Import necesarry packages

In [ ]:
```
#Install dependency
!pip install spotipy
!pip install nbconvert[webpdf]
!jupyter nbconvert --to webpdf --allow-chromium-download your-notebook-file.ipynb

#import from spotipy
import spotipy
from spotipy.oauth2 import SpotifyClientCredentials
from spotipy.oauth2 import SpotifyOAuth
import spotipy.util as util

#import following packages for data processing
from skimage import io #used to read and write images in various formats
import matplotlib.pyplot as plt #used for plotting
import pandas as pd #library used for data cleaning and analysis
from datetime import datetime

#import packages from scikit learn
from sklearn.preprocessing import MinMaxScaler #for feature normalization
from sklearn.metrics.pairwise import cosine_similarity #for finding similar songs
```

Import Dataset

In [ ]:
```
!pip install opendatasets
```

In [ ]:
```
import opendatasets as od
dataset = 'https://www.kaggle.com/zaheenhamidani/ultimate-spotify-tracks-db'
od.download(dataset)
```

In [ ]:
```
import os
```

In [ ]:
```
data_dir = './ultimate-spotify-tracks-db'
```

In [ ]:
```
os.listdir(data_dir)
```

In [ ]:
```
spotify_data = pd.read_csv('SpotifyFeatures.csv')
```

In [ ]:
```
spotify_data.head()
```

Feature Engineering

Using OHE (One Hot Encoding) convert categorical features (genre and key) into numerical features (so that the songs can be represented as vectors).

In [ ]:
```
spotify_features_df = spotify_data #store the data from the dataset in data frame variable
#using get dummies automatically run one hot encoding on the necesarry coloumns
key_OHE = pd.get_dummies(spotify_features_df.genre)
genre_OHE = pd.get_dummies(spotify_features_df.key)
```

Normalize features to prevent model from being skewed towards one attribute. Also allows the model to run more efficiently.

In [ ]:
```
scaled_features = MinMaxScaler().fit_transform([
  spotify_features_df['acousticness'].values,
  spotify_features_df['danceability'].values,
  spotify_features_df['duration_ms'].values,
  spotify_features_df['energy'].values,
  spotify_features_df['instrumentalness'].values,
  spotify_features_df['liveness'].values,
  spotify_features_df['loudness'].values,
  spotify_features_df['speechiness'].values,
  spotify_features_df['tempo'].values,
  spotify_features_df['valence'].values,
  ])
```

Store the normalized features into the dataframe

In [ ]:
```
spotify_features_df[['acousticness','danceability','duration_ms','energy','instrumentalness','liveness','loudness','speechiness','tempo','valence']] = scaled_features.T
```

Remove the features that have been converted to OHE as well as features that do not need to be considered to prove similarity

In [ ]:
```
spotify_features_df = spotify_features_df.drop('genre',axis=1)
spotify_features_df = spotify_features_df.drop('artist_name',axis=1)
spotify_features_df = spotify_features_df.drop('track_name',axis=1)
spotify_features_df = spotify_features_df.drop('key',axis=1)
spotify_features_df = spotify_features_df.drop('popularity',axis=1)
spotify_features_df = spotify_features_df.drop('mode',axis=1)
spotify_features_df = spotify_features_df.drop('time_signature',axis=1)
```

Add the OHE features to the dataframe

In [ ]:
```
spotify_features_df = spotify_features_df.join(key_OHE)
spotify_features_df = spotify_features_df.join(genre_OHE)
```

Connect to Spotify Web API to get user data

In [ ]:
```
client_id = 'c87cc9778bbb485ca8de60d4d948dd7e'
client_secret = '77a889ff0e454cd4864c5300ad386cde'
```

Create a dictionary of all of the users playlists

In [ ]:
```
scope = 'user-library-read' #provides reading access to the users library
#get authentication token
token = util.prompt_for_user_token(username = 'tushara.t', scope, client_id = client_id, client_secret=client_secret, redirect_uri = 'http://127.0.0.1:8000/callback')
#authenticate object
sp = spotipy.Spotify(auth = token)
playlists = sp.current_user_playlists()#gain read access to user playlists
playlist_dic={}
#add users playlists to dictionary (stores the playlist name and uri in key:value pair)
for i in playlists['items']:
    playlist_dic[i['name']] = i['uri'].split(':')[2]
```

Create a method that creates a dataframe for a specified playlist

In [ ]:
```
def generate_playlist_df(playlist_name, playlist_dic, spotify_data):
    #initialize dataframe variable
    playlist = pd.DataFrame()
    #populate the 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'])#converts to dates to datetime objects
    #filter out tracks whose track id's do not appear in the dataset and sort dataframe by date added in decending order
    playlist = playlist[playlist['track_id'].isin(spotify_data['track_id'].values)].sort_values('date_added',ascending = False)

    return playlist
```

Create Playlist Vector

Create a function to generate a playlist vector from the dataframes in order to perform cosine similarity

In [ ]:
```
def generate_playlist_vector(spotify_features_df, playlist_df, weight_factor):
    #create a dataframe of the tracks from the dataset's dataframe (spotify_features_df) whose track_id's are also
    #present in the playlist_df
    spotify_features_playlist = spotify_features_df[spotify_features_df['track_id'].isin(playlist_df['track_id'].values)]

    #merge the track_id and date_added columns of the playlist_df at the track_id column of the new df
    spotify_features_playlist = spotify_features_playlist.merge(playlist_df[['track_id','date_added']], on = 'track_id', how = 'inner')

    #create df of all tracks not in playlist (used for reccomendations)
    spotify_features_nonplaylist = spotify_features_df[~spotify_features_df['track_id'].isin(playlist_df['track_id'].values)]

    #Create a new df which has the spotify_features_playlist df sorted in descending order by date added
    playlist_feature_set = spotify_features_playlist.sort_values('date_added',ascending=False)

    #access date of most recently added track to playlist
    most_recent_date = playlist_feature_set.iloc[0,-1]

    #iterate through the rows in the spotify_features_playlist df
    for ix, rows in playlist_feature_set.iterrows():
        #add a new coloumn to the dataframe that contains how long from recent the song was added to the playlist
        playlist_feature_set.loc[ix,'days_from_recent']= int((most_recent_date.to_pydatetime() - row.iloc[-1].to_pydatetime()).days)

    #create a weight coloumn by applying the weight_factor on the days_from_recent column
    playlist_feature_set['weight']=playlist_feature_set['days_from_recent'].apply(lambda x: weight_factor ** (-x))
    #create new df that contains the weighted vector
    playlist_feature_set_weighted = playlist_feature_set.copy()
    #add in all rows and columns upto but not including -3 * weights into the df
    playlist_feature_set_weighted.update(playlist_feature_set_weighted.iloc[:,:-3].mul(playlist_feature_set_weighted.weight.astype(int),0))

    playlist_feature_set_weighted_final = playlist_feature_set_weighted.iloc[:, :-3]


    #return the sum of the index axis as the playlist vector and the df of all tracks not in playlist
    return playlist_feature_set_weighted_final.sum(axis = 0), spotify_features_nonplaylist
```

Generate z recomendations

Using cosine similarity between the playlist vector and songs not present in the playlist, z(user specified amount) songs simlar to those in the playlist will be reccomended.

In [ ]:
```
def generate_recommendation(spotify_data, playlist_vector, nonplaylist_df):

    non_playlist = spotify_data[spotify_data['track_id'].isin(nonplaylist_df['track_id'].values)]
    non_playlist['sim'] = cosine_similarity(nonplaylist_df.drop(['track_id'], axis = 1).values, playlist_vector.drop(labels = 'track_id').values.reshape(1, -1))[:,0]
    non_playlist_topsongs = non_playlist.sort_values('sim',ascending = False).head(z)
    non_playlist_topsongs['url'] = non_playlist_top15['track_id'].apply(lambda x: sp.track(x)['album']['images'][1]['url'])

    return  non_playlist_topsongs
```

In [ ]:
```
topSongs = generate_recommendation(spotify_data, playlist_vector, nonplaylist_df)
topSongs.head()
```