In [1]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-pyth
# For example, here's several helpful packages to load
import os
import numpy as np
import pandas as pd

import seaborn as sns
import plotly.express as px
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
from sklearn.metrics import euclidean_distances
from scipy.spatial.distance import cdist
from collections import defaultdict
import difflib

import warnings
warnings.filterwarnings("ignore")

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all file

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserv
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside o
```

In [2]:

```
import pandas as pd

data = pd.read_csv("data.csv")
artist_data = pd.read_csv("data_by_artist.csv")
genres_data = pd.read_csv("data_by_genres.csv")
year_data = pd.read_csv("data_by_year.csv")
w_genres_data = pd.read_csv("data_w_genres.csv")
```

In [3]:

```
data.head(2)
```

Out[3]:

	valence	year	acousticness	artists	danceability	duration_ms	energy	explicit
0	0.0594	1921	0.982	['Sergei Rachmaninoff', 'James Levine', 'Berli...	0.279	831667	0.211	0
1	0.9630	1921	0.732	['Dennis Day']	0.819	180533	0.341	0

In [4]:

```
data.shape
```

Out[4]:

```
(170653, 19)
```

In [5]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 170653 entries, 0 to 170652
Data columns (total 19 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   valence           170653 non-null  float64
 1   year              170653 non-null  int64  
 2   acousticness      170653 non-null  float64
 3   artists           170653 non-null  object 
 4   danceability      170653 non-null  float64
 5   duration_ms       170653 non-null  int64  
 6   energy            170653 non-null  float64
 7   explicit          170653 non-null  int64  
 8   id                170653 non-null  object 
 9   instrumentalness  170653 non-null  float64
 10  key               170653 non-null  int64  
 11  liveness          170653 non-null  float64
 12  loudness          170653 non-null  float64
 13  mode              170653 non-null  int64  
 14  name              170653 non-null  object 
 15  popularity        170653 non-null  int64  
 16  release_date      170653 non-null  object 
 17  speechiness       170653 non-null  float64
 18  tempo             170653 non-null  float64
dtypes: float64(9), int64(6), object(4)
memory usage: 24.7+ MB
```

In [6]:

```
artist_data.head(2)
```

Out[6]:

	mode	count	acousticness	artists	danceability	duration_ms	energy	instrumental
0	1	9	0.590111	"Cats" 1981 Original London Cast	0.467222	250318.555556	0.394003	0.01
1	1	26	0.862538	"Cats" 1983 Broadway Cast	0.441731	287280.000000	0.406808	0.08

In [7]:

```
artist_data.shape
```

Out[7]:

```
(28680, 15)
```

In [8]:

```
artist_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28680 entries, 0 to 28679
Data columns (total 15 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   mode              28680 non-null  int64  
 1   count             28680 non-null  int64  
 2   acousticness      28680 non-null  float64
 3   artists           28680 non-null  object 
 4   danceability      28680 non-null  float64
 5   duration_ms       28680 non-null  float64
 6   energy            28680 non-null  float64
 7   instrumentalness  28680 non-null  float64
 8   liveness          28680 non-null  float64
 9   loudness          28680 non-null  float64
 10  speechiness       28680 non-null  float64
 11  tempo             28680 non-null  float64
 12  valence           28680 non-null  float64
 13  popularity        28680 non-null  float64
 14  key               28680 non-null  int64  
dtypes: float64(11), int64(3), object(1)
memory usage: 3.3+ MB
```

In [9]:

```
genres_data.head(2)
```

Out[9]:

	mode	genres	acousticness	danceability	duration_ms	energy	instrumentalness	live
0	1	21st century classical	0.979333	0.162883	1.602977e+05	0.071317	0.606834	0
1	1	432hz	0.494780	0.299333	1.048887e+06	0.450678	0.477762	0

In [10]:

```
genres_data.shape
```

Out[10]:

```
(2973, 14)
```

In [11]:

```
genres_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2973 entries, 0 to 2972
Data columns (total 14 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   mode              2973 non-null   int64  
 1   genres            2973 non-null   object 
 2   acousticness      2973 non-null   float64
 3   danceability      2973 non-null   float64
 4   duration_ms       2973 non-null   float64
 5   energy            2973 non-null   float64
 6   instrumentalness  2973 non-null   float64
 7   liveness          2973 non-null   float64
 8   loudness          2973 non-null   float64
 9   speechiness       2973 non-null   float64
 10  tempo             2973 non-null   float64
 11  valence           2973 non-null   float64
 12  popularity        2973 non-null   float64
 13  key               2973 non-null   int64  
dtypes: float64(11), int64(2), object(1)
memory usage: 325.3+ KB
```

In [12]:

```
year_data.head(2)
```

Out[12]:

	mode	year	acousticness	danceability	duration_ms	energy	instrumentalness	livene
0	1	1921	0.886896	0.418597	260537.166667	0.231815	0.344878	0.205
1	1	1922	0.938592	0.482042	165469.746479	0.237815	0.434195	0.240

In [13]:

```
year_data.shape
```

Out[13]:

```
(100, 14)
```

In [14]:

```
year_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 14 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   mode              100 non-null    int64  
 1   year              100 non-null    int64  
 2   acousticness      100 non-null    float64
 3   danceability      100 non-null    float64
 4   duration_ms       100 non-null    float64
 5   energy            100 non-null    float64
 6   instrumentalness  100 non-null    float64
 7   liveness          100 non-null    float64
 8   loudness          100 non-null    float64
 9   speechiness       100 non-null    float64
 10  tempo             100 non-null    float64
 11  valence           100 non-null    float64
 12  popularity        100 non-null    float64
 13  key               100 non-null    int64  
dtypes: float64(11), int64(3)
memory usage: 11.1 KB
```

In [15]:

```
year_data.isnull().sum()
```

Out[15]:

```
mode                0
year                0
acousticness        0
danceability        0
duration_ms         0
energy              0
instrumentalness    0
liveness            0
loudness            0
speechiness         0
tempo               0
valence             0
popularity          0
key                 0
dtype: int64
```

In [16]:

```
import spotipy
from spotipy.oauth2 import SpotifyClientCredentials
from collections import defaultdict
from kaggle_secrets import UserSecretsClient
user_secrets = UserSecretsClient()
CLIENT_ID = user_secrets.get_secret("CLIENT_ID")
CLIENT_SECRET = user_secrets.get_secret("CLIENT_SECRET")

sp = spotipy.Spotify(auth_manager=SpotifyClientCredentials(client_id=CLIENT_ID,
                                                           client_secret=CLIENT_SECRET))
```

```
---------------------------------------------------------------------------
-
ModuleNotFoundError                       Traceback (most recent call las
t)
Cell In[16], line 4
      2 from spotipy.oauth2 import SpotifyClientCredentials
      3 from collections import defaultdict
----> 4 from kaggle_secrets import UserSecretsClient
      5 user_secrets = UserSecretsClient()
      6 CLIENT_ID = user_secrets.get_secret("CLIENT_ID")

ModuleNotFoundError: No module named 'kaggle_secrets'
```

In [17]:

```
import spotipy
from spotipy.oauth2 import SpotifyClientCredentials

# Replace 'YOUR_CLIENT_ID' and 'YOUR_CLIENT_SECRET' with your actual credentials
client_credentials_manager = SpotifyClientCredentials(client_id='YOUR_CLIENT_ID', client
sp = spotipy.Spotify(client_credentials_manager=client_credentials_manager)
```

In [18]:

```
import spotipy
from spotipy.oauth2 import SpotifyClientCredentials

# Initialize the Spotify API client
client_credentials_manager = SpotifyClientCredentials(client_id='YOUR_CLIENT_ID', client
sp = spotipy.Spotify(client_credentials_manager=client_credentials_manager)
```

In [19]:

```
'''
Finds song details from spotify dataset. If song is unavailable in dataset, it returns n
'''
def find_song(name, year):
    song_data = defaultdict()
    results = sp.search(q= 'track: {} year: {}'.format(name,year), limit=1)
    if results['tracks']['items'] == []:
        return None

    results = results['tracks']['items'][0]
    track_id = results['id']
    audio_features = sp.audio_features(track_id)[0]

    song_data['name'] = [name]
    song_data['year'] = [year]
    song_data['explicit'] = [int(results['explicit'])]
    song_data['duration_ms'] = [results['duration_ms']]
    song_data['popularity'] = [results['popularity']]

    for key, value in audio_features.items():
        song_data[key] = value

    return pd.DataFrame(song_data)
```

In [20]:

```
number_cols = ['valence', 'year', 'acousticness', 'danceability', 'duration_ms', 'energy
 'instrumentalness', 'key', 'liveness', 'loudness', 'mode', 'popularity', 'speechiness',
```

In [21]:

```
'''
Fetches song details from dataset. If info is unavailable in dataset, it will search det
'''
def get_song_data(song, spotify_data):
    try:
        song_data = spotify_data[(spotify_data['name'] == song['name'])
                                 & (spotify_data['year'] == song['year'])].iloc[0]
        print('Fetching song information from local dataset')
        return song_data

    except IndexError:
        print('Fetching song information from spotify dataset')
        return find_song(song['name'], song['year'])
```

In [22]:

```
'''
Fetches song info from dataset and does the mean of all numerical features of the song-d
'''
def get_mean_vector(song_list, spotify_data):
    song_vectors = []
    for song in song_list:
        song_data = get_song_data(song, spotify_data)
        if song_data is None:
            print('Warning: {} does not exist in Spotify or in database'.format(song['na
            continue
        song_vector = song_data[number_cols].values
        song_vectors.append(song_vector)

    song_matrix = np.array(list(song_vectors))#nd-array where n is number of songs in li
    #print(f'song_matrix {song_matrix}')
    return np.mean(song_matrix, axis=0) # mean of each ele in list, returns 1-d array
```

In [23]:

```
'''
Flattenning the dictionary by grouping the key and forming a list of values for respecti
'''
def flatten_dict_list(dict_list):
    flattened_dict = defaultdict()
    for key in dict_list[0].keys():
        flattened_dict[key] = [] # 'name', 'year'
    for dic in dict_list:
        for key,value in dic.items():
            flattened_dict[key].append(value) # creating list of values
    return flattened_dict
```

In [24]:

```
'''
Gets song list as input.
Get mean vectors of numerical features of the input.
Scale the mean-input as well as dataset numerical features.
calculate eculidean distance b/w mean-input and dataset.
Fetch the top 10 songs with maximum similarity.
'''
def recommend_songs( song_list, spotify_data, n_songs=10):

    metadata_cols = ['name', 'year', 'artists']
    song_dict = flatten_dict_list(song_list)

    song_center = get_mean_vector(song_list, spotify_data)
    #print(f'song_center {song_center}')
    scaler = song_cluster_pipeline.steps[0][1] # StandardScalar()
    scaled_data = scaler.transform(spotify_data[number_cols])
    scaled_song_center = scaler.transform(song_center.reshape(1, -1))
    distances = cdist(scaled_song_center, scaled_data, 'cosine')
    #print(f'distances {distances}')
    index = list(np.argsort(distances)[:, :n_songs][0])

    rec_songs = spotify_data.iloc[index]
    rec_songs = rec_songs[~rec_songs['name'].isin(song_dict['name'])]
    return rec_songs[metadata_cols].to_dict(orient='records')
```

In [25]:

```
recommend_songs([{'name': 'Blinding Lights', 'year': 2019}],  data)
```

```
      5 try:
----> 6     song_data = spotify_data[(spotify_data['name'] == song['nam
e'])
      7                             & (spotify_data['year'] == song['ye
ar'])].iloc[0]
      8     print('Fetching song information from local dataset')

File ~\anaconda3\lib\site-packages\pandas\core\indexing.py:1073, in _Lo
cationIndexer.__getitem__(self, key)
   1072 maybe_callable = com.apply_if_callable(key, self.obj)
-> 1073 return self._getitem_axis(maybe_callable, axis=axis)

File ~\anaconda3\lib\site-packages\pandas\core\indexing.py:1625, in _iL
ocIndexer._getitem_axis(self, key, axis)
   1624 # validate the location
-> 1625 self._validate_integer(key, axis)
   1627 return self.obj._ixs(key, axis=axis)

File ~\anaconda3\lib\site-packages\pandas\core\indexing.py:1557, in _iL
ocIndexer._validate_integer(self, key, axis)
```

In [26]:

```
recommend_songs([{'name': 'Fix You', 'year':2005}],  data)
```

```
Fetching song information from local dataset

---------------------------------------------------------------------------
-
NameError                                 Traceback (most recent call las
t)
Cell In[26], line 1
----> 1 recommend_songs([{'name': 'Fix You', 'year':2005}],  data)

Cell In[24], line 15, in recommend_songs(song_list, spotify_data, n_songs)
     13 song_center = get_mean_vector(song_list, spotify_data)
     14 #print(f'song_center {song_center}')
---> 15 scaler = song_cluster_pipeline.steps[0][1] # StandardScalar()
     16 scaled_data = scaler.transform(spotify_data[number_cols])
     17 scaled_song_center = scaler.transform(song_center.reshape(1, -1))

NameError: name 'song_cluster_pipeline' is not defined
```

In [27]:

```
recommend_songs([{'name': 'I Will Follow', 'year':2010},{'name': 'Come As You Are', 'yea
```

```
Fetching song information from local dataset
Fetching song information from local dataset

---------------------------------------------------------------------------
-
NameError                                 Traceback (most recent call las
t)
Cell In[27], line 1
----> 1 recommend_songs([{'name': 'I Will Follow', 'year':2010},{'name':
'Come As You Are', 'year':1991}],  data)

Cell In[24], line 15, in recommend_songs(song_list, spotify_data, n_songs)
     13 song_center = get_mean_vector(song_list, spotify_data)
     14 #print(f'song_center {song_center}')
---> 15 scaler = song_cluster_pipeline.steps[0][1] # StandardScalar()
     16 scaled_data = scaler.transform(spotify_data[number_cols])
     17 scaled_song_center = scaler.transform(song_center.reshape(1, -1))

NameError: name 'song_cluster_pipeline' is not defined
```

In [ ]:

```

```