```
 import os
 import sys
 from tempfile import NamedTemporaryFile
 from urllib.request import urlopen
 from urllib.parse import unquote, urlparse
 from urllib.error import HTTPError
 from zipfile import ZipFile
 import tarfile
 import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = 'spotify-dataset:https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-data-sets%2F1800580%2F2936818%2Fbundle%2Farchive.zip%3FX

 KAGGLE_INPUT_PATH='/kaggle/input'
 KAGGLE_WORKING_PATH='/kaggle/working'
 KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)


  try:
    os.symlink(KAGGLE_INPUT_PATH, os.path.join("..", 'input'), target_is_directory=True)
  except FileExistsError:
    pass
  try:
    os.symlink(KAGGLE_WORKING_PATH, os.path.join("..", 'working'), target_is_directory=True)
  except FileExistsError:
    pass


   for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
       directory, download_url_encoded = data_source_mapping.split(':')
       download_url = unquote(download_url_encoded)
       filename = urlparse(download_url).path
       destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
       try:
           with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
               total_length = fileres.headers['content-length']
               print(f'Downloading {directory}, {total_length} bytes compressed')
               dl = 0
               data = fileres.read(CHUNK_SIZE)
               while len(data) > 0:
                   dl += len(data)
                   tfile.write(data)
                   done = int(50 * dl / int(total_length))
                   sys.stdout.write(f"\r[{'=' * done}{' ' * (50-done)}] {dl} bytes downloaded")
                   sys.stdout.flush()
                   data = fileres.read(CHUNK_SIZE)
               if filename.endswith('.zip'):
                 with ZipFile(tfile) as zfile:
                   zfile.extractall(destination_path)
               else:
                 with tarfile.open(tfile.name) as tarfile:
                   tarfile.extractall(destination_path)
               print(f'\nDownloaded and uncompressed: {directory}')
       except HTTPError as e:
           print(f'Failed to load (likely expired) {download_url} to path {destination_path}')
           continue
       except OSError as e:
           print(f'Failed to load {download_url} to path {destination_path}')
           continue



      Downloading spotify-dataset, 17275602 bytes compressed
      [==================================================] 17275602 bytes downloaded

        print('Data  source import complete.')
```

```
Downloaded and uncompressed: spotify-dataset
Data source import complete.
```

Building Music Recommendation System using Spotify Dataset

Import 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
%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

import warnings
warnings.filterwarnings("ignore")
```

Read Data

```
data = pd.read_csv("../input/spotify-dataset/data/data.csv")
genre_data = pd.read_csv('../input/spotify-dataset/data/data_by_genres.csv')
year_data = pd.read_csv('../input/spotify-dataset/data/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
```

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
```

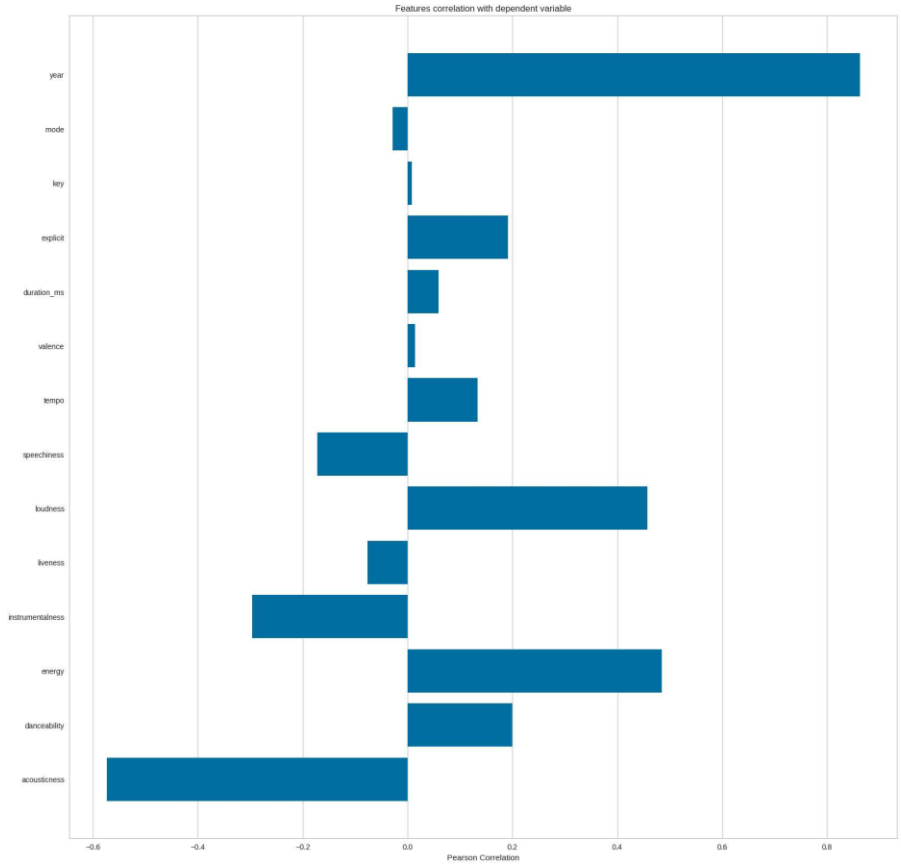
```
print(year_data.info())
```

```
<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

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

```
X, y = data[feature_names], data['popularity']

# Create a list of the feature names
features = np.array(feature_names)

# Instantiate the visualizer
visualizer = FeatureCorrelation(labels=features)

plt.rcParams['figure.figsize']=(20,20)
visualizer.fit(X, y)     # Fit the data to the visualizer
visualizer.show()
```

OUTPUT:




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

Data Understanding by Visualization and EDA

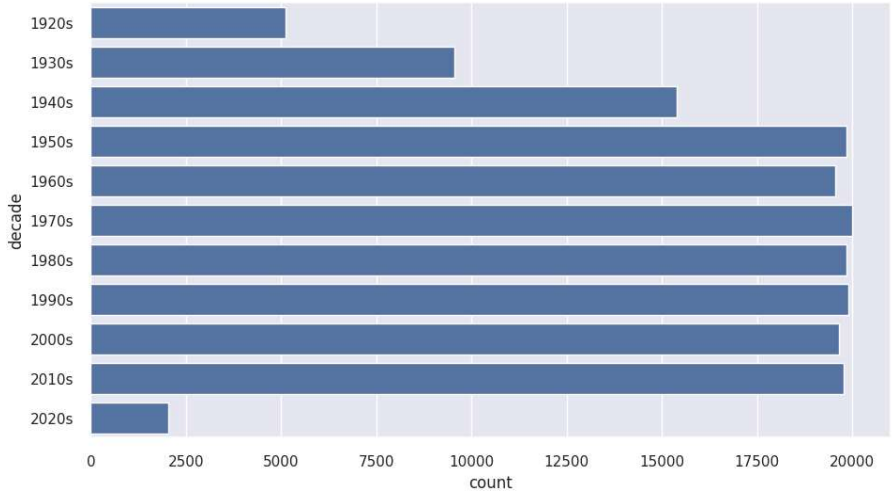
MusicOver Time

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

sns.set(rc={'figure.figsize':(11
sns.countplot(data['decade'])
```

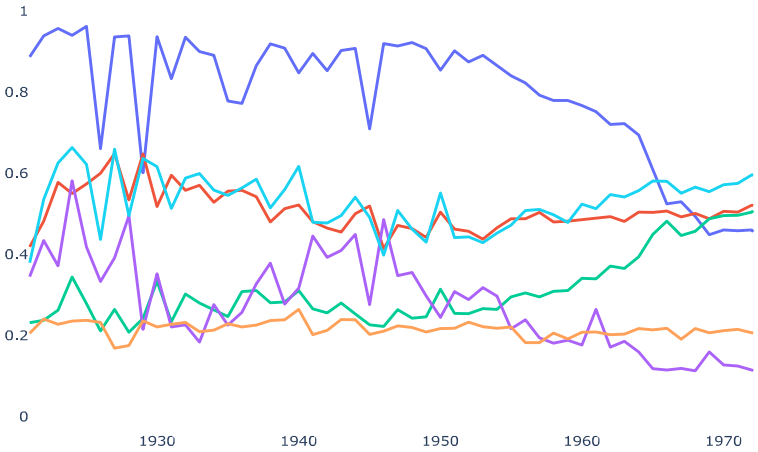
OUTPUT

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

                                ,6)})
```

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



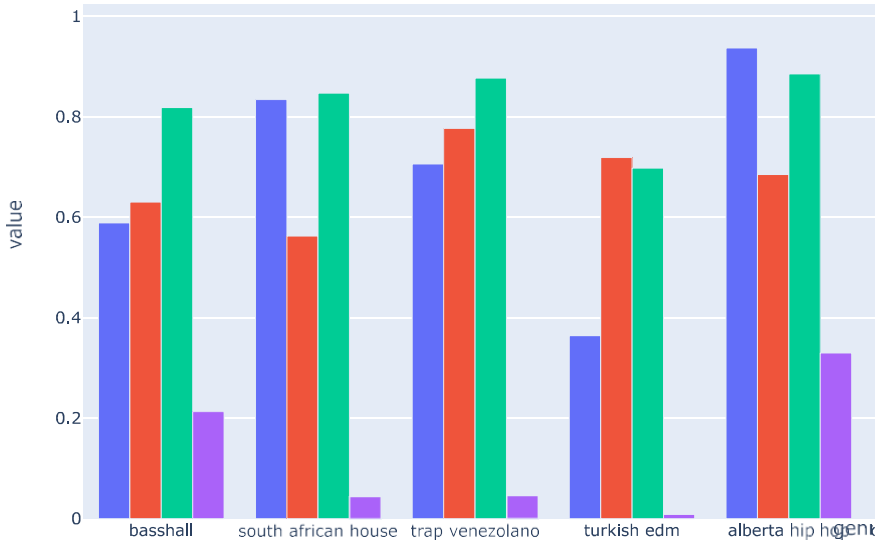

```
sound_features = ['acousticness', 'danceability', 'energy','instrumentalness','liveness',  'valence']
fig = px.line(year_data, x='year', y=sound_features)
fig.show()
```



Characteristics of Different Genres

```
fig = px.bar(top10_genres,
fig.show()

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

x='genres',  y=['valence',  'energy', 'danceability','acousticness'], barmode='group')
```

OUTPUT:

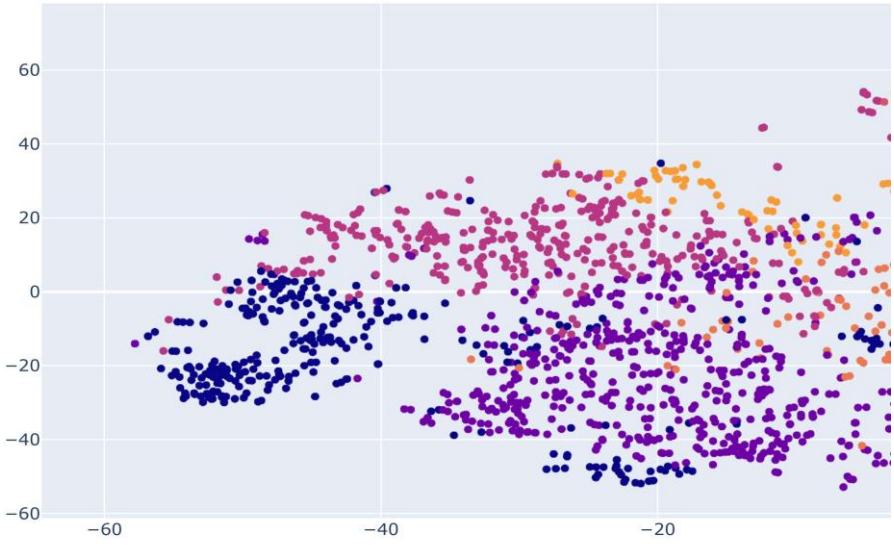


Clustering Genres with K-Means

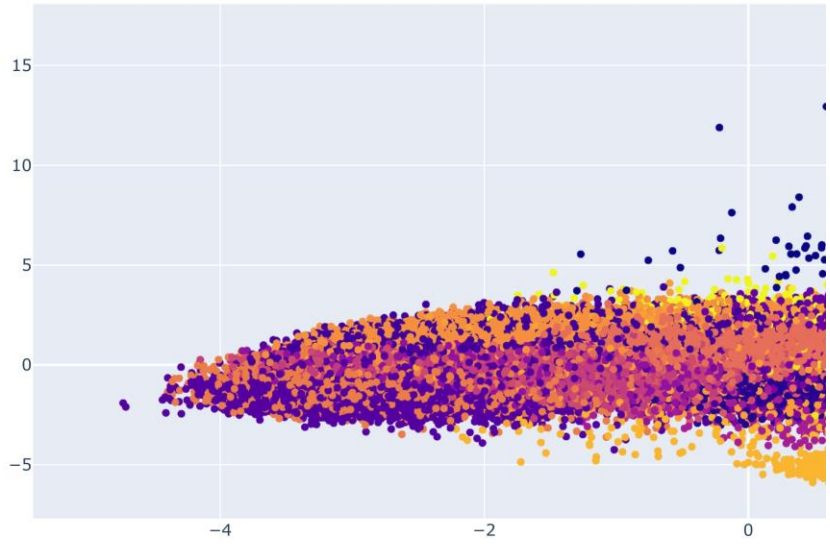
```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

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

```
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_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 0.008s...
[t-SNE] Computed neighbors for 2973 samples in 1.003s...
[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                                    76.106354
[t-SNE] KL divergence after 250 iterations with early exaggeration:
[t-SNE] KL divergence after 1000 iterations: 1.392001
```



```
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


# Visualizing the Clusters with 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()
```



Build Recommender System

```
!pip install spotipy
```

```
Collecting spotipy
  Downloading spotipy-2.23.0-py3-none-any.whl (29 kB)
Collecting redis>=3.5.3 (from spotipy)
  Downloading redis-5.0.3-py3-none-any.whl (251 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 251.8/251.8 kB 7.7 MB/s eta 0:00:00
```

```
Requirement already satisfied: requests>=2.25.0 in /usr/local/lib/python3.10/dist-packages (from spotipy) (2.31.0)
Requirement already satisfied: six>=1.15.0 in /usr/local/lib/python3.10/dist-packages (from spotipy) (1.16.0)
Requirement already satisfied: urllib3>=1.26.0 in /usr/local/lib/python3.10/dist-packages (from spotipy) (2.0.7)
Requirement already satisfied: async-timeout>=4.0.3 in /usr/local/lib/python3.10/dist-packages (from redis>=3.5.3->spotipy) (4.0.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.25.0->spotipy) (3.3
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.25.0->spotipy) (3.6)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.25.0->spotipy) (2024.2.2)
Installing collected packages: redis, spotipy
Successfully installed redis-5.0.3 spotipy-2.23.0
```

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

def find_song(name, year):
    # Initialize Spotify client
    sp = spotipy.Spotify(auth_manager=SpotifyClientCredentials(client_id=os.environ["SPOTIFY_CLIENT_ID"],
                                                               client_secret=os.environ["SPOTIFY_CLIENT_SECRET"]))

    # Dictionary to store song data
    song_data = defaultdict()

    # Search for the song
    results = sp.search(q='track:{} year:{}'.format(name, year), limit=1)
    if not results['tracks']['items']:
        print("Song not found.")
        return None

    # Get track ID and fetch audio features
    track_id = results['tracks']['items'][0]['id']
    try:
        audio_features = sp.audio_features(track_id)[0]
    except:
        print("Error fetching audio features.")
        return None

    # Extract basic track info
    song_data['name'] = name
    song_data['year'] = year
    song_data['explicit'] = int(results['tracks']['items'][0]['explicit'])
    song_data['duration_ms'] = results['tracks']['items'][0]['duration_ms']
    song_data['popularity'] = results['tracks']['items'][0]['popularity']

    # Extract audio features
    for key, value in audio_features.items():
        song_data[key] = value

    return dict(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'])].iloc[0]
        return song_data

    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)

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)           -1))

      song_center = get_mean_vector(song_list, spotify_data)
      scaler = song_cluster_pipeline.steps[0][1]
      scaled_data = scaler.transform(spotify_data[number_cols])
      scaled_song_center = scaler.transform(song_center.reshape(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')







      [{'name': 'Life is a Highway   From "Cars"',
                ~~~~
```