Metrics_AntonNew_HomeWork_5.ipynb

Файл Изменить Вид Вставка Среда выполнения Инструменты Справка

Комментировать Поделиться

+ Код + Текст

ОЗУ Диск

Редактирование

Подготовка

[39]
```
# imports
import numpy as np
import pandas as pd
import matplotlib
import plotly.express as px
from sqlalchemy import create_engine
from datetime import datetime, timedelta
from plotly.subplots import make_subplots
```

[40]
```
# connection
HOST = '37.139.42.145'
DBNAME = 'game-analytics'
USER = 'analytics'
PASSWORD = 'BRtTaqYiJyr29WXN'
TABLE_SCHEMA = 'data_viz_1068.project_dataset'
engine = create_engine(f'postgresql://{USER}:{PASSWORD}@{HOST}/{DBNAME}')
```

[41]
```
# getting table from database
project_dataset = pd.read_sql(f"""
    SELECT *
    FROM {TABLE_SCHEMA}
""", con=engine)

# converting some types to datetime
project_dataset['event_date'] = pd.to_datetime(project_dataset['event_date'])
project_dataset['cohort_date'] = pd.to_datetime(project_dataset['cohort_date'])

# dropping full duplicates (if they exists)
project_dataset.drop_duplicates()

# printing
project_dataset.head(2)
```

	event_time	event_date	event_name	revenue_usd	region	country	device_type	platform	cohort_date	user_id	user_type	content_id	currency	revenu
0	2020-12-14 21:16:20	2020-12-14	LaunchApp	0.0	EU	RU	xiaomi-Redmi Note 7	android	2020-09-29	120512	non_organic	None	None	0
1	2020-12-14 21:16:41	2020-12-14	LaunchApp	0.0	AS	KG	samsung-SM-G928F	android	2020-12-12	212364	organic	None	None	0

[42]
```
project_dataset = project_dataset.dropna(subset=['event_time'])
```

Дополнение основного датасета

[43]
```
# copy of main df
refined_df = project_dataset.copy()
```

[44]
```
# adding CIS region
refined_df['region'] = (np.where(
                        refined_df['country'].isin(['RU', 'UA', 'BE', 'AZ', 'AM', 'KZ', 'KG', 'MD', 'TJ', 'TM', 'UZ']),
                        'CIS',
                        refined_df['region']))
```

[45]
```
# adding install datetime
install_times = refined_df[refined_df['event_name']=='FirstLaunchApp']
install_times = install_times.drop_duplicates(subset=('user_id'))
install_times = install_times[['user_id', 'event_time']]
install_times = install_times.rename(columns={'event_time' : 'install_time'})
install_times['install_date'] = install_times['install_time'].dt.date
refined_df = refined_df.merge(install_times, on='user_id')
refined_df.head(1)
```

	event_time	event_date	event_name	revenue_usd	region	country	device_type	platform	cohort_date	user_id	user_type	content_id	currency	revenue
0	2020-12-14 21:16:41	2020-12-14	LaunchApp	0.0	CIS	KG	samsung-SM-G928F	android	2020-12-12	212364	organic	None	None	0.0

```
[46]
# Removing events which happend before install times.
refined_df = refined_df[refined_df['event_time'] >= refined_df['install_time']].copy()
# adding living hours
refined_df['lifehour'] = ((refined_df['event_time'] - refined_df['install_time']).dt.total_seconds() / 3600).astype('int')
# adding living days
refined_df['lifeday'] = (refined_df['event_time'] - refined_df['install_time']).dt.days
refined_df.head(2)
```

	event_time	event_date	event_name	revenue_usd	region	country	device_type	platform	cohort_date	user_id	user_type	content_id	currency	reve
0	2020-12-14 21:16:41	2020-12-14	LaunchApp	0.0	CIS	KG	samsung-SM-G928F	android	2020-12-12	212364	organic	None	None	
1	2020-12-14 23:56:25	2020-12-14	achieve_level_6	0.0	CIS	KG	samsung-SM-G928F	android	2020-12-12	212364	organic	None	None	

```
[47]
refined_df = refined_df[refined_df['install_date'] > pd.to_datetime('2020-11-10')]
refined_df = refined_df[refined_df['install_date'] < pd.to_datetime('2021-02-10')]
refined_df = refined_df[refined_df['lifeday'] < 90]
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/ops/array_ops.py:73: FutureWarning:

Comparison of Timestamp with datetime.date is deprecated in order to match the standard library behavior.  In a future version these will be considered
```

Функции

AVERAGE CARPU

```
[48]
def GetAverageCarpu(df, cohort_name):
    install_users = df.groupby(['install_date']).agg(install_users=('user_id', 'nunique'))
    df = df.merge(install_users, on='install_date')
    df = df.groupby(['install_date', 'lifeday']).apply(lambda g: pd.Series({
        'total_revenue' : g['revenue_usd'].sum(),
        'total_users' : g['install_users'].iloc[0],
        })).reset_index()
    df = df.groupby(['install_date']).apply(lambda g: g.assign(
        cum_sum=g['total_revenue'].cumsum(),
        CARPU=g['total_revenue'].cumsum()/g['total_users'].iloc[0]
        )).droplevel(0)
    df.sort_values(by=['install_date', 'lifeday'])
    df = df.groupby('lifeday')['CARPU'].mean().reset_index()
    df['cohort'] = cohort_name
    return df

```

```
[49]
def ShowAverageCarpuForCohort(cohort, name):
    result = GetAverageCarpu(cohort[cohort['region']=='CIS'], name + '_cis')
    result = result.append(GetAverageCarpu(cohort[cohort['region']=='NA'], name + '_na'))

    display(px.line(result, x='lifeday', y='CARPU', color='cohort').show())
```

```
[50]
def GetCarpuPerDay(df):
    install_users = df.groupby(['install_date']).agg(install_users=('user_id', 'nunique'))
    df = df.merge(install_users, on='install_date')
    df = df.groupby(['install_date', 'lifeday']).apply(lambda g: pd.Series({
        'total_revenue' : g['revenue_usd'].sum(),
        'total_users' : g['install_users'].iloc[0],
        })).reset_index()
    df = df.groupby(['install_date']).apply(lambda g: g.assign(
        cum_sum=g['total_revenue'].cumsum(),
        CARPU=g['total_revenue'].cumsum()/g['total_users'].iloc[0]
        )).droplevel(0)
    df.sort_values(by=['install_date', 'lifeday'])
    #pivot = df.pivot(index='install_date', columns='lifeday', values='CARPU')
    return df
```

```
[51]
    df = GetCarpuPerDay(refined_df)
    pivot = df.pivot(index='install_date', columns='lifeday', values='CARPU')
```

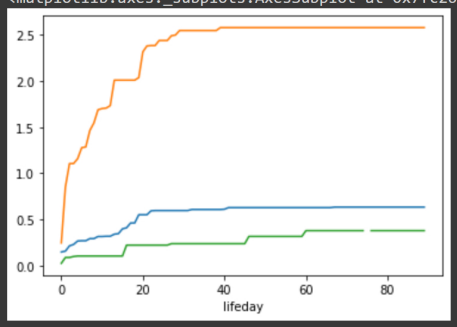
Ради интереса построим CARPU для нескольких случайных дней инсталла.

На графиках видно что в конце концов CARPU выходит на плато и больше значимо не растет

На графиках видно что в конце концов CAC'у выходит на плато и больше значимо не растет.

```
[52] 1  display(pivot.iloc[11].plot())
     2  display(pivot.iloc[31].plot())
     3  display(pivot.iloc[60].plot())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc26bda1dd0>
<matplotlib.axes._subplots.AxesSubplot at 0x7fc26bda1dd0>
<matplotlib.axes._subplots.AxesSubplot at 0x7fc26bda1dd0>
```



ROLLING

```
[53] 1  def get_arpdau(df):
     2      result = df.groupby('event_date')['revenue_usd'].sum() / df.groupby('event_date')['user_id'].nunique()
     3      result = result.mean()
     4      return result
```

```
[54] 1  def get_rolling_retention(data_frame):
     2      copy_df = data_frame
     3      copy_df = copy_df.groupby('user_id', as_index = False).agg(max_lifeday=('lifeday', 'max'))
     4
     5      df_1_to_24 = pd.DataFrame({'lifeday' : range(0, 90)})
     6      df_1_to_24['for_merge'] = 1
     7      copy_df['for_merge'] = 1
     8
     9      df_rolling = pd.merge(df_1_to_24, copy_df, on = 'for_merge')
    10      df_rolling = df_rolling[df_rolling['max_lifeday'] >= df_rolling['lifeday']]
    11      df_rolling = (df_rolling.groupby('lifeday', as_index = False).agg(unique_users=('user_id', 'nunique'))
    12                                                                    .sort_values(by='lifeday'))
    13      df_rolling['rolling_retention'] = round(df_rolling['unique_users'] / df_rolling['unique_users'][0], 4) * 100
    14
    15      return df_rolling
```

```
[55] 1  def show_ltv_via_rolling(df, name):
     2      retention = get_rolling_retention(df)
     3      lifetime = (retention['rolling_retention'] / 100).sum()
     4      arpdau = get_arpdau(df)
     5      result = lifetime * arpdau
     6      print(name + " " + str(result))
     7      return result
```

```
[56] 1  def show_rolling_ltv(cohort, name):
     2      show_ltv_via_rolling(cohort[cohort['region']=='CIS'], name + '_cis')
     3      show_ltv_via_rolling(cohort[cohort['region']=='NA'], name + '_na')
     4      print('----')
```

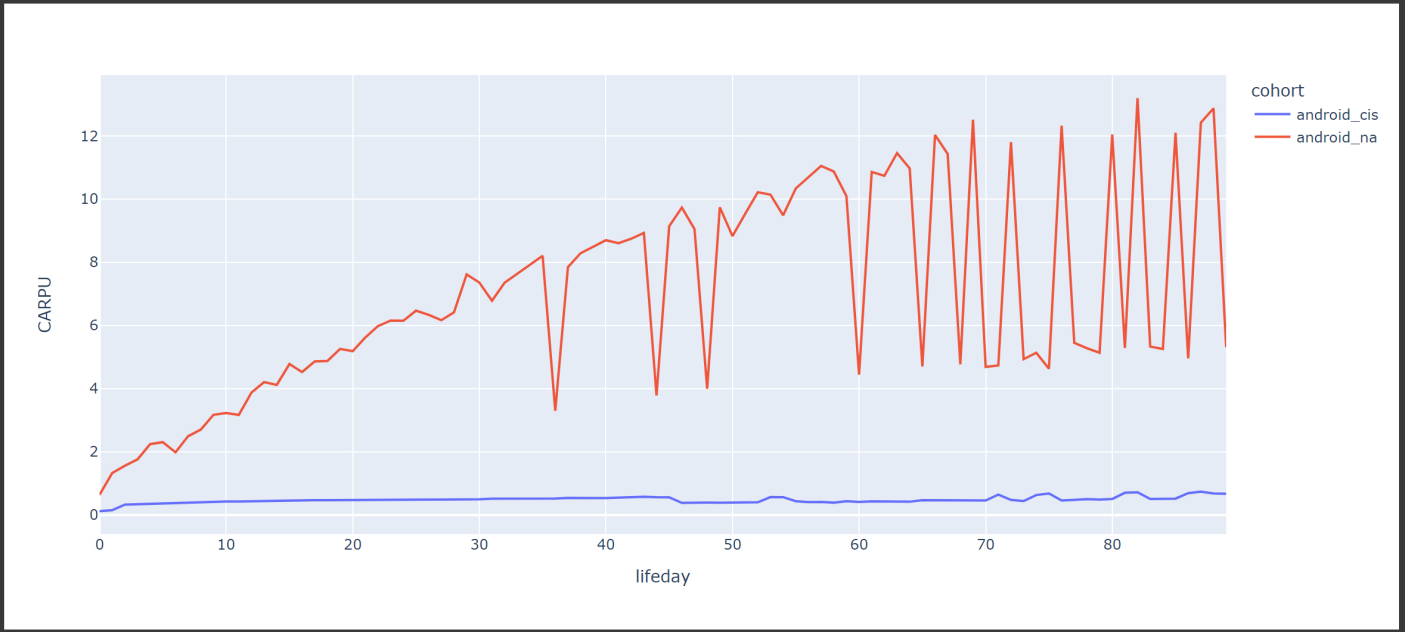
Retention

```
[57] 1  def get_retention_rate(data_frame):
     2      copy_df = data_frame
     3
     4      copy_df = copy_df.groupby('lifeday', as_index = False).agg(unique_users=('user_id', 'nunique')).sort_values(by='lifeday')
     5      copy_df['retention'] = round(copy_df['unique_users'] / copy_df['unique_users'][0], 4) * 100
     6
     7      return copy_df
     8
     9
    10  def show_ltv_via_retention(df, name):
    11      retention = get_retention_rate(df)
    12      lifetime = (retention['retention'] / 100).sum()
    13      arpdau = get_arpdau(df)
    14      result = lifetime * arpdau
    15      print(name + " " + str(result))
    16      return result
    17
    18
    19  def show_retention_ltv(cohort, name):
```

```
    show_ltv_via_retention(cohort[cohort['region']=='CIS'], name + '_cis')
    show_ltv_via_retention(cohort[cohort['region']=='NA'], name + '_na')
    print('----')
```

1. Android

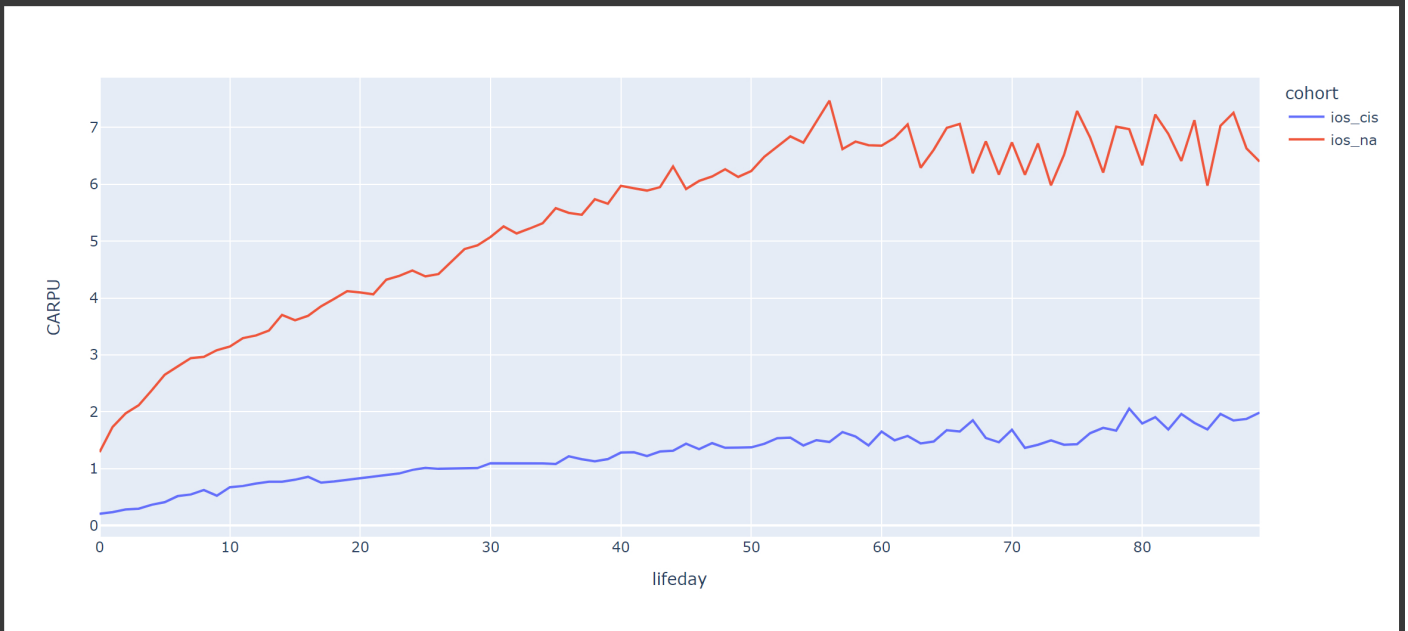
```
[58]  ShowAverageCarpuForCohort(refined_df[refined_df['platform']=='android'], 'android')
      show_rolling_ltv(refined_df[refined_df['platform']=='android'], 'android')
      show_retention_ltv(refined_df[refined_df['platform']=='android'], 'android')
```




```
None
android_cis 0.7012674812181283
android_na 8.71079309853721
----
android_cis 0.34849130010341867
android_na 4.209621212843139
----
```

1. IOS

```
[59]  ShowAverageCarpuForCohort(refined_df[refined_df['platform']=='ios'], 'ios')
      show_rolling_ltv(refined_df[refined_df['platform']=='ios'], 'ios')
      show_retention_ltv(refined_df[refined_df['platform']=='ios'], 'ios')
```

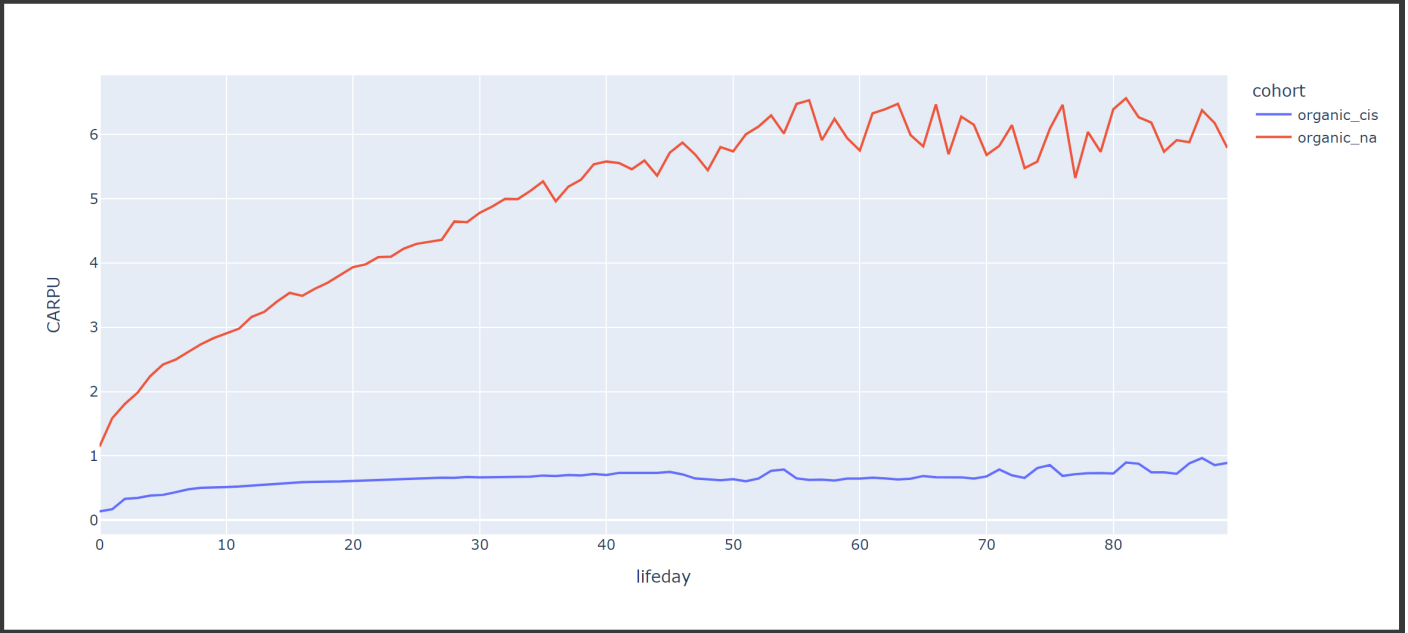



```
None
ios_cis 1.7815401796828456
ios_na 10.79870833512914
----
ios_cis 0.8579563440828234
```

ios_na 5.108650599790054
---

1. ORGANIC

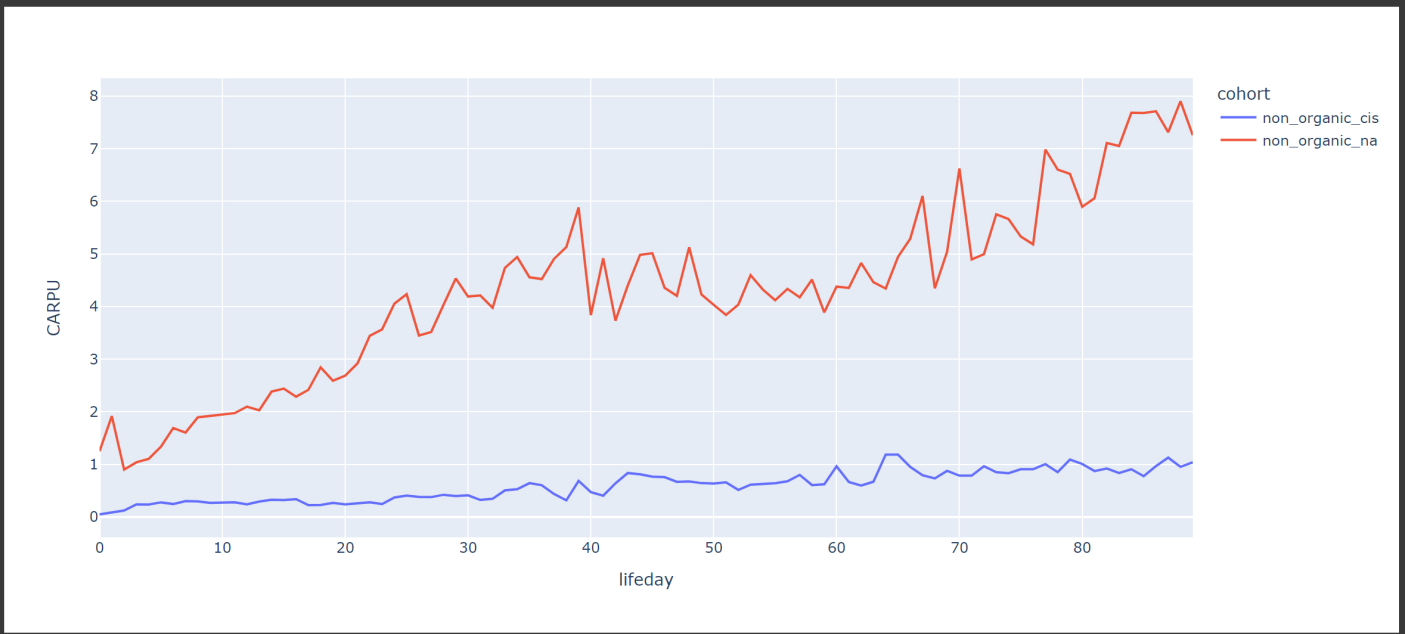
```
[60]  1  ShowAverageCarpuForCohort(refined_df[refined_df['user_type']=='organic'], 'organic')
      2  show_rolling_ltv(refined_df[refined_df['user_type']=='organic'], 'organic')
      3  show_retention_ltv(refined_df[refined_df['user_type']=='organic'], 'organic')
```



None
organic_cis 0.9665162635374484
organic_na 10.188857490966747
---
organic_cis 0.4771642295991386
organic_na 4.835725610876053
---

1. NON ORGANIC

```
[61]  1  ShowAverageCarpuForCohort(refined_df[refined_df['user_type']=='non_organic'], 'non_organic')
      2  show_rolling_ltv(refined_df[refined_df['user_type']=='non_organic'], 'non_organic')
      3  show_retention_ltv(refined_df[refined_df['user_type']=='non_organic'], 'non_organic')
```



None
non_organic_cis 0.647981809089102
non_organic_na 9.531639423966949
---
non_organic_cis 0.3095068071540245
non_organic_na 4.575676407937269

Вывод

Все методы показывают разные значения LTV.
Разница весьма существенна. LTV от retention rate самый маленький он примерно в 2 раза меньше чем LTV от rolling retention.
Среднее CARPU N-ного где то по середине между ними (если считать максимально стабильное значение = LTV).

Все методы показывают что среди всех когорт NA - безусловный лидер по LTV.

Если смотреть графики то среднее CARPU N-ного дня стабилизируется примерно на 60 день.

Какой метод самый практичный?
Если любому из этих методов можно доверять. И мы не знаем каой метод самый точный. То самым практичным будет метод который проще всего выполнить. Для меня это LTV через Retention Rate.