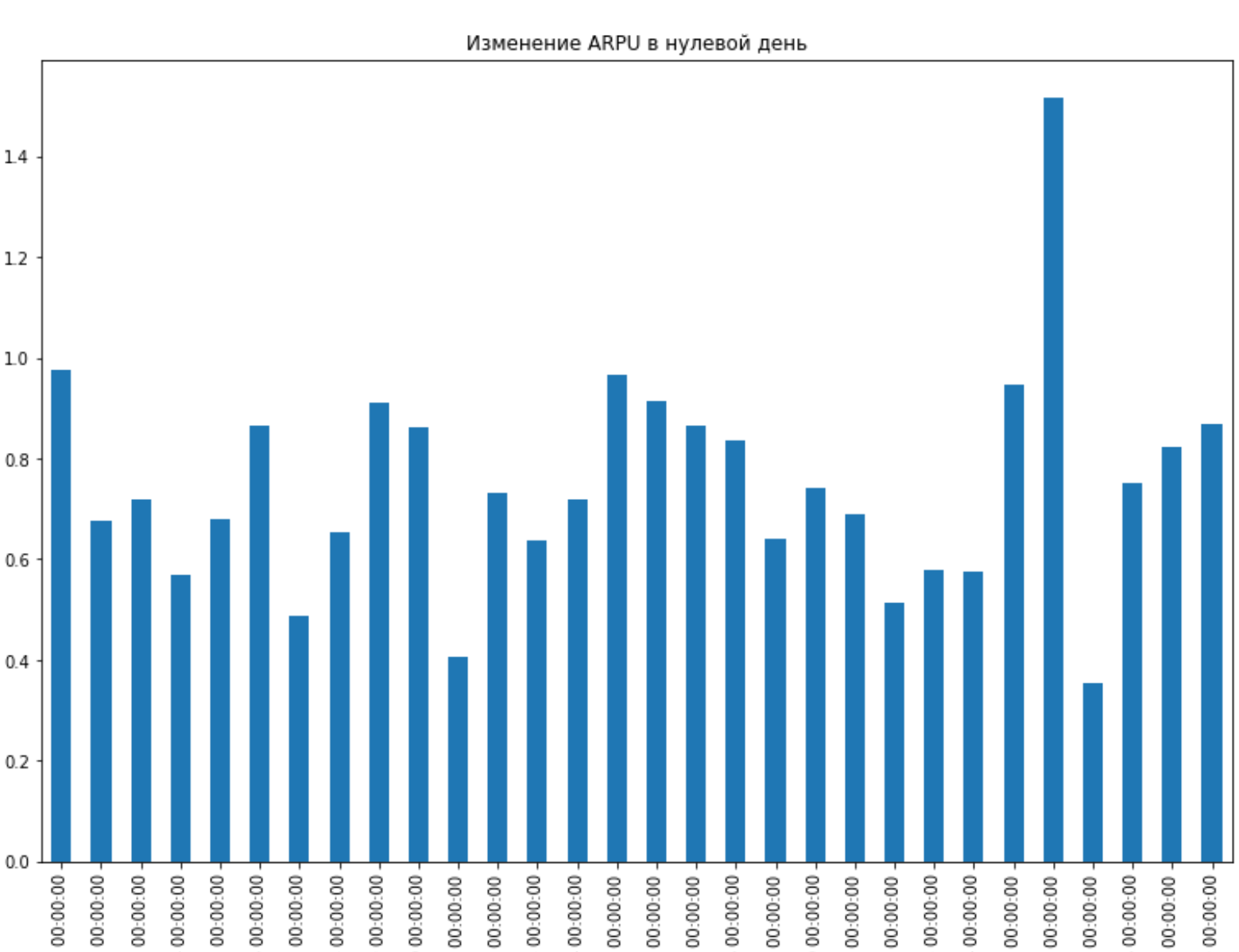
In [28]:
```
plt.figure(figsize=(25, 15))
plt.title('ARPU')
sns.heatmap(arpu_pivot, annot=True, fmt='.2f', linewidths=1, linecolor='gray')
```

Out[28]:
```
<AxesSubplot:title={'center':'ARPU'}, xlabel='lifetime', ylabel='registration_date'>
```



In [29]:
```
plt.figure(figsize=(13, 9))
plt.title('Изменение ARPU в нулевой день')
arpu_pivot[0].plot.bar()
```

Out[29]:
```
<AxesSubplot:title={'center':'Изменение ARPU в нулевой день'}, xlabel='registration_date'>
```



Как видно, сильнее всего выделяется ARPU в нулевой день в когорте 2019-06-26.

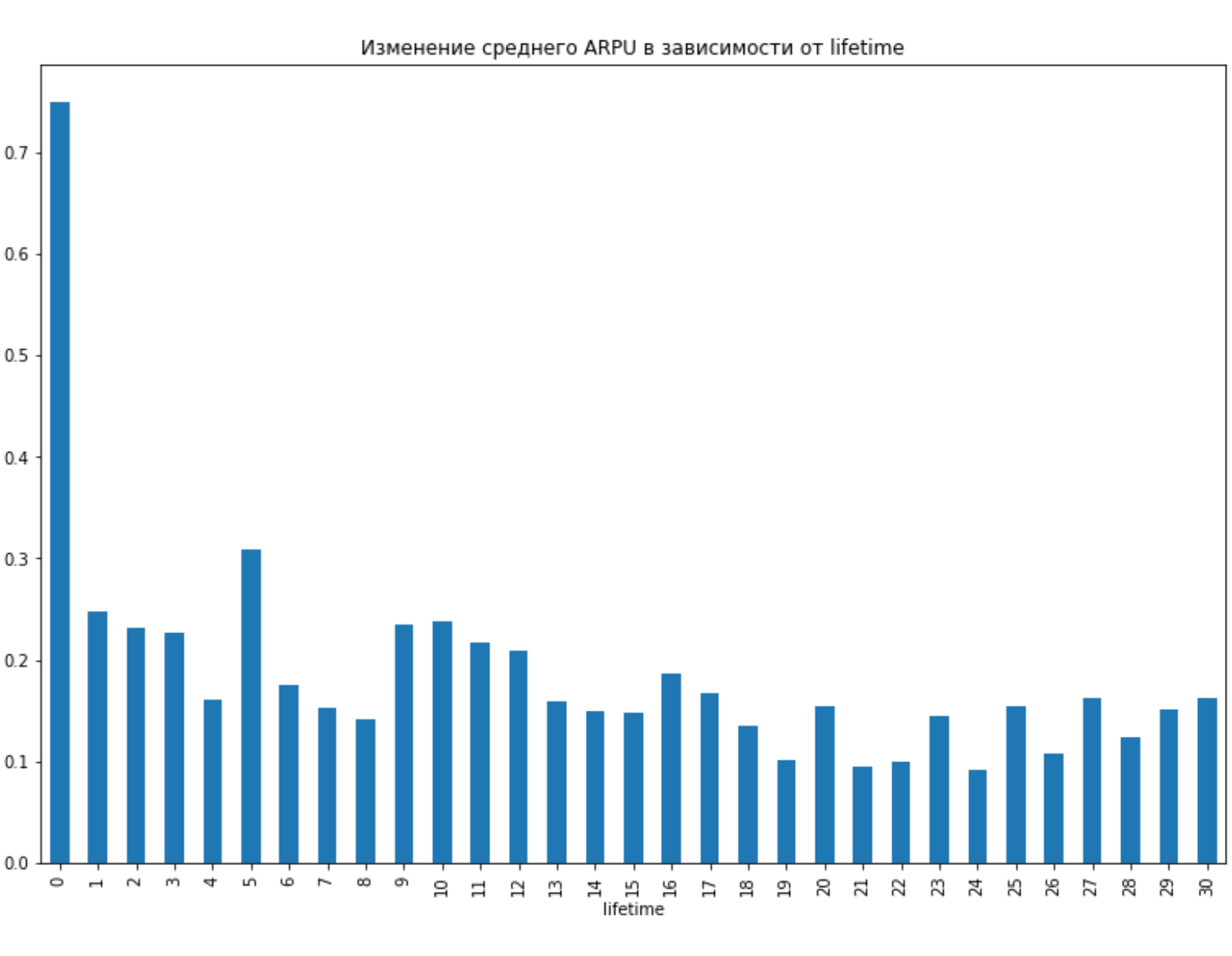
Посчитаем средний ARPU в зависимости от lifetime:

In [30]:
```
arpu_mean_by_lifetime = arpu_pivot.mean()
print(arpu_mean_by_lifetime)
```

```
lifetime
0     0.748997
1     0.247612
2     0.231219
3     0.226372
4     0.161454
5     0.308641
6     0.175224
7     0.153147
8     0.140943
9     0.234994
10    0.237923
11    0.216741
12    0.208700
13    0.158504
14    0.148857
15    0.148024
16    0.187117
17    0.167570
18    0.134404
19    0.100599
20    0.153664
21    0.094581
22    0.099685
23    0.144491
24    0.091224
25    0.153679
26    0.108511
27    0.161853
28    0.124374
29    0.150843
30    0.162709
dtype: float64
```

In [31]:
```
plt.figure(figsize=(13, 9))
plt.title('Изменение среднего ARPU в зависимости от lifetime')
arpu_mean_by_lifetime.plot.bar()
```

Out[31]:
```
<AxesSubplot:title={'center':'Изменение среднего ARPU в зависимости от lifetime'}, xlabel='li
fetime'>
```

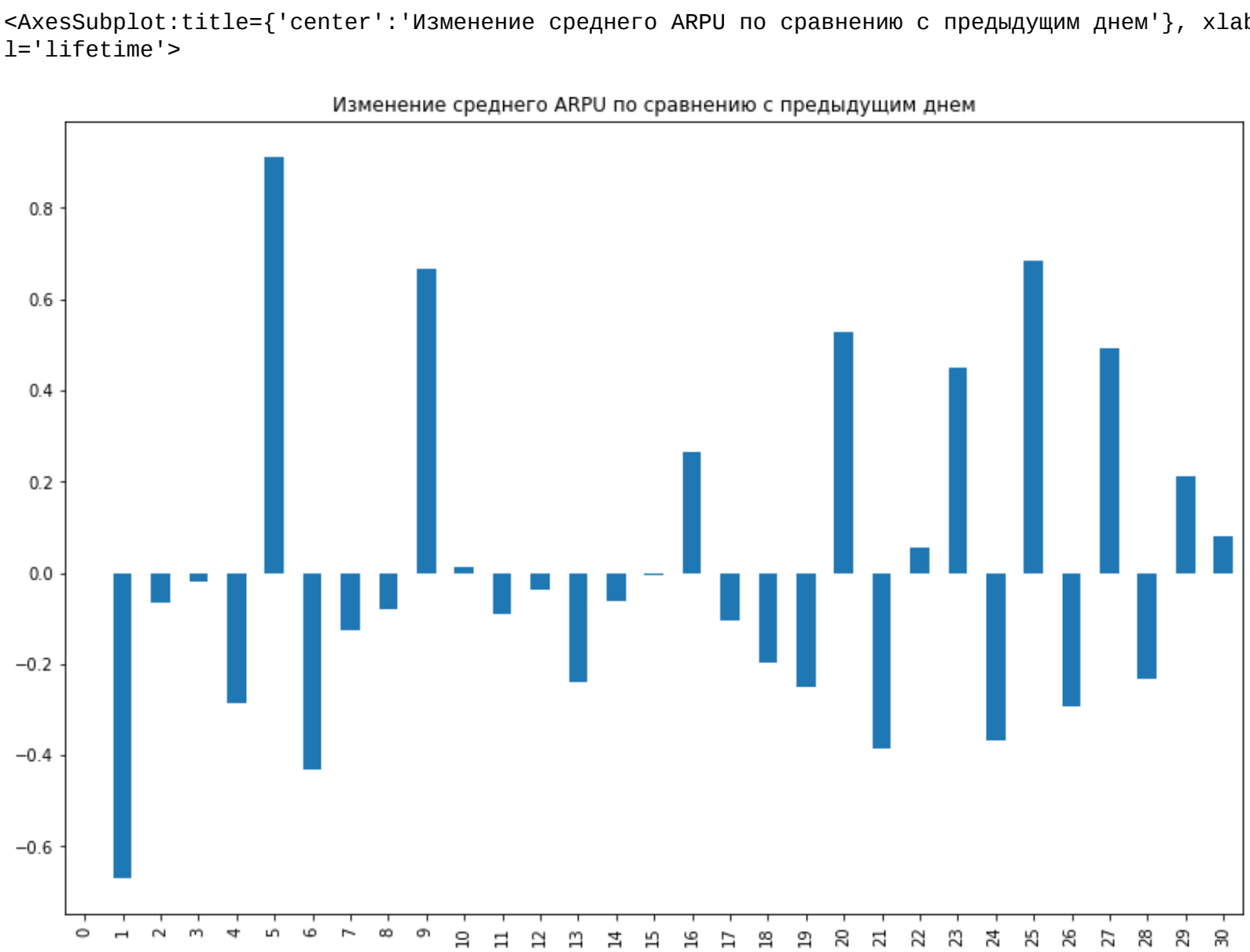


Из графика можно сделать вывод, что сильнее всего ARPU подниматся на 5 день lifetime, также начиная с 9 дня наблюдается рост.

Точнее о росте APRU -посмотрим на гистограмму изменения среднего ARPU относительно предыдущего дня:

In [32]:
```
plt.figure(figsize=(13, 9))
plt.title('Изменение среднего ARPU по сравнению с предыдущим днем')
arpu_mean_by_lifetime.pct_change().plot.bar()
```

Out[32]:
```
<AxesSubplot:title={'center':'Изменение среднего ARPU по сравнению с предыдущим днем'}, xlabe
l='lifetime'>
```



Как видно, после 16 дня наблюдается картина, где ARPU чередуется днями роста и спада