Etap 1

autorzy

- Julia Jodczyk
- Filip Pawłowski

Zadanie:

W ramach projektu wcielamy się w rolę analityka pracującego dla portalu„Pozytywka” – serwisu muzycznego, który swoim użytkownikom pozwala na odtwarzanie ulubionych utworów online. Praca na tym stanowisku nie jest łatwa –zadanie dostajemy w formie enigmatycznego opisu i to do nas należy doprecyzowanie szczegółów tak, aby dało się je zrealizować. To oczywiście wymaga zrozumienia problemu, przeanalizowania danych, czasami negocjacji z szefostwem. Same modelemusimy skonstruowaćtak, aby gotowe były do wdrożenia produkcyjnego – pamiętając, że w przyszłości będą pojawiać się kolejne ich wersje,z którymi będziemy eksperymentować.

“Jakiś czas temu wprowadziliśmy konta premium, które uwalniają użytkowników od słuchania reklam. Nie są one jednak jeszcze zbyt popularne – czy możemy się dowiedzieć, które osoby są bardziej skłonne do zakupu takiego konta?”

Definicja problemu biznesowego

Z biznesowego punktu widzenia, chcemy móc identyfikować użytkowników skłonnych do zakupu konta premium, aby bardziej skupić się na tej grupie pod względem reklamowania kont premium. Proces identyfikacji powinien odbywać się w pewnych odstępach czasowych - być może zainteresowanie użytkowników kontem premium w zależności od różnych czynników się zmienia.

Zadanie modelowania

Zadaniem modelowania będzie klasyfikacja binarna:

1. użytkownicy, którzy kupili konto premium
2. użytkonicy, którzy nie kupili konta premium.

Założenia

- Dane będą analizowane w odstępach czasowych, które zdefiniujemy po analizie danych. Oznacza to, że jeśli użytkownik kupi konto premium w n-tym analizowanym okresie, to zostaje wykluczony z analizy w okresach następujących po nim.
- Głównym źródłem analizy będą historie sesji użytkowników (jak długo przebywają na platformie, z jaką częstotliwością wyświetlają się im reklamy, itp.).
- Pod uwagę weźmiemy również pozostałe cechy użytkowników - ulubione gatunki i miejsce zamieszkania.

Kryteria sukcesu

- Klasyfikator działający na poziomie 70%.

Analiza danych

In [53]:
```
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
import matplotlib.dates as mdates

users_df = pd.read_json("users.json")
tracks_df = pd.read_json("tracks.json")
artists_df = pd.read_json("artists.json")
sessions_df = pd.read_json("sessions.json")
```

Analiza zbiorów danych

Analizę zbiorów przeprowadziliśmy na dwóch wersjach zbioru danych. Poniższe wydruki pochodzą z pracy z nowszą wersją.

Plik users.json

Sprawdzenie czy w danych istnieją wartości nieokreślone (*null*):

In [54]:
```
users_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 954 entries, 0 to 953
Data columns (total 6 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   user_id           954 non-null    int64 
 1   name              954 non-null    object
 2   city              954 non-null    object
 3   street            954 non-null    object
 4   favourite_genres  954 non-null    object
 5   premium_user      954 non-null    bool  
dtypes: bool(1), int64(1), object(4)
memory usage: 38.3+ KB
```

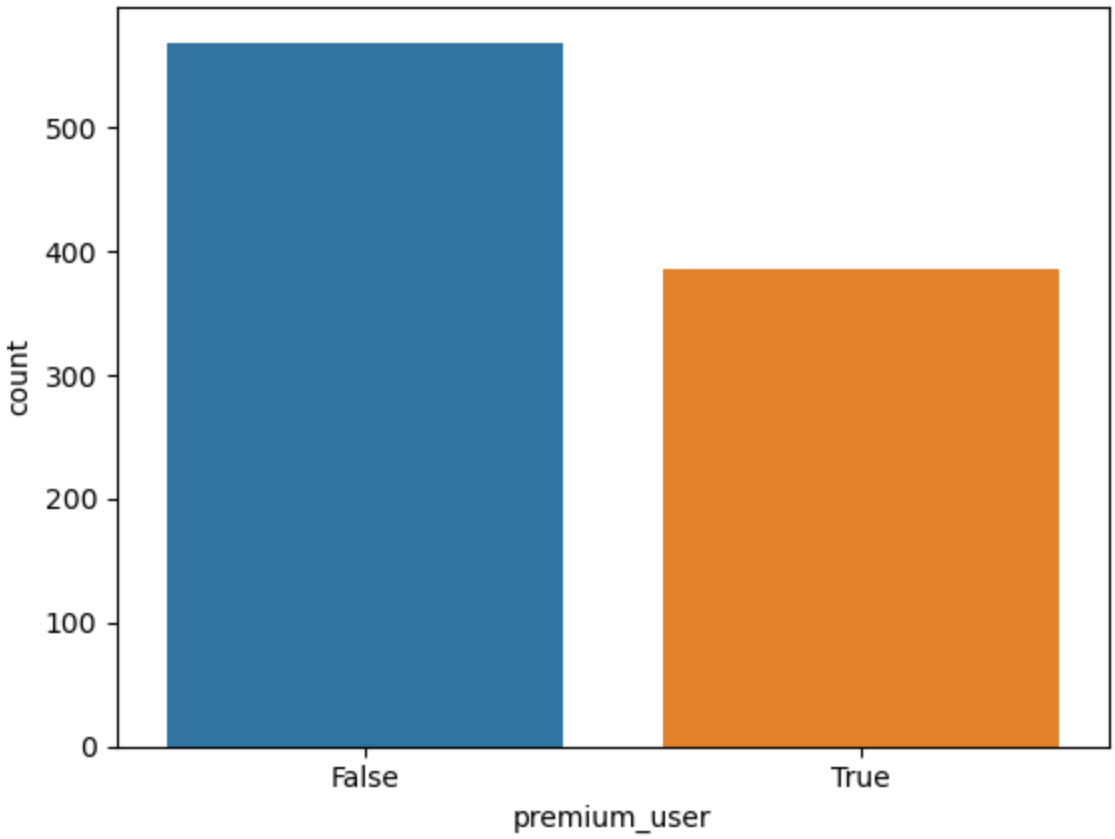
In [55]:
```
print("Number of uniqe values: ", users_df.loc[:,'user_id'].nunique(), "\nNu
```

```
Number of uniqe values:  954
Number of values:  954
```

Stosunek ilości użytkowników zwykłych do użytkowników premium:

In [56]:
```
sns.countplot(data=users_df, x='premium_user')
```

Out[56]:
```
<Axes: xlabel='premium_user', ylabel='count'>
```



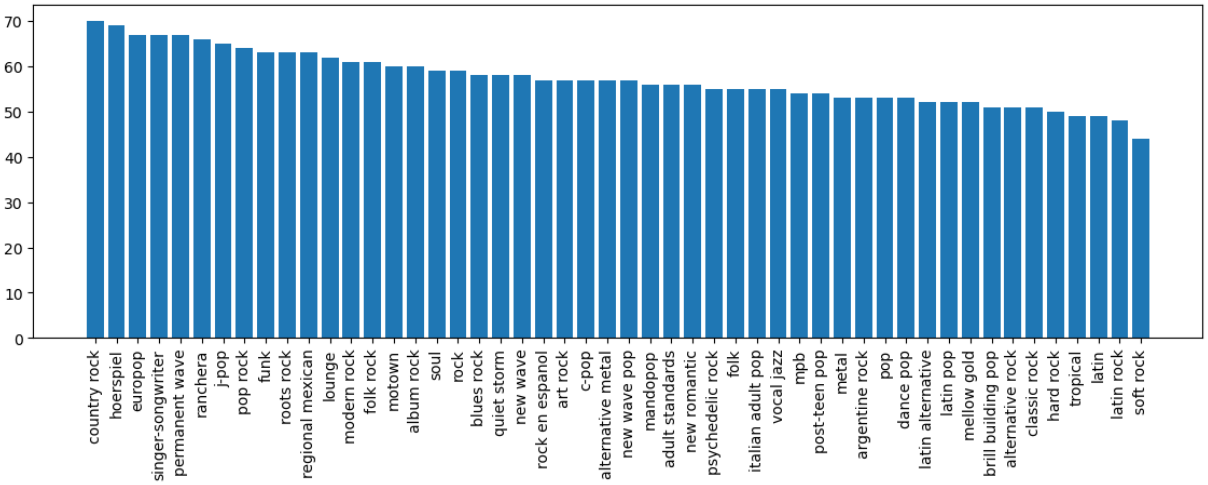
Rozkład ulubionych gatunków muzycznych użytkowników:

In [57]:
```
def to_1D(series):
  return pd.Series([x for _list in series for x in _list])
```

```
users_df.favourite_genres.fillna('', inplace=True)
unique_values = to_1D(users_df["favourite_genres"]).value_counts().index.tol
unique_value_counts = to_1D(users_df["favourite_genres"]).value_counts().val

fig, ax = plt.subplots(figsize = (14,4))
plt.xticks(rotation=90)
ax.bar(unique_values, unique_value_counts)
```

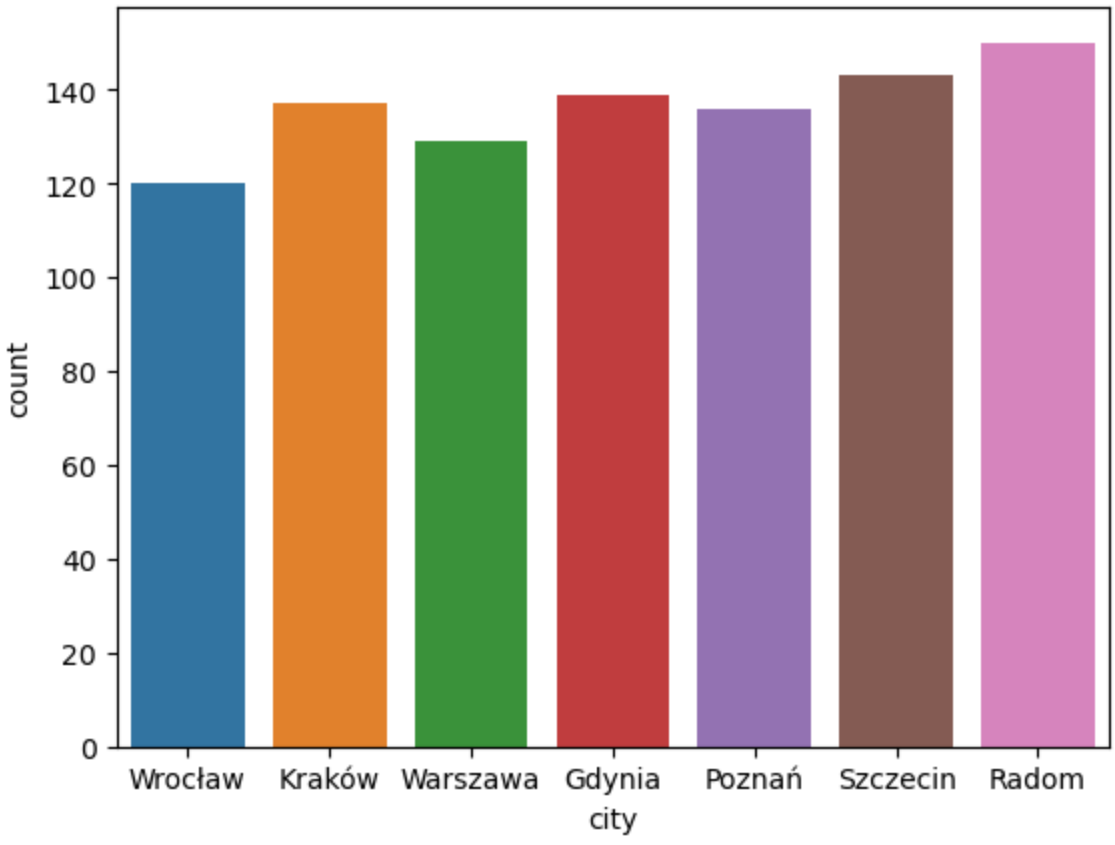
Out[57]: <BarContainer object of 50 artists>



Rozkład miast użytkowników:

In [58]:
```
sns.countplot(data=users_df, x='city')
```

Out[58]: <Axes: xlabel='city', ylabel='count'>



Plik sessions.json

Sprawdzenie czy w danych istnieją wartości nieokreślone (*null*):

In [6]:
```
sessions_df.loc[:, 'timestamp'] = pd.to_datetime(sessions_df.loc[:, 'timesta
sessions_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 484214 entries, 0 to 484213
Data columns (total 5 columns):
 #   Column      Non-Null Count   Dtype         
---  ------      --------------   -----         
 0   timestamp   484214 non-null  datetime64[ns]
 1   user_id     484214 non-null  int64         
 2   track_id    484214 non-null  object        
 3   event_type  484214 non-null  object        
 4   session_id  484214 non-null  int64         
dtypes: datetime64[ns](1), int64(2), object(2)
memory usage: 18.5+ MB
```

```
/tmp/ipykernel_22429/2145538459.py:1: DeprecationWarning: In a future versio
n, `df.iloc[:, i] = newvals` will attempt to set the values inplace instead o
f always setting a new array. To retain the old behavior, use either `df[df.c
olumns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i, newvals)
`
  sessions_df.loc[:, 'timestamp'] = pd.to_datetime(sessions_df.loc[:, 'timest
amp'])
```

In [7]:
```
sessions_df.loc[:, 'event_type'].unique()
```

Out[7]:
```
array(['PLAY', 'LIKE', 'ADVERTISEMENT', 'SKIP', 'BUY_PREMIUM'],
      dtype=object)
```

Sprawdzenie, czy istnieją sesje o zerowej długości:

In [8]:
```
sessions_df.sort_values('timestamp', ascending=True, inplace=True)
differences = list()

zero_duration_sessions = 0
for session_id, session in sessions_df.groupby('session_id'):
    sessions_starting_points = session.loc[session.loc[:,'event_type'] == 'P
    sessions_starting_points['prev_timestamp'] = sessions_starting_points['t

    # Calculate the time difference between consecutive records
    sessions_starting_points['time_diff'] = sessions_starting_points['timest
    zero_duration_sessions += np.sum(sessions_starting_points['time_diff']==

zero_duration_sessions
```

Out[8]:
```
0
```

Sprawdzenie, czy wszyscy użytkownicy z pliku sessions.json są w pliku users.json

In [9]:
```
assert len([x for x in sessions_df.loc[:, 'user_id'].unique() if x in users_
```

In [10]:
```
print("First date in the dataset: ", sessions_df.loc[:, "timestamp"].min().s
print("Last date in the dataset: ", sessions_df.loc[:, "timestamp"].max().st
```

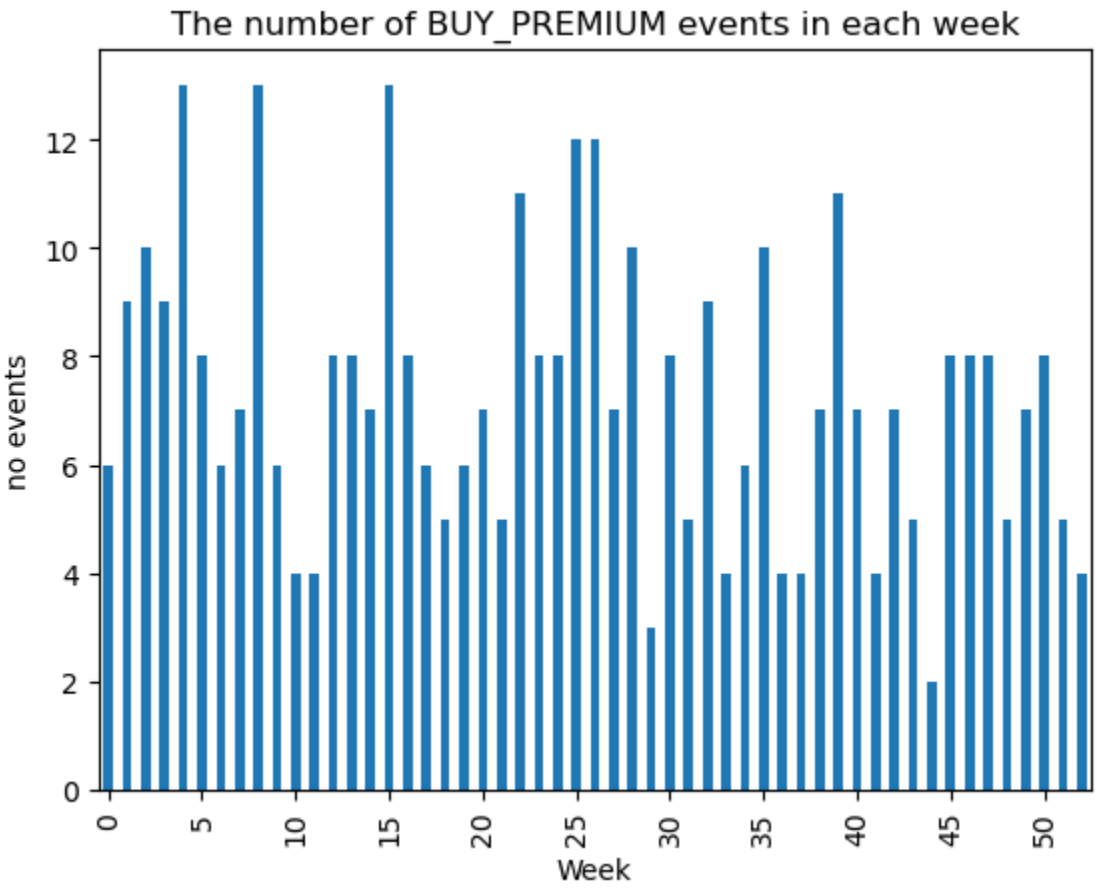
```
First date in the dataset:  31/03/2022
Last date in the dataset:  31/03/2023
```

Wykres ilości kupionych kont premium w kolejnych tygodniach:

In [11]:
```
# group by week and count the number of records in each week
weekly_counts = sessions_df[sessions_df['event_type'] == 'BUY_PREMIUM'].grou
ax = weekly_counts.plot(kind="bar")
ax.set_title("The number of BUY_PREMIUM events in each week")
ax.set_xlabel("Week")
ax.set_ylabel("no events")

ax.set_xticks(range(0, len(weekly_counts), 5))
ax.set_xticklabels(range(0, len(weekly_counts), 5))

plt.show()
```



Wnioski

Pierwsza wersja danych

Do naszych potrzeb najbardziej użyteczne będą pliki *users.json* i *sessions.json*. Dodatkowo w celu identyfikacji gatunku używane będą pliki *tracks.json* i *artists.json*.

Wspólne defekty danych:

- wiele brakujących wartości (null),
- mała ilość danych,
- niereprezentatywne dane.

Dodatkowo:

1. Plik users.json:
    - Atrybut o nazwie "id" ma niedeskryptywną nazwę oraz jest obecny tylko w kilku próbkach.
    - Brak jednoznacznego oznaczania lokalizacji - aleja -> al., ulica -> ul.
2. Plik sessions.json:
    - Niektóre sesje trwają 0 sekund,
    - Brak wartości "event_type" dla reklam.

Druga wersja danych

- Dane zostały wyczyszczone - nie ma wartości null,
- danych jest więcej,
- obejmują tylko jeden rok,
- większość problemów zaznaczonych przy pierwszej wersji danych została rozwiązana,
- dane są zbilansowane pod względem miejsca zamieszkania i ulubionych gantunków muzycznych użytkowników,
- rozkład zakupu premium w różnych okresach czasu (tygodniach) jest nierównomierny.

Model bazowy klasyfikatora

In [50]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neighbors import NearestCentroid
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MultiLabelBinarizer, LabelBinarizer
```

In [13]:
```
mlb = MultiLabelBinarizer(sparse_output=True)
lb = LabelBinarizer()

users_df = users_df.join(
            pd.DataFrame.sparse.from_spmatrix(
                mlb.fit_transform(users_df.pop('favourite_genres')),
```

```
                index=users_df.index,
                columns=mlb.classes_))


users_df = users_df.join(
            pd.DataFrame(
                lb.fit_transform(users_df.pop('city')),
                index=users_df.index,
                columns=lb.classes_))
```

Kontrolne sprawdzenie, czy nie ma wartości null:

In [14]:
```
assert users_df.isna().sum().sum() == 0
```

In [15]:
```
def calculate_ads_time(df):
    ads_mask = df.loc[:,'event_type'] == "ADVERTISEMENT"
    ads_time = np.timedelta64(0)
    for _, action in df.loc[ads_mask].iterrows():
        difference = action.next_timestamp - action.timestamp
        if difference < np.timedelta64(0):
            pass
        ads_time += difference
    return ads_time
```

Wyznaczanie stosunku między czasem reklam a czasem pozostałych akcji

In [48]:
```
sessions_filtered = pd.DataFrame()
time_comparison_df = pd.DataFrame()

for user_id, user_actions in sessions_df.groupby('user_id'):
    user_bought_premium = False
    user_ads_time = np.timedelta64(0)
    user_all_time = np.timedelta64(0)

    for session_id, session in user_actions.groupby('session_id'):
        if user_bought_premium:
            break

        session.sort_values(by=['timestamp'])
        user_all_time += session.iloc[-1, session.columns.get_loc("timestamp

        premium_bought_mask = session.loc[:,'event_type'] == "BUY_PREMIUM"
        bought_premium_in_session = premium_bought_mask.any()

        if bought_premium_in_session:
            time_of_buy_premium = session.loc[premium_bought_mask].timestamp
            session = session.loc[session.loc[:, 'timestamp'] <= time_of_buy
            user_bought_premium = True

        sessions_filtered = pd.concat([sessions_filtered, session])
        session.loc[:, 'next_timestamp'] = session.loc[:, 'timestamp'].shift

        user_ads_time += calculate_ads_time(session)

    # ads_time = np.sum((user_actions.loc[ads_mask]["next_timestamp"] - user
```

```
    session_times_df = pd.DataFrame(
        {
            'all_time': user_all_time/ np.timedelta64(1, 's'),
            'ads_time': user_ads_time/ np.timedelta64(1, 's'),
         },
            index=[user_id],
    )
    time_comparison_df = pd.concat([time_comparison_df, session_times_df])
```

In [18]:
```
users_df = users_df.join(
    pd.DataFrame(data=time_comparison_df.loc[:,'ads_time']/time_comparison_d
    on="user_id")
```

Klasyfikator minimalno-odległościowy jako klasyfikator bazowy

In [51]:
```
X = users_df.drop(["premium_user", "street", "name", "user_id"], axis=1)
y = users_df.loc[:, "premium_user"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, ra
kneigh = NearestCentroid()
kneigh.fit(X_train, y_train)

y_hat = kneigh.predict(X_test)
score = accuracy_score(y_test, y_hat)
score
```

```
/home/julia/anaconda3/lib/python3.10/site-packages/sklearn/utils/validation.p
y:768: UserWarning: pandas.DataFrame with sparse columns found.It will be con
verted to a dense numpy array.
  warnings.warn(
/home/julia/anaconda3/lib/python3.10/site-packages/sklearn/utils/validation.p
y:768: UserWarning: pandas.DataFrame with sparse columns found.It will be con
verted to a dense numpy array.
  warnings.warn(
```

Out[51]:
```
0.5206349206349207
```

K najbliższych sąsiadów jako klasyfikator bazowy

In [46]:
```
X = users_df.drop(["premium_user", "street", "name", "user_id"], axis=1)
y = users_df.loc[:, "premium_user"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, ra
kneigh = KNeighborsClassifier()
kneigh.fit(X_train, y_train)

y_hat = kneigh.predict(X_test)
score = accuracy_score(y_test, y_hat)
score
```

```
/home/julia/anaconda3/lib/python3.10/site-packages/sklearn/utils/validation.p
y:768: UserWarning: pandas.DataFrame with sparse columns found.It will be con
verted to a dense numpy array.
  warnings.warn(
/home/julia/anaconda3/lib/python3.10/site-packages/sklearn/utils/validation.p
y:768: UserWarning: pandas.DataFrame with sparse columns found.It will be con
verted to a dense numpy array.
  warnings.warn(
```

Out[46]: 0.6063492063492063

Wnioski z bazowego modelu predykcyjnego

- model K najbliższych sąsiadów poradził sobie lepiej niż klasyfikator minimalno-odległościowy,
- może być konieczne stworzenie dodatkowych atrybutów,
- model do osiągnięcia najlepszych wyników należy dostroić (np. poprzez przeszukiwanie po hipersiatce).

Dalsze plany pracy

- zbudowanie bardziej złożonego modelu,
- feature generation
  - dodanie do analizy trendów czasowych - potrzebna kolejna wersja danych, obejmująca więcej niż jeden rok sesji,
  - momenent i częstotliwość pojawiania się reklam w trakcie sesji (np. czy było to przed ulubioną piosenką)
- sprawdzenie korelacji atrybutów, odrzucenie silnie skorelowanych atrybutów.