Spotify Recommended System

October 2, 2024

Importing Libraries

[1]:
```
import os
import numpy as np
import pandas as pd

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

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

import spotipy
from spotipy.oauth2 import SpotifyClientCredentials
from spotipy.oauth2 import SpotifyOAuth
```

Quick look at the Dataset

[2]:
```
data = pd.read_csv("Ruru/data.csv")
data.head(5)
```

[2]:
```
   valence  year  acousticness  \
0   0.0594  1921         0.982
1   0.9630  1921         0.732
2   0.0394  1921         0.961
3   0.1650  1921         0.967
4   0.2530  1921         0.957

                                              artists  danceability  \
0  ['Sergei Rachmaninoff', 'James Levine', 'Berli…          0.279
1                                      ['Dennis Day']         0.819
2  ['KHP Kridhamardawa Karaton Ngayogyakarta Hadi…          0.328
3                                    ['Frank Parker']         0.275
```

```
4                                     ['Phil Regan']        0.418

   duration_ms  energy  explicit                      id  instrumentalness  \
0       831667   0.211         0  4BJqT0PrAfrxzMOxytFOIz          0.878000
1       180533   0.341         0  7xPhfUan2yNtyFG0cUWkt8          0.000000
2       500062   0.166         0  1o6I8BglA6ylDMrIELygv1          0.913000
3       210000   0.309         0  3ftBPsC5vPBKxYSee08FDH          0.000028
4       166693   0.193         0  4d6HGyGT8e121BsdKmw9v6          0.000002

   key  liveness  loudness  mode  \
0   10     0.665   -20.096     1
1    7     0.160   -12.441     1
2    3     0.101   -14.850     1
3    5     0.381    -9.316     1
4    3     0.229   -10.096     1

                                                name  popularity release_date  \
0  Piano Concerto No. 3 in D Minor, Op. 30: III. …           4         1921
1                            Clancy Lowered the Boom           5         1921
2                                          Gati Bali           5         1921
3                                          Danny Boy           3         1921
4                        When Irish Eyes Are Smiling           2         1921

   speechiness    tempo
0       0.0366   80.954
1       0.4150   60.936
2       0.0339  110.339
3       0.0354  100.109
4       0.0380  101.665
```

[3]:
```
genre = pd.read_csv("Ruru/data_by_genres.csv")
genre.head(5)
```

[3]:
```
   mode                  genres  acousticness  danceability   duration_ms  \
0     1  21st century classical      0.979333      0.162883  1.602977e+05
1     1                   432hz      0.494780      0.299333  1.048887e+06
2     1                   8-bit      0.762000      0.712000  1.151770e+05
3     1                      []      0.651417      0.529093  2.328809e+05
4     1              a cappella      0.676557      0.538961  1.906285e+05

     energy  instrumentalness  liveness   loudness  speechiness       tempo  \
0  0.071317          0.606834  0.361600 -31.514333     0.040567   75.336500
1  0.450678          0.477762  0.131000 -16.854000     0.076817  120.285667
2  0.818000          0.876000  0.126000  -9.180000     0.047000  133.444000
3  0.419146          0.205309  0.218696 -12.288965     0.107872  112.857352
4  0.316434          0.003003  0.172254 -12.479387     0.082851  112.110362
```

```
    valence  popularity  key
0  0.103783   27.833333    6
1  0.221750   52.500000    5
2  0.975000   48.000000    7
3  0.513604   20.859882    7
4  0.448249   45.820071    7
```

[4]:

```python
year = pd.read_csv("Ruru/data_by_year.csv")
year.head(5)
```

[4]:

```
   mode  year  acousticness  danceability    duration_ms    energy  \
0     1  1921      0.886896      0.418597  260537.166667  0.231815
1     1  1922      0.938592      0.482042  165469.746479  0.237815
2     1  1923      0.957247      0.577341  177942.362162  0.262406
3     1  1924      0.940200      0.549894  191046.707627  0.344347
4     1  1925      0.962607      0.573863  184986.924460  0.278594

   instrumentalness  liveness   loudness  speechiness       tempo   valence  \
0          0.344878  0.205710 -17.048667     0.073662  101.531493  0.379327
1          0.434195  0.240720 -19.275282     0.116655  100.884521  0.535549
2          0.371733  0.227462 -14.129211     0.093949  114.010730  0.625492
3          0.581701  0.235219 -14.231343     0.092089  120.689572  0.663725
4          0.418297  0.237668 -14.146414     0.111918  115.521921  0.621929

   popularity  key
0    0.653333    2
1    0.140845   10
2    5.389189    0
3    0.661017   10
4    2.604317    5
```

[5]:

```python
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
```

[6]:

```
genre.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
```

[7]:

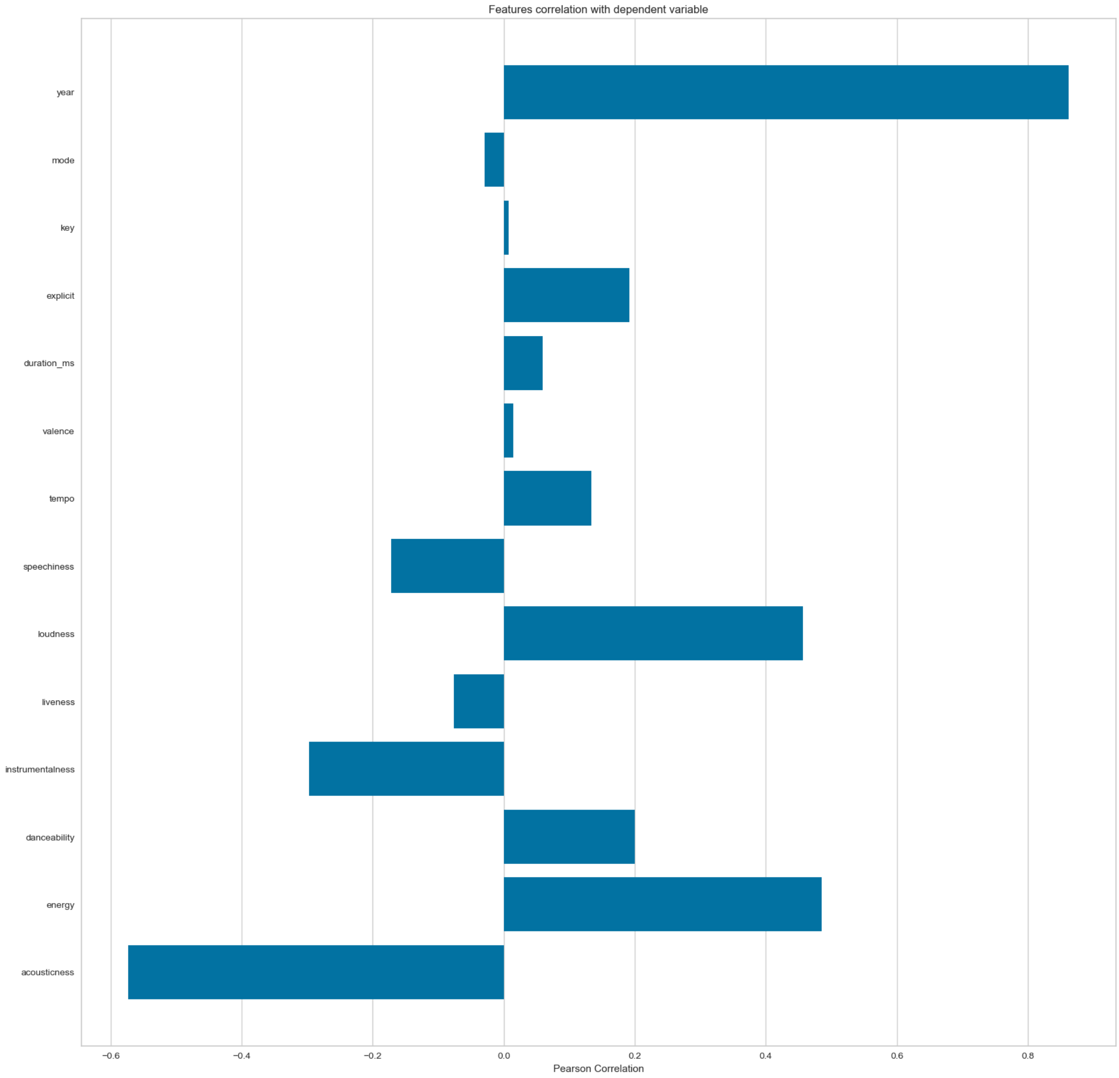
```
year.info()
```

```
<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
```

Visualization of Dataset

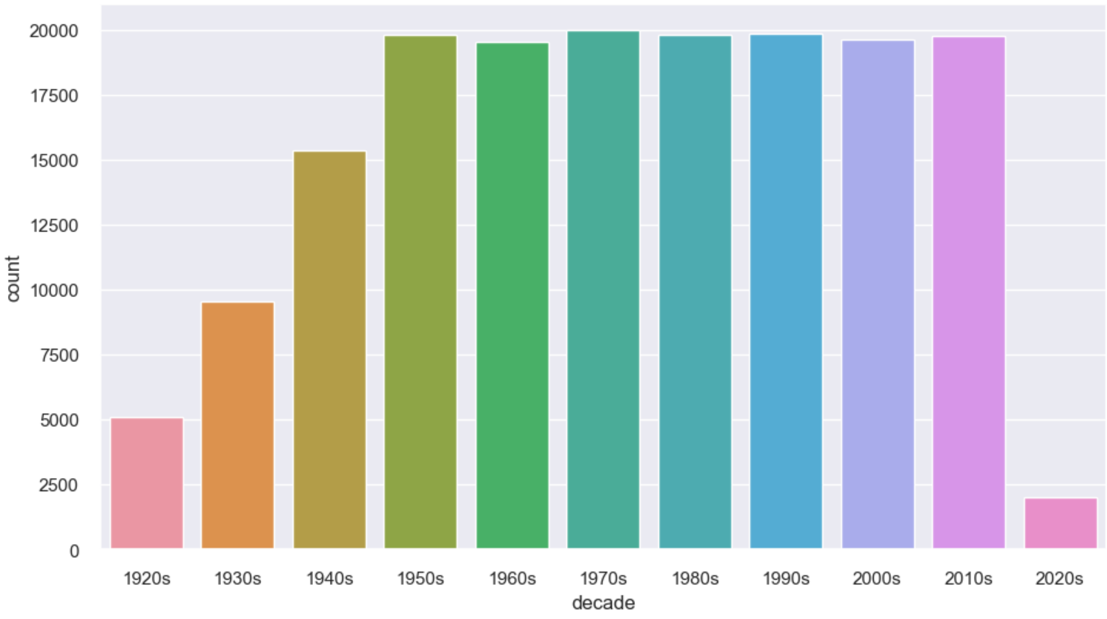
[8]:

```
from yellowbrick.target import FeatureCorrelation

feature_names =␣
 ↪['acousticness','energy','danceability','instrumentalness','liveness',
                ␣
 ↪'loudness','speechiness','tempo','valence','duration_ms','explicit','key','mode','year']
X,y = data[feature_names],data['popularity']

features = np.array(feature_names)

visualizer = FeatureCorrelation(labels=features)

plt.rcParams['figure.figsize']=(20,20)
visualizer.fit(X,y)
visualizer.show()
```



```
[8]: <Axes: title={'center': 'Features correlation with dependent variable'},
     xlabel='Pearson Correlation'>
```

```
[10]: def get_decade(year):
          period = int(year/10)*10
          decade = '{}s'.format(period)
          return decade
      data['decade'] = data['year'].apply(get_decade)

      sns.set(rc={'figure.figsize':(11,6)})
      sns.countplot(x='decade',data=data)
      plt.show()
```



[9]:
```
sound_features =␣
  ↪['acousticness','danceability','energy','instrumentalness','liveness','valence']
fig=px.line(year,x='year',y=sound_features)
fig.show()
```

[12]:
```
top_genres = genre.nlargest(10, 'popularity')

fig=px.
  ↪bar(top_genres,x='genres',y=['valence','energy','danceability','acousticness'],barmode='gro
fig.show()
```

Importing KMeans Libraries

[10]:
```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

cluster_pipeline = Pipeline([('scaler',␣
  ↪StandardScaler()),('kmeans',KMeans(n_clusters=10,random_state=42,n_init='auto',algorithm='e
X = genre.select_dtypes(np.number)
cluster_pipeline.fit(X)
genre['cluster'] = cluster_pipeline.predict(X)
```

[ ]:
```
Visualization of the Data using Kmean
```

[11]:
```
from sklearn.manifold import TSNE
```

```
tsne_pipeline = Pipeline([('scaler', StandardScaler()),('tsne', TSNE(n_components=2, verbose=1))])
genre_embedding = tsne_pipeline.fit_transform(X)
projection = pd.DataFrame(columns=['x','y'], data=genre_embedding)
projection['genres'] = genre['genres']
projection['cluster'] = genre['cluster']

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

```
[t-SNE] Computing 91 nearest neighbors…
[t-SNE] Indexed 2973 samples in 0.027s…
[t-SNE] Computed neighbors for 2973 samples in 0.358s…
[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: 76.105965
[t-SNE] KL divergence after 1000 iterations: 1.393252
```

[12]:

```
cluster_pipeline = Pipeline([('scaler', StandardScaler()), ('kmeans', KMeans(n_clusters=20, verbose=False, n_init=10))],verbose=False)
X = data.select_dtypes(include=np.number)
number_cols = list(X.columns)
cluster_pipeline.fit(X)
cluster_labels=cluster_pipeline.predict(X)
data['cluster_label'] = cluster_labels
```

[13]:

```
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'])
fig.show()
```

[17]:

```
pip install python-dotenv
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: python-dotenv in
c:\programdata\anaconda3\lib\site-packages (0.21.0)
```

```
Note: you may need to restart the kernel to use updated packages.
```

importing Spotify API and Writing function for music recommendation

[34]:
```
import spotipy
from dotenv import load_dotenv
from spotipy.oauth2 import SpotifyClientCredentials
from collections import defaultdict

load_dotenv()

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

def find_music(name):
    music_data=defualtdict()
    redult=sp.search(q = 'track: {}'.format(name),limit=1)
    if  results['tracks']['items']==[]:
        return None
    results = results['items']['tracks'][0]
    track_id = results['id']
    audio_features = sp.audio_features(track_id)[0]

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

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

    #return pd.DataFrame(song_data)
    return pd.DataFrame(music_data)
```

[35]:
```
from collections import defaultdict
from sklearn.metrics.pairwise import euclidean_distances
from scipy.spatial.distance import cdist
import difflib

number_cols =␣
 ↪['valence','year','acousticness','danceability','duration_ms','energy','explicit','instrume

def get_music(song,spotify_data):
    try:
        music_data = spotify_data[(spotify_data['name'] == song['name'])
```

```
                                 &(spotify_data['year'] == song['year'])].
↪iloc[0]
        return music_data

    except IndexError:
        return find_song(song['name'],song['year'])

def mean_vector(song_list,spotify_data):

    song_vectors =  []

    for song in song_list:
        music_data = get_music(song,spotify_data)
        if music_data is None:
           print('Warning : {} does Not exit in Spotify or database'.
↪format(song['name']))
           continue
        music_vector =  music_data[number_cols].values
        music_vector.append(music_vector)

    music_matrix = np.array(list(music_vectors))
    return np.mean(music_matrix, axis=0)
```

[36]:

```
def flatten_dict_list(dict_list):
    flattened_dict = defaultdict(list)
    for dictionary in dict_list:
        for key, value in dictionary.items():
            flattened_dict[key].append(value)
    return flattened_dict

def mean_vector(song_list, spotify_data):
    music_vectors = []
    for song in song_list:
        music_data = spotify_data[(spotify_data['name'].str.lower() ==␣
↪song['name'].lower())]
        if music_data.empty:
            continue
        music_vector = music_data[number_cols].values[0]  # Extract feature␣
↪vector
        music_vectors.append(music_vector)
    music_matrix = np.array(music_vectors)
    return np.mean(music_matrix, axis=0)



def recommend_music(spotify_data, song_title=None, artist=None, year=None,␣
↪genre=None, n_songs=10):
    filtered_data = spotify_data.copy()

    if song_title and isinstance(song_title, str):
        filtered_data = filtered_data[filtered_data['name'].str.lower() ==␣
↪song_title.lower()]

    if artist and isinstance(artist, str):
        filtered_data = filtered_data[filtered_data['artists'].apply(lambda x:␣
↪artist.lower() in [a.lower() for a in eval(x)])]

    if year and isinstance(year, int):
        filtered_data = filtered_data[filtered_data['year'] == year]

    if genre and isinstance(genre, str) and 'genre' in filtered_data.columns:
        filtered_data = filtered_data[filtered_data['genre'].str.lower() ==␣
↪genre.lower()]

    if filtered_data.empty:
        return []

    # Select only the numeric features used during training
    numeric_features = filtered_data[number_cols]  # Assuming `number_cols` is␣
↪a list of features used during fitting

    scaler = cluster_pipeline.steps[0][1]  # Assuming scaler is from your␣
↪pipeline
    scaled_data = scaler.transform(numeric_features)
    cluster_labels = cluster_pipeline.steps[1][1].predict(scaled_data)

    recommendations = filtered_data.copy()
    recommendations['cluster'] = cluster_labels
    chosen_cluster = cluster_labels[0]  # Choose the cluster of the first match

    recommendations = recommendations[recommendations['cluster'] ==␣
↪chosen_cluster]

    recommendations = recommendations.sort_values('popularity',␣
↪ascending=False).head(n_songs)

    columns_to_return = ['name', 'artists', 'year', 'popularity']
    if 'genre' in recommendations.columns:
        columns_to_return.append('genre')

    return recommendations[columns_to_return].to_dict(orient='records')
```

Importing HTML and Display for final recommendation of music

[37]:

```python
from IPython.display import display, HTML

def display_recommendations(recommendations):
    html_content = """
    <div style="background-color:#191414; color: white; padding: 20px; font-family: 'Arial', sans-serif; border-radius: 10px;">
        <h2 style="text-align: center; color: #1DB954;">Spotify Song Recommendations</h2>
        <ul style="list-style-type: none; padding: 0;">
    """
    for song in recommendations:
        html_content += f"""
        <li style='margin: 10px 0; padding: 15px; background-color: #282828; border-radius: 8px; display: flex; align-items: center;'>
            <div style='flex-grow: 1;'>
                <strong style='font-size: 18px;'>{song['name']}</strong>
                <span style='color: #b3b3b3;'>by {song['artists']}</span>
            </div>
            <div style='text-align: right;'>
                <span style='color: #1DB954; font-size: 12px;'>Spotify</span>
            </div>
        </li>
        """
    html_content += "</ul></div>"


    display(HTML(html_content))


recommended_songs = recommend_music(data, song_title='', artist='eminem')#(,year=)
display_recommendations(recommended_songs)
```

```
<IPython.core.display.HTML object>
```

[ ]:

```python

```