Music Recommendation System Using Content based

In [4]:
```
# 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-python
# 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 files under the input direc

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 preserved as output when you c
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

In [6]:
```
data = pd.read_csv("data.csv")
genre_data = pd.read_csv('data_by_genres.csv')
year_data = pd.read_csv('data_by_year.csv')
artist_data = pd.read_csv('data_by_artist.csv')
```

In [8]:
```
data.head(2)
```

Out[8]:

	valence	year	acousticness	artists	danceability	duration_ms	energy	explicit	id	instrumentalness	key	l
0	0.0594	1921	0.982	['Sergei Rachmaninoff', 'James Levine', 'Berli...	0.279	831667	0.211	0	4BJqT0PrAfrxzMOxytFOIz	0.878	10	
1	0.9630	1921	0.732	['Dennis Day']	0.819	180533	0.341	0	7xPhfUan2yNtyFG0cUWkt8	0.000	7	

In [9]:
```
genre_data.head(2)
```

Out[9]:

	mode	genres	acousticness	danceability	duration_ms	energy	instrumentalness	liveness	loudness	speechiness	tempo	vale
0	1	21st century classical	0.979333	0.162883	1.602977e+05	0.071317	0.606834	0.3616	-31.514333	0.040567	75.336500	0.10
1	1	432hz	0.494780	0.299333	1.048887e+06	0.450678	0.477762	0.1310	-16.854000	0.076817	120.285667	0.22

In [10]:
```
year_data.head(2)
```

Out[10]:

	mode	year	acousticness	danceability	duration_ms	energy	instrumentalness	liveness	loudness	speechiness	tempo	valen
0	1	1921	0.886896	0.418597	260537.166667	0.231815	0.344878	0.20571	-17.048667	0.073662	101.531493	0.3793
1	1	1922	0.938592	0.482042	165469.746479	0.237815	0.434195	0.24072	-19.275282	0.116655	100.884521	0.5355

In [11]:
```
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 [15]:

```
data['decade'] = data['year'].apply(lambda year : f'{(year//10)*10}s' )
```

EDA(Exploratory Data Analysis)

In [27]:

```
sound_features = ['acousticness', 'danceability', 'energy', 'instrumentalness', 'liveness', 'valence']
fig = px.line(year_data, x='year', y=sound_features,title='Trend of various sound features over decades')
fig.show()
```

In [28]:

```
top10_genres = genre_data.nlargest(10, 'popularity')

fig = px.bar(top10_genres, x='genres', y=['valence', 'energy', 'danceability', 'acousticness'], barmode='group'
            title='Trend of various sound features over top 10 genres')
fig.show()
```

In [19]:

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

cluster_pipeline = Pipeline([('scaler', StandardScaler()), ('kmeans', KMeans(n_clusters=12))])
X = genre_data.select_dtypes(np.number)
cluster_pipeline.fit(X)
genre_data['cluster'] = cluster_pipeline.predict(X)
```

In [20]:

```
''' Visualizing the Clusters with t-SNE
 is an unsupervised Machine Learning algorithm.
 It has become widely used in bioinformatics and more generally in data science to visualise the structure of
 high dimensional data in 2 or 3 dimensions.
 While t-SNE is a dimensionality reduction technique, it is mostly used for visualization and not data pre-proc
 (like you might with PCA). For this reason, you almost always reduce the dimensionality down to 2 with t-SNE,
 so that you can then plot the data in two dimensions.
'''
from sklearn.manifold import TSNE

tsne_pipeline = Pipeline([('scaler', StandardScaler()), ('tsne', TSNE(n_components=2, verbose=1))])
genre_embedding = tsne_pipeline.fit_transform(X) # returns np-array of coordinates(x,y) for each record after T.
projection = pd.DataFrame(columns=['x', 'y'], data=genre_embedding)
projection['genres'] = genre_data['genres']
projection['cluster'] = genre_data['cluster']

fig = px.scatter(
    projection, x='x', y='y', color='cluster', hover_data=['x', 'y', 'genres'],title='Clusters of genres')
fig.show()
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 2973 samples in 0.000s...
[t-SNE] Computed neighbors for 2973 samples in 0.360s...
[t-SNE] Computed conditional probabilities for sample 1000 / 2973
[t-SNE] Computed conditional probabilities for sample 2000 / 2973
[t-SNE] Computed conditional probabilities for sample 2973 / 2973
[t-SNE] Mean sigma: 0.777516
[t-SNE] KL divergence after 250 iterations with early exaggeration: 76.106194
[t-SNE] KL divergence after 1000 iterations: 1.392478
```

Visualizing the Clusters with PCA

Prinipal Component Analysis is an unsupervised learning algorithm that is used for the dimensionality reduction in machine learning. One of the most major differences between PCA and t-SNE is it preserves only local similarities whereas PA preserves large pairwise distance maximize variance. It takes a set of points in high dimensional data and converts it into low dimensional data. '''

In [21]:
```
song_cluster_pipeline = Pipeline([('scaler', StandardScaler()),
                                  ('kmeans', KMeans(n_clusters=25,
                                   verbose=False))
                                 ], verbose=False)

X = data.select_dtypes(np.number)
song_cluster_pipeline.fit(X)
song_cluster_labels = song_cluster_pipeline.predict(X)
data['cluster_label'] = song_cluster_labels
```

In [22]:
```
from sklearn.decomposition import PCA

pca_pipeline = Pipeline([('scaler', StandardScaler()), ('PCA', PCA(n_components=2))])
song_embedding = pca_pipeline.fit_transform(X)
projection = pd.DataFrame(columns=['x', 'y'], data=song_embedding)
projection['title'] = data['name']
projection['cluster'] = data['cluster_label']

fig = px.scatter(
    projection, x='x', y='y', color='cluster', hover_data=['x', 'y', 'title'],title='Clusters of songs')
fig.show()
```

In [24]:

```
!pip install spotipy
```

```
Collecting spotipy
  Obtaining dependency information for spotipy from https://files.pythonhosted.org/packages/b8/e8/4c099f9431ec9
a86f576b344702cd4446d1ff7df09b172dc1951f25d58b1/spotipy-2.23.0-py3-none-any.whl.metadata
  Downloading spotipy-2.23.0-py3-none-any.whl.metadata (3.3 kB)
Collecting redis>=3.5.3 (from spotipy)
  Obtaining dependency information for redis>=3.5.3 from https://files.pythonhosted.org/packages/65/f2/540ad079
10732733138beb192991c67c69e7f6ebf549ce1a3a77846cbae7/redis-5.0.4-py3-none-any.whl.metadata
  Downloading redis-5.0.4-py3-none-any.whl.metadata (9.3 kB)
Requirement already satisfied: requests>=2.25.0 in c:\users\haris\anaconda3\lib\site-packages (from spotipy) (2
.31.0)
Requirement already satisfied: six>=1.15.0 in c:\users\haris\appdata\roaming\python\python311\site-packages (fr
om spotipy) (1.16.0)
Requirement already satisfied: urllib3>=1.26.0 in c:\users\haris\anaconda3\lib\site-packages (from spotipy) (1.
26.16)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\haris\anaconda3\lib\site-packages (from req
uests>=2.25.0->spotipy) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\haris\anaconda3\lib\site-packages (from requests>=2.25.
0->spotipy) (3.4)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\haris\anaconda3\lib\site-packages (from requests>
=2.25.0->spotipy) (2023.7.22)
Downloading spotipy-2.23.0-py3-none-any.whl (29 kB)
Downloading redis-5.0.4-py3-none-any.whl (251 kB)
   ---------------------------------------- 0.0/252.0 kB ? eta -:--:--
   ------ -------------------------------- 41.0/252.0 kB 960.0 kB/s eta 0:00:01
   ----------------- ---------------------- 112.6/252.0 kB 1.3 MB/s eta 0:00:01
   -------------------------- ------------- 163.8/252.0 kB 1.2 MB/s eta 0:00:01
   ---------------------------------------  245.8/252.0 kB 1.4 MB/s eta 0:00:01
   ---------------------------------------- 252.0/252.0 kB 1.2 MB/s eta 0:00:00
Installing collected packages: redis, spotipy
Successfully installed redis-5.0.4 spotipy-2.23.0
```

In [25]:

```
!pip install kaggle
```

```
Collecting kaggle
  Downloading kaggle-1.6.12.tar.gz (79 kB)
     ---------------------------------------- 0.0/79.7 kB ? eta -:--:--
     -------------------- ------------------- 41.0/79.7 kB ? eta -:--:--
     ---------------------------------------- 79.7/79.7 kB 1.1 MB/s eta 0:00:00
  Preparing metadata (setup.py): started
  Preparing metadata (setup.py): finished with status 'done'
Requirement already satisfied: six>=1.10 in c:\users\haris\appdata\roaming\python\python311\site-packages (from
kaggle) (1.16.0)
Requirement already satisfied: certifi>=2023.7.22 in c:\users\haris\anaconda3\lib\site-packages (from kaggle) (
2023.7.22)
Requirement already satisfied: python-dateutil in c:\users\haris\appdata\roaming\python\python311\site-packages
(from kaggle) (2.8.2)
Requirement already satisfied: requests in c:\users\haris\anaconda3\lib\site-packages (from kaggle) (2.31.0)
Requirement already satisfied: tqdm in c:\users\haris\anaconda3\lib\site-packages (from kaggle) (4.65.0)
Requirement already satisfied: python-slugify in c:\users\haris\anaconda3\lib\site-packages (from kaggle) (5.0.
2)
Requirement already satisfied: urllib3 in c:\users\haris\anaconda3\lib\site-packages (from kaggle) (1.26.16)
Requirement already satisfied: bleach in c:\users\haris\anaconda3\lib\site-packages (from kaggle) (4.1.0)
Requirement already satisfied: packaging in c:\users\haris\appdata\roaming\python\python311\site-packages (from
bleach->kaggle) (23.0)
Requirement already satisfied: webencodings in c:\users\haris\anaconda3\lib\site-packages (from bleach->kaggle)
(0.5.1)
Requirement already satisfied: text-unidecode>=1.3 in c:\users\haris\anaconda3\lib\site-packages (from python-s
lugify->kaggle) (1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\haris\anaconda3\lib\site-packages (from req
uests->kaggle) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\haris\anaconda3\lib\site-packages (from requests->kaggl
e) (3.4)
Requirement already satisfied: colorama in c:\users\haris\anaconda3\lib\site-packages (from tqdm->kaggle) (0.4.
6)
Building wheels for collected packages: kaggle
  Building wheel for kaggle (setup.py): started
  Building wheel for kaggle (setup.py): finished with status 'done'
  Created wheel for kaggle: filename=kaggle-1.6.12-py3-none-any.whl size=102984 sha256=e0338659ed4a869af0abc449
9942966ae2551f0c3973ce141c19e098a68747b7
  Stored in directory: c:\users\haris\appdata\local\pip\cache\wheels\f3\eb\e9\819c2d9eac90204eec8579430759f75a1
d6dbe4cd0b93f53bc
Successfully built kaggle
Installing collected packages: kaggle
Successfully installed kaggle-1.6.12
```

In [ ]:

```
import spotipy
from spotipy.oauth2 import SpotifyClientCredentials
from collections import defaultdict

sp = spotipy.Spotify(auth_manager=SpotifyClientCredentials(client_id=os.environ["SPOTIFY_CLIENT_ID"],
                                                           client_secret=os.environ["SPOTIFY_CLIENT_SECRET"]))

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 [ ]:

```
'''
Finds song details from spotify dataset. If song is unavailable in dataset, it returns none.
'''
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']]
```

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

    return pd.DataFrame(song_data)
```

In [ ]:

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

In [ ]:

```python
'''
Fetches song info from dataset and does the mean of all numerical features of the song-data.
'''
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['name']))
            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 list. It contains all num
    #print(f'song_matrix {song_matrix}')
    return np.mean(song_matrix, axis=0) # mean of each ele in list, returns 1-d array
```

In [ ]:

```python
'''
Flattenning the dictionary by grouping the key and forming a list of values for respective key.
'''
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 [ ]:

```python
'''
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 [ ]:

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

[{'name': 'Best News Ever', 'year': 2017, 'artists': "['MercyMe']"}, {'name': 'Bit By Bit', 'year': 2012, 'artists': "['Mother Mother']"},

{'name': 'A Different World (feat. Corey Taylor)',

'year': 2016,'artists': "['Korn', 'Corey Taylor']"},

{'name': 'Sober', 'year': 2015, 'artists': "['Selena Gomez']"},

{'name': "Don't Say Goodnight", 'year': 2014, 'artists': "['Hot Chelle Rae']"},

{'name': 'Sight of the Sun - Single Version', 'year': 2014, 'artists': "['fun.']"},

{'name': 'This Life', 'year': 2011, 'artists': "['Curtis Stigers', 'The Forest Rangers']"},

{'name': 'Re-Do', 'year': 2012, 'artists': "['Modern Baseball']"}, {'name': 'I Will Follow', 'year': 2010, 'artists': "['Chris Tomlin']"},

{'name': "Livin' The Dream", 'year': 2016, 'artists': "['Drake White']"}]