[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sb
from tqdm import tqdm
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
import warnings
warnings.filterwarnings('ignore')
music_data = pd.read_csv("C:/Users/samuela/Desktop/ML/Spotify Tracks Dataset.csv")
music_data.head()
music_data.isnull().sum()
music_data.info()
music_data.shape
music_data.isnull().sum().plot.bar()
plt.show()
music_data.select_dtypes(np.number)
music_data["explicit"] = music_data["explicit"].astype(int)
music_data.head()
visual_data = music_data.drop(columns=['song', 'artist', 'year', 'genre'])
plt.figure(figsize=(20, 20))
for i in tqdm(np.arange(1, len(visual_data.columns))):
plt.subplot(9, 2, i)
sb.barplot(x=music_data.year,y=visual_data[visual_data.columns[i]])
plt.xticks(rotation=45);
plt.show()
plt.subplots(figsize=(12, 8))
sb.heatmap(visual_data.corr(), annot=True, square=True)
plt.show()
from sklearn.preprocessing import OneHotEncoder
unique_genres = set()
for genre_list in music_data["genre"]:
genres = genre_list.split(",")
for genre in genres:
```

```
unique_genres.add(genre)
# Create a one-hot encoding for the genre column
encoder = OneHotEncoder()
encoder.fit([[genre] for genre in unique_genres])
# Encode the genre data
encoded_genres = []
for genres in music_data["genre"]:
genres = genres.split(",")
one_hot = [0 if genre not in genres else 1 for genre in unique_genres]
encoded_genres.append(one_hot)
import os
import seaborn as sns
import plotly.express as px
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.metrics import euclidean_distances
from scipy.spatial.distance import cdist
import warnings
warnings.filterwarnings("ignore")
def normalize_column(col):
max_d = music_data[col].max()
min_d = music_data[col].min()
music_data[col] = (music_data[col] - min_d)/(max_d - min_d)
num_types = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
num = music_data.select_dtypes(include=num_types)
for col in num.columns:
if col != 'year':
normalize_column(col)
music_data.select_dtypes(np.number).drop(columns = ['year']).plot(kind='box',␣
 ,figsize=(20, 20) ,fontsize=10)
model = TSNE(n_components = 2, random_state = 0)
music_data_modified = music_data.select_dtypes(np.number).drop(columns=['year'])
tsne_data = model.fit_transform(music_data_modified)
plt.style.use('seaborn')
plt.figure(figsize = (7, 7))
plt.scatter(tsne_data[:,0], tsne_data[:,1], marker= '*')
plt.show()
# Create a DataFrame from encoded genres
encoded_genres_df = pd.DataFrame(encoded_genres, columns=list(unique_genres))
# Concatenate the encoded genres DataFrame with the original dataset
music_data = pd.concat([music_data, encoded_genres_df], axis=1)
# View the dataset with the encoded genres
print(music_data.head())
from sklearn.cluster import KMeans
km = KMeans(n_clusters=10)
```

```
cat = km.fit_predict(num)
music_data['cat'] = cat
normalize_column('cat')
music_data.cat[:10]
cluster_pipeline = Pipeline([('scaler', StandardScaler()), ('kmeans',
    KMeans(n_clusters=10))])
X = music_data.select_dtypes(np.number)
cluster_pipeline.fit(X)
music_data['cluster'] = cluster_pipeline.predict(X)

import plotly.express as px
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'] = music_data['genre']
projection['cluster'] = music_data['cluster']
clu = px.scatter(projection, x='x', y='y', color='cluster', hover_data=['x',
    'y', 'genres'])
clu
clu.show()
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
X = music_data.select_dtypes(np.number).drop(columns =
    ['cat','cluster','year']).copy()
y = music_data['cluster']
X_train, X_rem, y_train, y_rem = train_test_split(X,y, train_size=0.8,
    random_state=0)
X_valid, X_test, y_valid, y_test = train_test_split(X_rem,y_rem, test_size=0.5)

print(X_train.shape), print(y_train.shape)
print(X_valid.shape), print(y_valid.shape)
print(X_test.shape), print(y_test.shape)
knn1= KNeighborsClassifier(metric='cosine', algorithm='brute', n_neighbors=1)
knn5= KNeighborsClassifier(metric='cosine', algorithm='brute', n_neighbors=5)
knn10= KNeighborsClassifier(metric='cosine', algorithm='brute', n_neighbors=10)
knn5.fit(X_train, y_train)
knn1.fit(X_train, y_train)
knn10.fit(X_train, y_train)
knn5.fit(X_valid, y_valid)
knn1.fit(X_valid, y_valid)
knn10.fit(X_valid, y_valid)
knn5.fit(X_train, y_train)
knn1.fit(X_train, y_train)
knn10.fit(X_train, y_train)
y_pred_5 = knn5.predict(X_valid)
y_pred_1 = knn1.predict(X_valid)
```

```
y_pred_10 = knn1.predict(X_valid)

from sklearn.metrics import accuracy_score
print("Accuracy with k=5", accuracy_score(y_valid, y_pred_5)*100)
print("Accuracy with k=1", accuracy_score(y_valid, y_pred_1)*100)
print("Accuracy with k=10", accuracy_score(y_valid, y_pred_10)*100)
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_valid, y_pred_1))
print(confusion_matrix(y_valid, y_pred_5))
print(confusion_matrix(y_valid, y_pred_10))
print(classification_report(y_valid, y_pred_1))
print(classification_report(y_valid, y_pred_5))
print(classification_report(y_valid, y_pred_10))
print(classification_report(y_valid, y_pred_1))
print(classification_report(y_valid, y_pred_5))
print(classification_report(y_valid, y_pred_10))
cmap = sb.cubehelix_palette(as_cmap=True)
plt.figure(figsize = (15,5))
plt.subplot(1,2,1)
plt.scatter(tsne_data_X_valid[:,0], tsne_data_X_valid[:,1], c=y_pred_5, marker=
  '*', s=100, cmap=cmap)
plt.title("Predicted values with k=5", fontsize=20)
plt.subplot(1,2,2)
plt.scatter(tsne_data_X_valid[:,0], tsne_data_X_valid[:,1], c=y_pred_1, marker=
  '*', s=100, cmap=cmap)
plt.title("Predicted values with k=1", fontsize=20)
plt.show()
plt.subplot(1,2,2)
plt.scatter(tsne_data_X_valid[:,0], tsne_data_X_valid[:,1], c=y_pred_10,
  marker= '*', s=100, cmap=cmap)
plt.title("Predicted values with k=10", fontsize=20)
plt.show()
from fuzzywuzzy import process
X_test
recommendation_set = music_data.merge(X_test, how = 'inner' ,indicator=False)
recommendation_set
def recommender(song_name, data,model):
idx=process.extractOne(song_name, recommendation_set['song'])[2]
print('Song Selected:-',recommendation_set['song'][idx], 'Index: ',idx)
print('Searching for recommendations.....')
requiredSongs = recommendation_set.select_dtypes(np.number).drop(columns =
  ['cat','cluster','year']).copy()
distances, indices = model.kneighbors(requiredSongs.iloc[idx].values.
  reshape(1,-1))
for i in indices:
print(music_data['song'][i] + " " + music_data['artist'][i])
def get_song_info(row_number):
```

```
song_info = recommendation_set.loc[row_number, ["song", "artist"]]
return song_info
# Get user input for song name and artist name
user_song = input("Enter the song name: ")
user_artist = input("Enter the artist name: ")
# Find the row number of the song in the recommendation_set
matching_rows = recommendation_set[(recommendation_set["song"] == user_song) &
 (recommendation_set["artist"] == user_artist)]
if len(matching_rows) == 0:
print("Song not found.")
else:
row_number = matching_rows.index[0]
# Get the song info using the row number
song_info = get_song_info(row_number)
print("Song name: ", song_info["song"])
print("Artist name: ", song_info["artist"])
# Use the song name for recommendation
song_name = song_info["song"]
recommender(song_name, X_test, knn5)
```