CS 593: Data Mining II: Advanced Algorithms for Mining Big Data

•••

Spotify Recommendation Engine

Professor: Mr. Khashayar Dehnad

Team Members

Bhagyashree Ingale: 1047532

Abhishek Panda: 10478684

Table of Contents

Introduction

As we know, music and movie streaming services have been increasingly popular in recent years. People are getting more and more interested as the time moves forward by adding more and more to the collection.

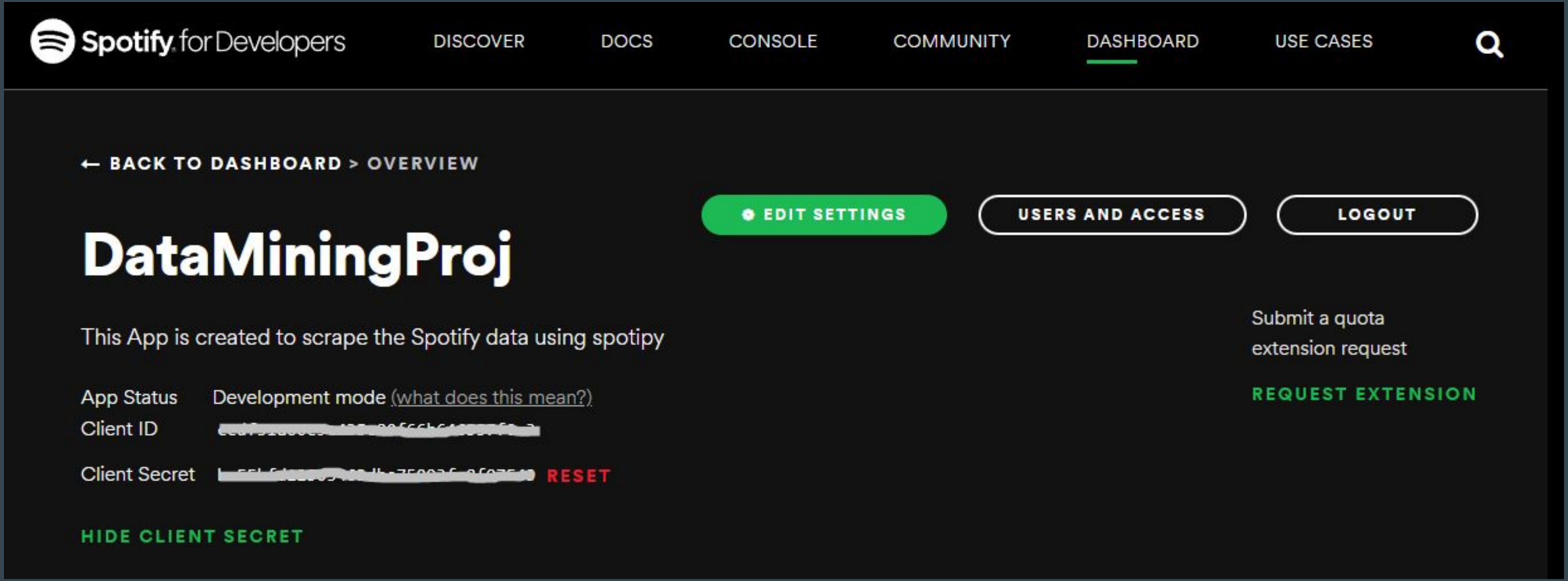
By offering the most appropriate and individualized services for each user, a recommendation system plays a critical role in ensuring a positive user experience in an application.

Spotify's recommendation algorithm has also contributed significantly to the company's success. Most of the recent developers use the Spotify recommendation algorithm to learn more about it to implement it to their own APIs.

In this project, we are building a Spotify recommendation engine based on Content-Based Filtering

Spotify Web API dashboard

We first, login with our Spotify account credentials. After that we have to create an application which generates a unique client id and client secret, that gives us access to the Spotify Web API



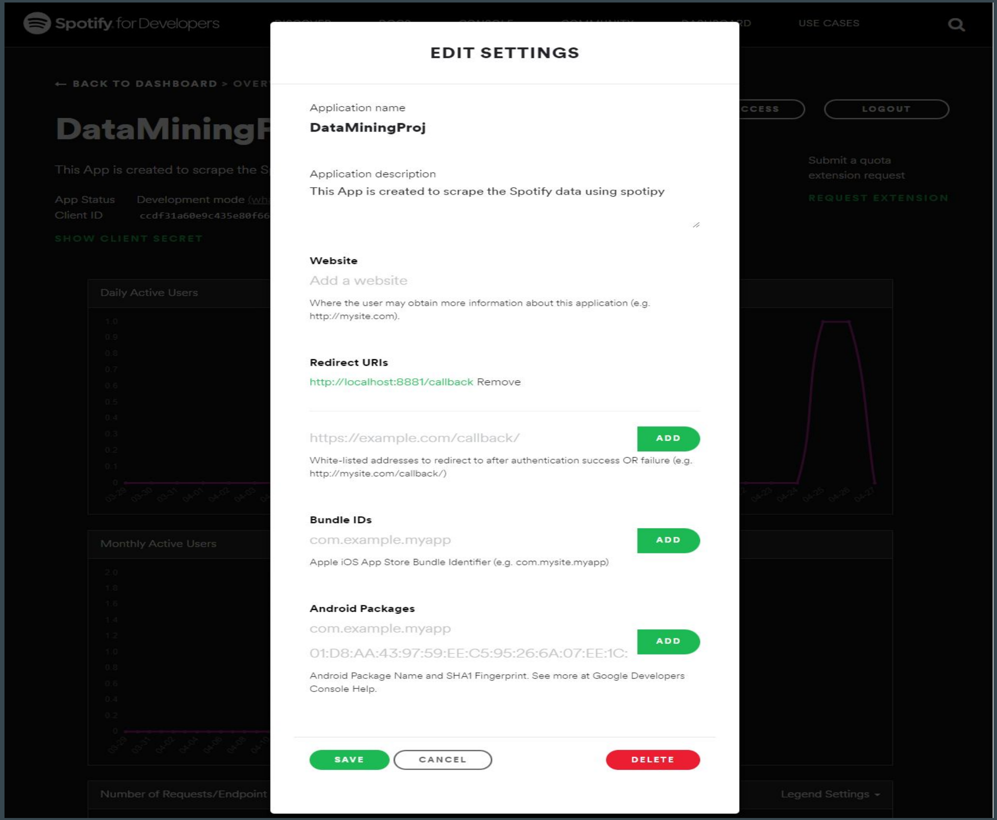
DataMiningProj

This App is created to scrape the Spotify data using spotipy

App Status	Development mode (what does this mean?)
Client ID	[redacted]
Client Secret	[redacted] RESET

Submit a quota extension request

Further, we have also added localhost URL in settings which will later be used for authentication.



EDIT SETTINGS

Application name
DataMiningProj

Application description
This App is created to scrape the Spotify data using spotipy

Website
Add a website
Where the user may obtain more information about this application (e.g. http://mysite.com).

Redirect URIs
http://localhost:8881/callback Remove

https://example.com/callback/ ADD
White-listed addresses to redirect to after authentication success OR failure (e.g. http://mysite.com/callback/)

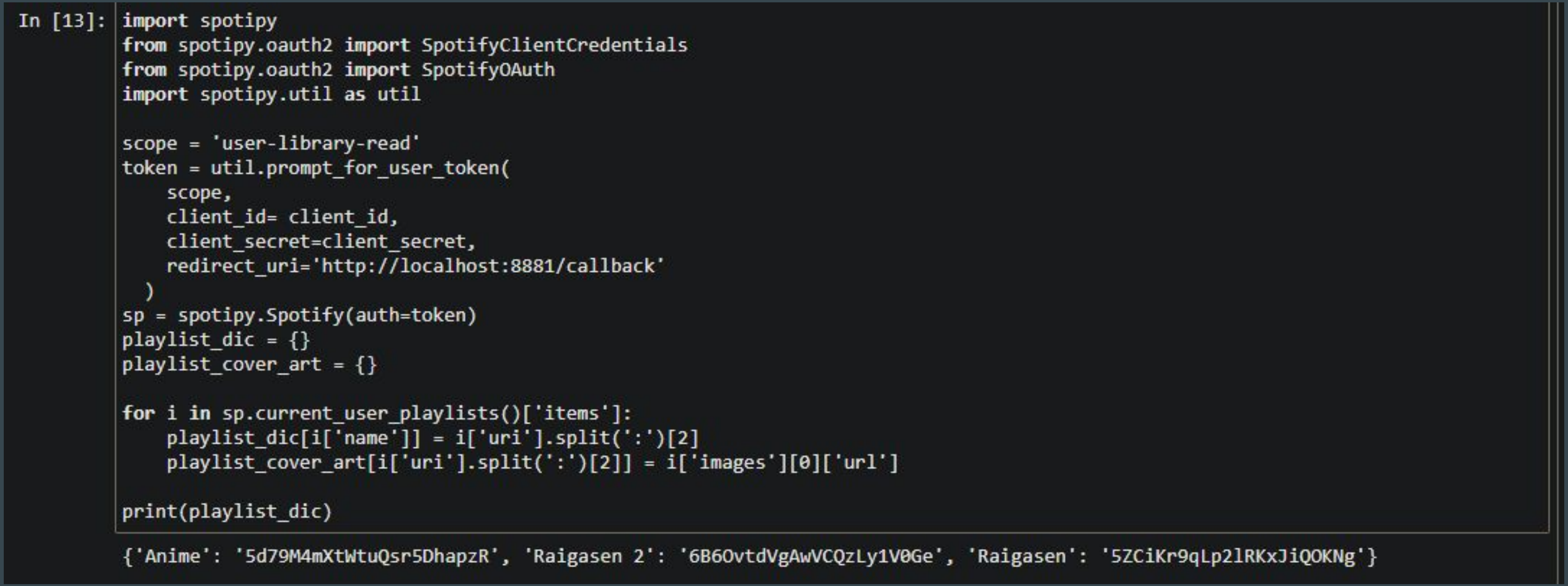
Bundle IDs
com.example.myapp ADD
Apple iOS App Store Bundle Identifier (e.g. com.mysite.myapp)

Android Packages
com.example.myapp
01:D8:AA:43:97:59:EE:C5:95:26:6A:07:EE:1C: ADD
Android Package Name and SHA1 Fingerprint. See more at Google Developers Console Help.

SAVE CANCEL DELETE

Storing playlist details in python dictionary

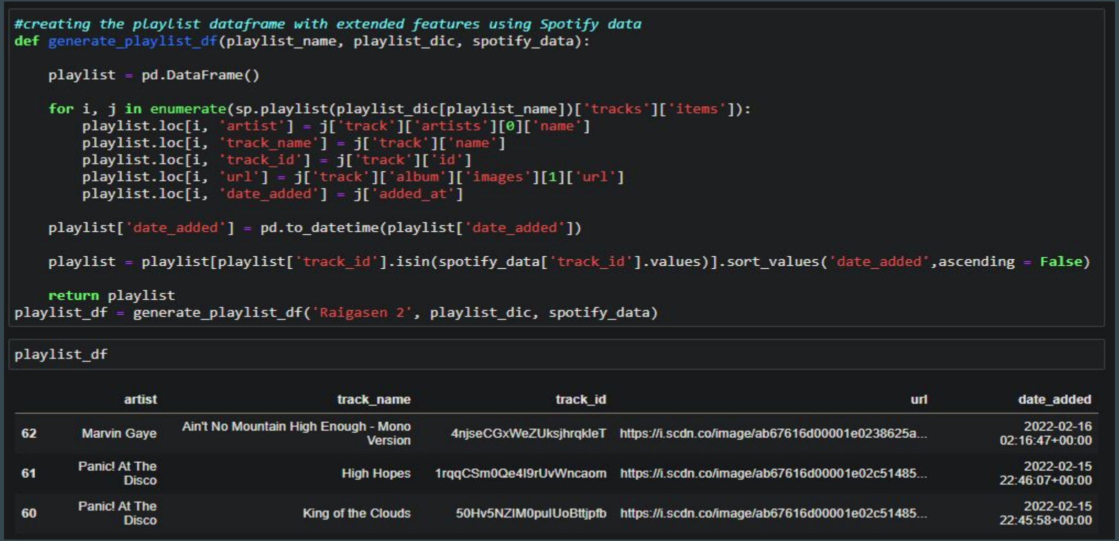
Below we are storing the playlist detail and playlist cover artist details in a Dictionary format. We must use the extracted client code and secret client code from the Spotify dashboard. If the user is not registered it will pass a “User not registered error”



```
In [13]: import spotipy
         from spotipy.oauth2 import SpotifyClientCredentials
         from spotipy.oauth2 import SpotifyOAuth
         import spotipy.util as util

         scope = 'user-library-read'
         token = util.prompt_for_user_token(
             scope,
             client_id= client_id,
             client_secret=client_secret,
             redirect_uri='http://localhost:8881/callback'
           )
         sp = spotipy.Spotify(auth=token)
         playlist_dic = {}
         playlist_cover_art = {}

         for i in sp.current_user_playlists()['items']:
             playlist_dic[i['name']] = i['uri'].split(':')[2]
             playlist_cover_art[i['uri'].split(':')[2]] = i['images'][0]['url']

         print(playlist_dic)

         {'Anime': '5d79M4mXtWtuQsr5DhapzR', 'Raigasen 2': '6B6OvtdVgAwVCQzLy1V0Ge', 'Raigasen': '5ZCiKr9qLp21RKxJiQOKNg'}
```

Storing our Playlist in a dataframe

Below, we are creating a playlist dataframe using the feature extraction from the specified datas. The playlist is populated with the mentioned below columns. After the function finishes execution, it returns the desired playlist.



```python
#creating the playlist dataframe with extended features using Spotify data
def generate_playlist_df(playlist_name, playlist_dic, spotify_data):

    playlist = pd.DataFrame()

    for i, j in enumerate(sp.playlist(playlist_dic[playlist_name])['tracks']['items']):
        playlist.loc[i, 'artist'] = j['track']['artists'][0]['name']
        playlist.loc[i, 'track_name'] = j['track']['name']
        playlist.loc[i, 'track_id'] = j['track']['id']
        playlist.loc[i, 'url'] = j['track']['album']['images'][1]['url']
        playlist.loc[i, 'date_added'] = j['added_at']

    playlist['date_added'] = pd.to_datetime(playlist['date_added'])

    playlist = playlist[playlist['track_id'].isin(spotify_data['track_id'].values)].sort_values('date_added',ascending = False)

    return playlist
playlist_df = generate_playlist_df('Raigasen 2', playlist_dic, spotify_data)
```

```python
playlist_df
```

	artist	track_name	track_id	url	date_added
62	Marvin Gaye	Ain't No Mountain High Enough - Mono Version	4njseCGxWeZUksjhrqkleT	https://i.scdn.co/image/ab67616d00001e0238625a...	2022-02-16 02:16:47+00:00
61	Panic! At The Disco	High Hopes	1rqqCSm0Qe4l9rUvWncaom	https://i.scdn.co/image/ab67616d00001e02c51485...	2022-02-15 22:46:07+00:00
60	Panic! At The Disco	King of the Clouds	50Hv5NZIM0pulUoBttjpfb	https://i.scdn.co/image/ab67616d00001e02c51485...	2022-02-15 22:45:58+00:00

Dataset and Features

To suggest a new playlist based on resemblance, We will need more information about the characteristics of songs in the Spotify app.

We'll use these qualities to determine the similarity between our playlist and songs that aren't on it.

I utilized a Kaggle dataset for my project : Spotify Tracks DB

https://www.kaggle.com/datasets/zaheenhamidani/ultimate-spotify-tracks-db

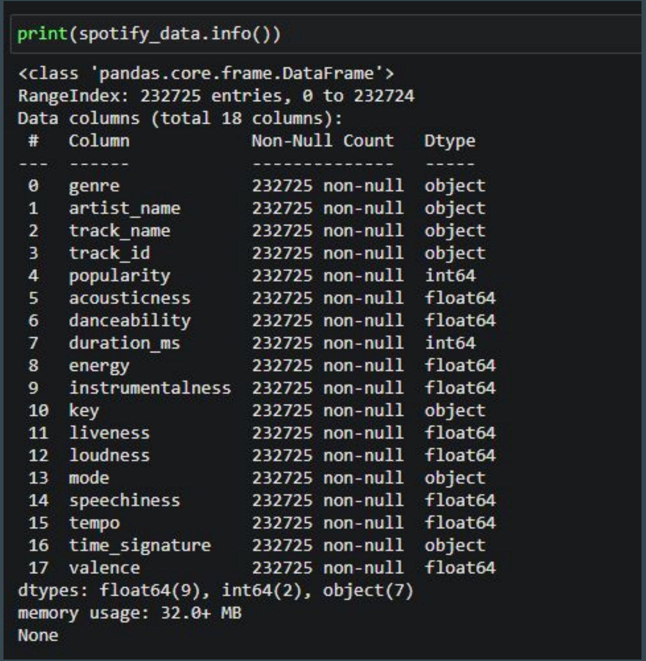
EXPLORATORY DATA ANALYSIS

The data which we have selected is from Kaggle that was uploaded by ZAHEEN HAMIDANI. It is a collection of few audio features data from the Spotify Web API which is populated into a Big Query database. This database features can be extracted as per developers' preference.

The database contains about 232k tracks.

In which there are about 26 genres.

There are about 14564 artists and few other contents.



```
print(spotify_data.info())
```

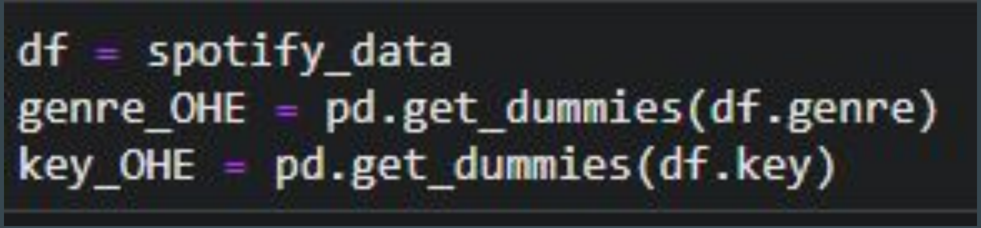
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 232725 entries, 0 to 232724
Data columns (total 18 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   genre             232725 non-null  object 
 1   artist_name       232725 non-null  object 
 2   track_name        232725 non-null  object 
 3   track_id          232725 non-null  object 
 4   popularity        232725 non-null  int64  
 5   acousticness      232725 non-null  float64
 6   danceability      232725 non-null  float64
 7   duration_ms       232725 non-null  int64  
 8   energy            232725 non-null  float64
 9   instrumentalness  232725 non-null  float64
 10  key               232725 non-null  object 
 11  liveness          232725 non-null  float64
 12  loudness          232725 non-null  float64
 13  mode              232725 non-null  object 
 14  speechiness       232725 non-null  float64
 15  tempo             232725 non-null  float64
 16  time_signature    232725 non-null  object 
 17  valence           232725 non-null  float64
dtypes: float64(9), int64(2), object(7)
memory usage: 32.0+ MB
None
```

Feature engineering

In the dataset, we can observe that multiple columns represent the possible features for a song. Out of these, few features are categorical (columns having discrete values) like genre, key, popularity index, etc.

Therefore, the first step would be to convert these categorical features into one-hot encoding (OHE) so that our songs can be represented as vectors in a feature space.

To keep it simple, for now, we are only taking two categorical features into our consideration.



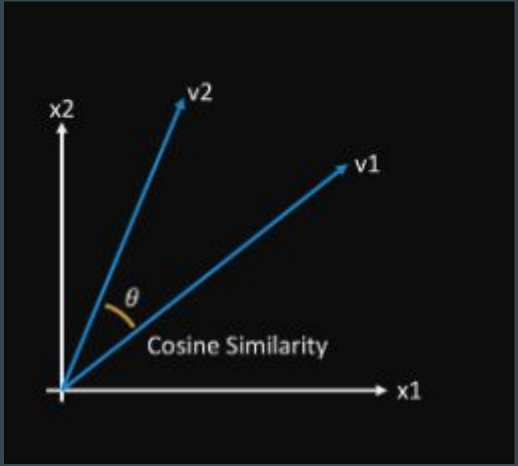
```
df = spotify_data
genre_OHE = pd.get_dummies(df.genre)
key_OHE = pd.get_dummies(df.key)
```

Cosine similarity

Cosine similarity measures the similarity between two vectors

It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction.

Lesser the angle between the two vectors more is the similarity.



Creating playlist vector

To perform the cosine similarity between our playlist and the songs not present in our playlist, we will try to summarize our playlist in one vector.

This vector will represent our playlist in the feature space, and we will be able to find songs similar to the songs in our playlist.

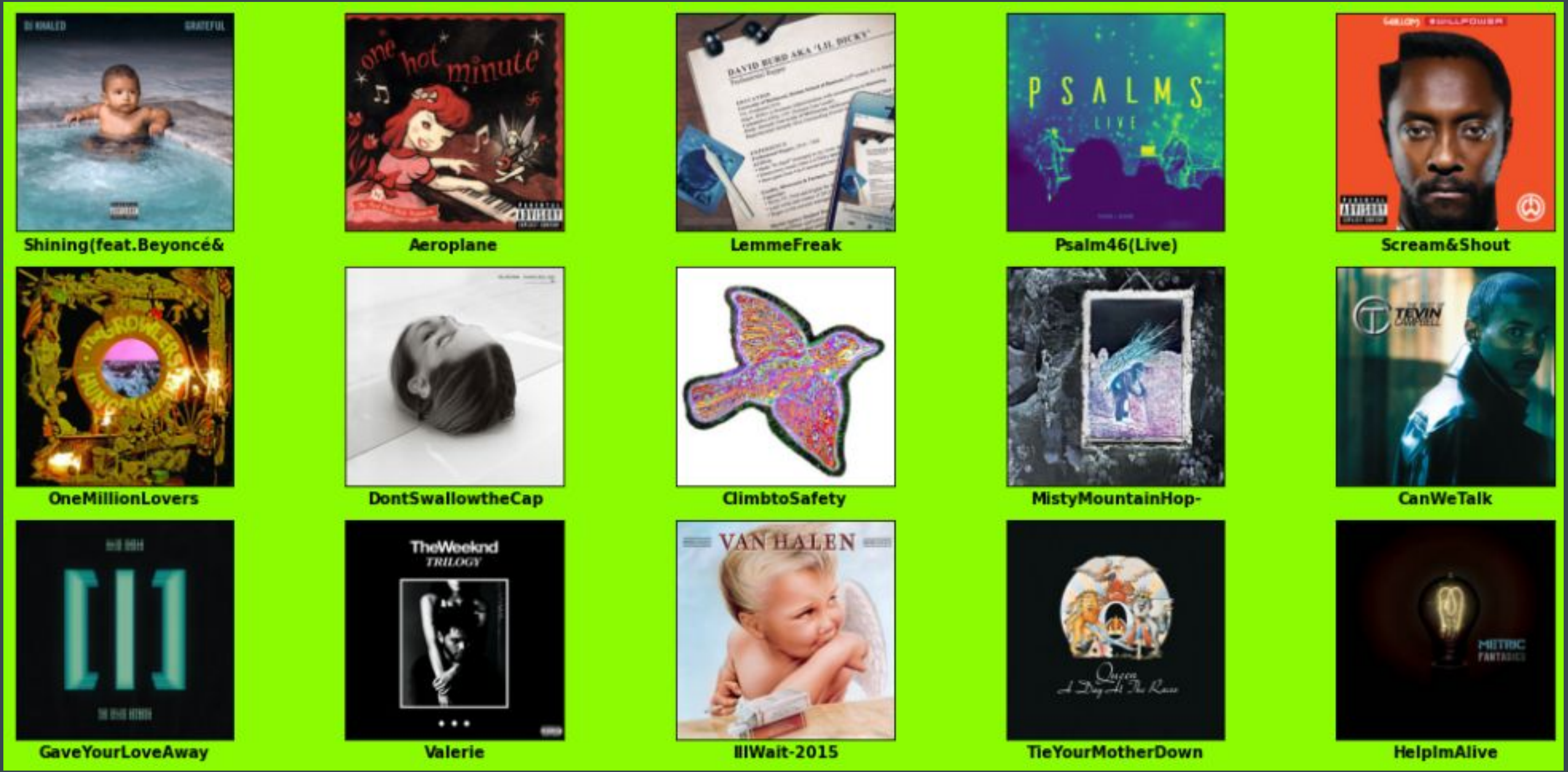
Generating recommendations

As stated before, we are going to use cosine similarity as a similarity metric to determine the songs that are very much similar to our playlist.

We will perform the cosine similarity between our playlist vector and songs not present in our playlist.

Next, we will reverse sort dataframe based on the cosine similarity column. Finally, we will generate the top 15 recommendations of songs as our recommended playlist.

Visualizing the cover-art of the recommended playlist



Conclusions

Using the concept of cosine similarity, we have seen that we were able to generate recommendations of songs that are similar to the songs in our playlist.

One thing to keep in mind is that we have used the Content-Based Filtering technique in our scenario. As a result, the model will only be able to generate suggestions based on the interests of that unique user. As a result, a user's capacity to broaden their existing interests is limited.

Limitations

The Spotify song features data that we utilized to create the recommendation algorithm may not include all of the songs in your playlist. As a result, we may not always be able to fully utilize the playlist to produce suggestions.

Users must have at least one playlist on their Spotify account to utilize this recommender, which is a disadvantage for a first-time user of the Spotify app.