Libraries

```
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 warnings
warnings.filterwarnings("ignore", category=FutureWarning)
```

Datasets

```
data = pd.read_csv("data.csv")
genre_data = pd.read_csv("data_by_genres.csv")
year_data = pd.read_csv("data_by_year.csv")

print(data.info())
```

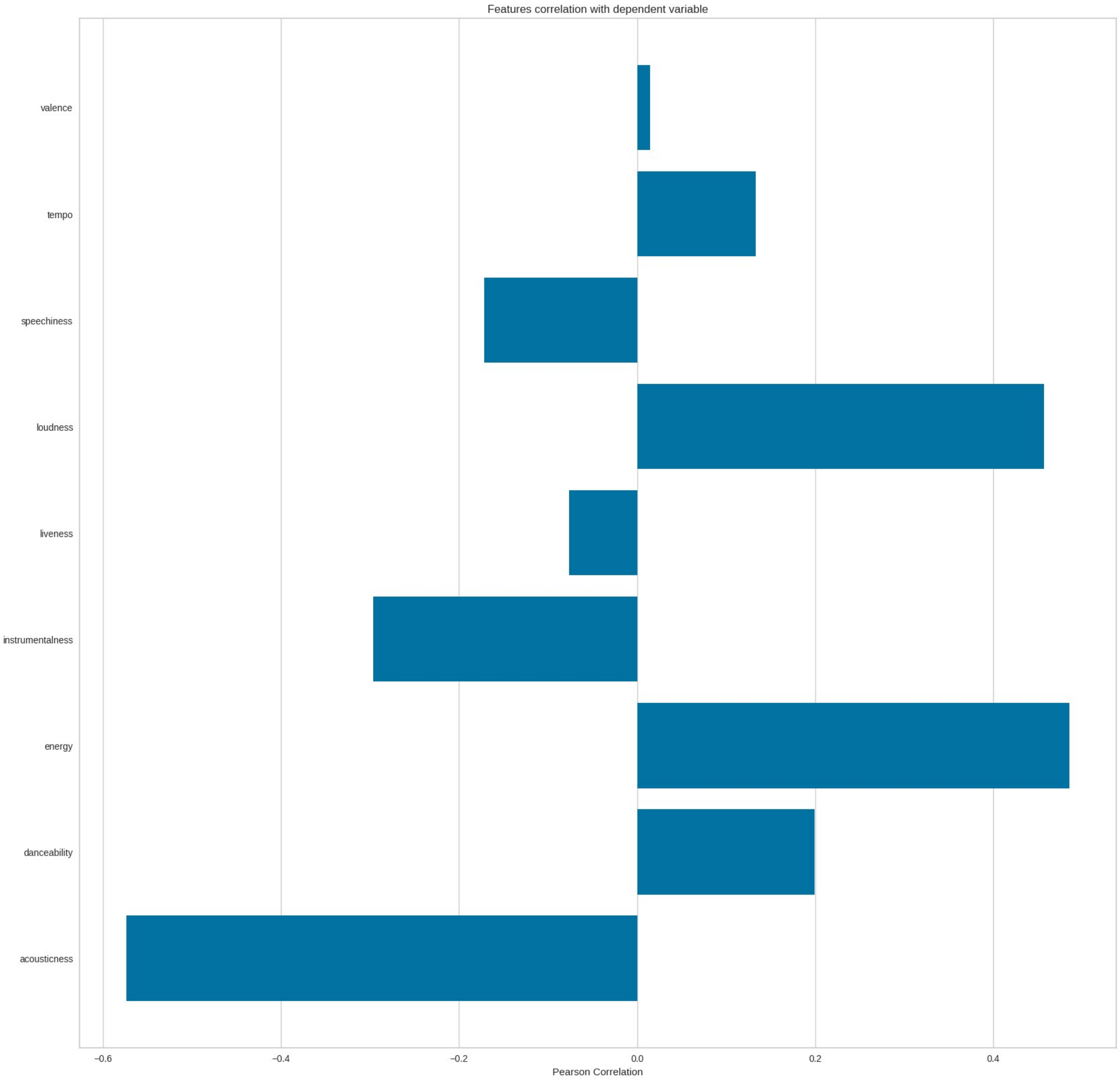
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 170653 entries, 0 to 170652
Data columns (total 19 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   valence           170653 non-null  float64
 1   year              170653 non-null  int64
 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
None
```

```python
print(genre_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
None
```

```python
print(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
None

from yellowbrick.target import FeatureCorrelation
features_names = ['acousticness', 'danceability', 'energy',
'instrumentalness', 'liveness', 'loudness', 'speechiness', 'tempo',
'valence']

# Assign features_names to features before using it
features = features_names

X, y = data[features], data['popularity']

#list of feature names - This line is redundant now
# features = np.array(features_names)

# Visualizer
visualizer = FeatureCorrelation(labels=features)

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



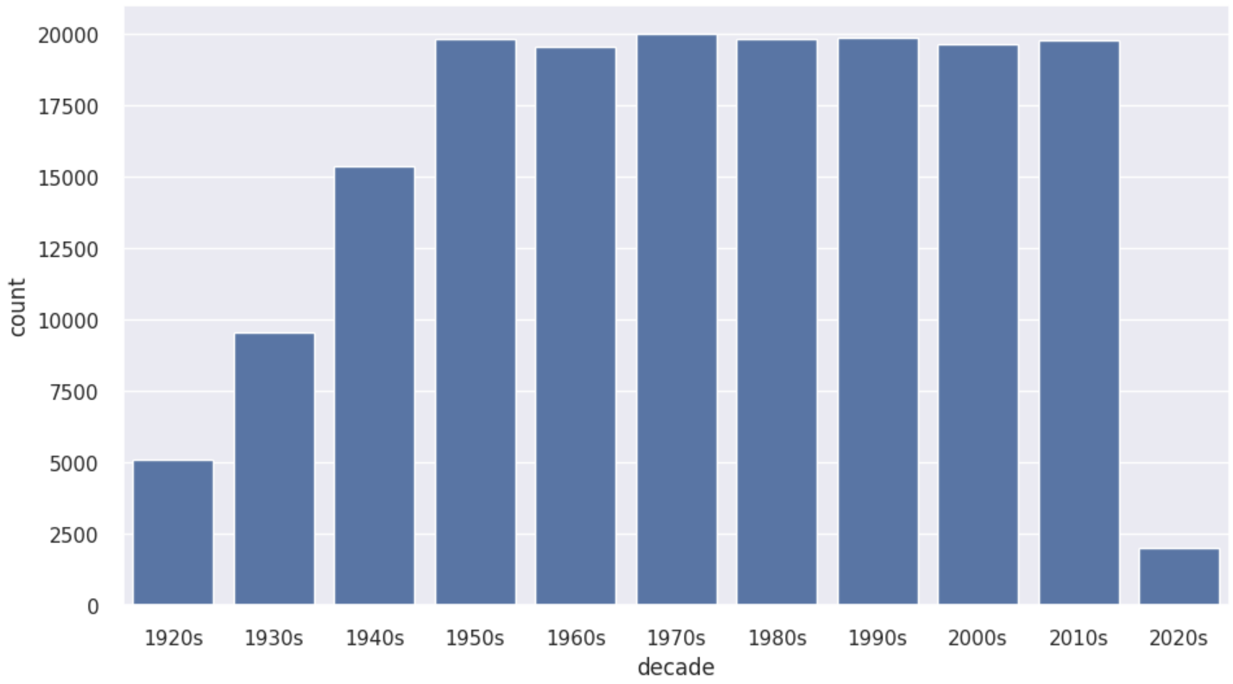
```
<Axes: title={'center': 'Features correlation with dependent
variable'}, xlabel='Pearson Correlation'>
```

Visualize

```
def get_decade(year):
  period_start = int(year/10) * 10
  decade = '{}s'.format(period_start)
  return decade

data['decade'] = data['year'].apply(get_decade)
```

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

```
<Axes: xlabel='decade', ylabel='count'>
```



```python
sound_featrues = ['acousticness', 'danceability', 'energy',
'instrumentalness', 'liveness', 'valence']
fig = px.line(year_data, x='year', y=sound_featrues)
fig.show()
```

Different Genres

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

fig = px.bar(top10_genres, x='genres', y=['valence', 'energy',
'danceability', 'acousticness'], barmode='group')
fig.show()
```

Cluster with KMeans

divide the generes into ten clustersbased on numerical audio features

```python
# import KMeans, StandardScaler, and Pipeline

cluster_pipline = Pipeline([('scaler', StandardScaler()), ('kmeans',
KMeans(n_clusters=10))])
X = genre_data.select_dtypes(np.number)
```

```
cluster_pipline.fit(X)
genre_data['cluster'] = cluster_pipline.predict(X)

#Visualize Genre cluster with t-SNE

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_data['genres']
projection['cluster'] = genre_data['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 0.005s...
[t-SNE] Computed neighbors for 2973 samples in 0.312s...
[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:
76.102287
[t-SNE] KL divergence after 1000 iterations: 1.394000
```

```
#Clustering Songs with KMeans

song_cluster_pipeline = Pipeline([('scaler', StandardScaler()),
                                  ('kmeans', KMeans(n_clusters=20,
                                   verbose=False))
                                 ], verbose=False)

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

# Visualize the CLusters in PCA

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()
```

Recommender System

```
!pip install spotipy
```

```
Collecting spotipy
  Downloading spotipy-2.25.1-py3-none-any.whl.metadata (5.1 kB)
Collecting redis>=3.5.3 (from spotipy)
  Downloading redis-7.0.1-py3-none-any.whl.metadata (12 kB)
Requirement already satisfied: requests>=2.25.0 in
/usr/local/lib/python3.12/dist-packages (from spotipy) (2.32.4)
Requirement already satisfied: urllib3>=1.26.0 in
/usr/local/lib/python3.12/dist-packages (from spotipy) (2.5.0)
Requirement already satisfied: charset_normalizer<4,>=2 in
/usr/local/lib/python3.12/dist-packages (from requests>=2.25.0-
>spotipy) (3.4.4)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.12/dist-packages (from requests>=2.25.0-
>spotipy) (3.11)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.12/dist-packages (from requests>=2.25.0-
>spotipy) (2025.10.5)
Downloading spotipy-2.25.1-py3-none-any.whl (31 kB)
Downloading redis-7.0.1-py3-none-any.whl (339 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0.0/339.9 kB ? eta -:--:--
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 339.9/339.9 kB 23.6 MB/s eta
0:00:00
```

```
import os
import spotipy
from spotipy.oauth2 import SpotifyClientCredentials

# Update environment variables with your credentials
os.environ['SPOTIPY_CLIENT_ID'] = 'YOUR_CLIENT_ID'
os.environ['SPOTIPY_CLIENT_SECRET'] = 'YOUR_CLIENT_SECRET'

# Reconnect to Spotipy with updated credentials
auth_manager = SpotifyClientCredentials()
sp = spotipy.Spotify(auth_manager=auth_manager)


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]

    for col in numbers_cols:
      song_data[col] = [None]

    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)

from collections import defaultdict
from sklearn.metrics import euclidean_distances
from scipy.spatial.distance import cdist
import difflib

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


def get_song_data(song, spotify_data):
    try:
        song_data = spotify_data[(spotify_data['name'] ==
song['name']) & (spotify_data['year'] == song['year'])]
        if song_data.empty:
            print(f"Warning: Song '{song['name']}' from {song['year']}
not found in dataset.")
            return None  # Or raise an exception if preferred
        else:
            return song_data.iloc[0]

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


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)

    if not song_vectors:  # Check if song_vectors is empty
        return None

    song_matrix = np.array(list(song_vectors))
    return np.mean(song_matrix, axis=0)


def flatten_dict_list(dict_list):

    flattened_dict = defaultdict()
    for key in dict_list[0].keys():
        flattened_dict[key] = []

    for dictionary in dict_list:
        for key, value in dictionary.items():
            flattened_dict[key].append(value)

    return flattened_dict


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)

    if song_center is None: # Handle the case where no songs are found
        return []


    scaler = song_cluster_pipeline.steps[0][1]
    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')
    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')
```

```python
recommend_songs([{'name': 'Come As You Are', 'year':1991},
                {'name': 'Smells Like Teen Spirit', 'year': 1991},
                {'name': 'Lithium', 'year': 1992},
                {'name': 'All Apologies', 'year': 1993},
                {'name': 'Stay Away', 'year': 1993}],  data)
```

```
Warning: Song 'Stay Away' from 1993 not found in dataset.
Warning: Stay Away does not exist in Spotify or in database
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/utils/
validation.py:2739: UserWarning:

X does not have valid feature names, but StandardScaler was fitted
with feature names
```

```
[{'name': 'Hanging By A Moment', 'year': 2000, 'artists':
"['Lifehouse']"},
 {'name': 'Kiss Me', 'year': 1997, 'artists': "['Sixpence None The
Richer']"},
 {'name': "Breakfast At Tiffany's",
  'year': 1995,
  'artists': "['Deep Blue Something']"},
 {'name': 'Otherside', 'year': 1999, 'artists': "['Red Hot Chili
Peppers']"},
 {'name': "It's Not Living (If It's Not With You)",
  'year': 2018,
  'artists': "['The 1975']"},
 {'name': 'No Excuses', 'year': 1994, 'artists': "['Alice In
Chains']"},
 {'name': 'Wherever You Will Go', 'year': 2001, 'artists': "['The
Calling']"},
 {'name': 'Ballbreaker', 'year': 1995, 'artists': "['AC/DC']"},
 {'name': 'Runaway (U & I)', 'year': 2015, 'artists': "['Galantis']"},
 {'name': "Club Can't Handle Me (feat. David Guetta)",
  'year': 2010,
  'artists': "['Flo Rida', 'David Guetta']"}]
```

```python
def get_song_input():
  """Gets song name and year input from the user."""

  song_list = []
  while True:
    song_name = input("Enter song name (or type 'done' to finish): ")
    if song_name.lower() == 'done':
      break

    try:
      song_year = int(input("Enter song year: "))
    except ValueError:
      print("Invalid year. Please enter a number.")
      continue

    song_list.append({'name': song_name, 'year': song_year})

  return song_list


def recommend_songs_interactive(data):
  """Gets song input from the user and provides recommendations."""

  song_list = get_song_input()
  if song_list:
    recommendations = recommend_songs(song_list, data)
    if recommendations:
      print("\nHere are some song recommendations:")
      for song in recommendations:
        print(f"- {song['name']} ({song['year']}) by
{song['artists']}")
    else:
      print("No recommendations found based on your input.")
  else:
    print("No songs entered. Please provide song names and years for
recommendations.")

def get_song_data(song, spotify_data):
    try:
        # Convert both input and dataframe song names to lowercase for
case-insensitive comparison
        song_data = spotify_data[(spotify_data['name'].str.lower() ==
song['name'].lower()) & (spotify_data['year'] == song['year'])]
        if song_data.empty:
            print(f"Warning: Song '{song['name']}' from {song['year']}
not found in dataset.")
            return None
        else:
            return song_data.iloc[0]

    except IndexError:
        # Assuming find_song handles potential case sensitivity or
uses a different search method
        return find_song(song['name'], song['year'])

recommend_songs_interactive(data)
```

```
Enter song name (or type 'done' to finish): just the way you are
Enter song year: 2010
Enter song name (or type 'done' to finish): done

Here are some song recommendations:
- Just the Way You Are (2010) by ['Bruno Mars']
- Heart Of Glass (Live from the iHeart Festival) (2020) by ['Miley
Cyrus']
- Back To You - From 13 Reasons Why – Season 2 Soundtrack (2018) by
['Selena Gomez']
- What I've Done (2007) by ['Linkin Park']
- Float On (2004) by ['Modest Mouse']
- CROWN (2019) by ['TOMORROW X TOGETHER']
- Fire and the Flood (2014) by ['Vance Joy']
- Tongue Tied (2011) by ['Grouplove']
- Back To You (2018) by ['Selena Gomez']
- Lovesick Girls (2020) by ['BLACKPINK']

/usr/local/lib/python3.12/dist-packages/sklearn/utils/
validation.py:2739: UserWarning:

X does not have valid feature names, but StandardScaler was fitted
with feature names
```