import pandas as pd

from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering

import matplotlib.pyplot as plt

import seaborn as sns

from scipy import stats  
import numpy as np

from sklearn.decomposition import PCA

tracks\_df = pd.read\_csv("spotify\_tracks.csv")

tracks\_df.head()

#removed duplicates

tracks\_df = tracks\_df.groupby(['track\_name', 'artists']).first().reset\_index()

tracks\_df.shape

#histogram

tracks\_df.hist(bins = 30, figsize=(20,15))

plt.suptitle("Histograms for each variables")

plt.show()

#heatmap

numeric\_df = tracks\_df.drop(columns=['track\_id', 'artists', 'album\_name', 'track\_name', 'track\_genre'])

corr = numeric\_df.corr()

#strong relationship between variables

plt.scatter(x='loudness', y='energy', data=tracks\_df, s=0.5)

plt.suptitle("Scatter plot of the relationship between Loudness and Energy")

plt.xlabel('Loudness')

plt.ylabel('Energy')

plt.show()

plt.scatter(x='energy', y='acousticness', data=tracks\_df, s=0.5)

plt.suptitle("Scatter plot of the relationship between Energy and Acousticness")

plt.xlabel('Energy')

plt.ylabel('Acousticness')

plt.show()

plt.figure(figsize=(20,15))

sns.heatmap(corr, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5, square=True)

plt.title('Correlation Matrix Heatmap')

plt.show()

**Part 2:**

#To avoid TypeErrors

numeric\_columns = ['acousticness', 'danceability', 'duration\_ms', 'energy', 'instrumentalness',

'key', 'liveness', 'loudness', 'mode', ‘popularity’, 'speechiness', 'tempo',

'time\_signature', 'valence'] # Adjust this list based on your actual numeric columns

numeric\_df = tracks\_df[numeric\_columns]

numeric\_df = numeric\_df.dropna(subset=numeric\_columns)

# Z-score method to get rid of outliers

def RemoveOutliersZScore(df, threshold=3):

z\_scores = stats.zscore(df)

abs\_z\_scores = abs(z\_scores)

filtered\_entries = (abs\_z\_scores < threshold).all(axis=1)

return df[filtered\_entries]

# Applying method to df

Trimmed\_df = RemoveOutliersZScore(numeric\_df)

# Showing before and after of df

print(f"Original df:\n{numeric\_df.shape}")

print(f"Updated df:\n{Trimmed\_df.shape}")

# Elbow Method

def ElbowMethod(df):

Sum\_Of\_Distance = []

for i in range(1, 11):

kmeans = KMeans(n\_clusters=i, random\_state=42)

kmeans.fit(df)

Sum\_Of\_Distance.append(kmeans.inertia\_)

plt.figure(figsize=(10, 6))

plt.plot(range(1, 11), Sum\_Of\_Distance, marker='o')

plt.title('Elbow Method')

plt.xlabel('Number of clusters')

plt.ylabel('Error')

plt.show()

# Showing the optimal amount of clusters

ElbowMethod(Trimmed\_df)

subsampled\_df = Trimmed\_df.sample(frac=0.005, random\_state=42)

# Apply PCA to reduce the number of dimensions to 2

pca = PCA(n\_components=2)

reduced\_data = pca.fit\_transform(subsampled\_df)

# KMeans Clustering

kmeans = KMeans(n\_clusters=5, random\_state=42)

kmeans.fit(Trimmed\_df)

# Ensure indices and lengths match before assignment (added for Part 3)

if len(kmeans.labels\_) == len(Trimmed\_df.index):

tracks\_df.loc[Trimmed\_df.index, 'kmeans\_cluster'] = kmeans.labels\_

# Fill NaN values with a placeholder (e.g., -1) before converting to integers

tracks\_df['kmeans\_cluster'].fillna(-1, inplace=True)

tracks\_df['kmeans\_cluster'] = tracks\_df['kmeans\_cluster'].astype(int) # Ensure the cluster labels are integers

else:

print("Error: Mismatch in length of indices and KMeans labels.")

# Print unique values in kmeans\_cluster to verify (added for Part 3)

print(tracks\_df['kmeans\_cluster'].unique())

#Function to plot clusters for reusability

def plot\_clusters(data, labels, title):

plt.figure(figsize=(10, 6))

sns.scatterplot(x='energy', y='acousticness', hue=labels, palette='viridis', data=data, s=50)

plt.title(title)

plt.xlabel('Energy')

plt.ylabel('Acousticness')

plt.legend(loc='best')

plt.show()

def plot\_clusters\_pca(data, reduced\_data, labels, title):

plt.figure(figsize=(10, 6))

sns.scatterplot(x=reduced\_data[:, 0], y=reduced\_data[:, 1], hue=labels, palette='viridis', s=50)

plt.title(title)

plt.xlabel('Energy')

plt.ylabel('Acousticness')

plt.legend(loc='best')

plt.show()

#Plotting Kmeans clustering

plot\_clusters(tracks\_df, 'kmeans\_cluster', 'KMeans Clusters')

#DBSCAN Clustering

dbscan = DBSCAN(eps=0.1, min\_samples=15)

labels = dbscan.fit\_predict(reduced\_data)

# Add the labels to the df

subsampled\_df = subsampled\_df.reset\_index()

subsampled\_df['dbscan\_cluster'] = labels

plot\_clusters\_pca(subsampled\_df, reduced\_data, labels, 'DBSCAN Clustering with Adjusted Parameters')

#Agglomerative Clustering

Agglomerative = AgglomerativeClustering(n\_clusters=5, linkage='ward')

labels = Agglomerative.fit\_predict(reduced\_data)

#Labels

subsampled\_df = subsampled\_df.reset\_index()

subsampled\_df['Agglomerative\_Cluster'] = labels

#Plotting Agglomerative Clustering

plot\_clusters\_pca(subsampled\_df, reduced\_data, labels, 'Agglomerative Clustering with PCA Components')

**Part 3:**  
# Only numeric columns are used

numeric\_df\_with\_clusters = tracks\_df[numeric\_columns + ['kmeans\_cluster']]

# Calculate the correlation matrix

corr = numeric\_df\_with\_clusters.corr()

# Check if all clusters are included

clusters\_present = tracks\_df['kmeans\_cluster'].unique()

for cluster in clusters\_present:

if cluster not in corr.index:

corr.loc[cluster] = [0] \* len(corr.columns)

corr[cluster] = 0

# Print the correlation matrix

print(corr)

# Two valid song ids for testing purposes

valid\_ids = tracks\_df.loc[Trimmed\_df.index, 'track\_id'].head(2).tolist()

print("Valid song IDs for testing:", valid\_ids)

# Prompt the user to enter their favorite songs' IDs (ensure they are string-based)

ids = input('Enter comma-separated IDs of your favorite songs:\n> ').strip().split(',')

# Trim whitespace from each ID

ids = [id.strip() for id in ids]

# Filter the dataframe to get the user's favorite songs

favorites = tracks\_df[tracks\_df['track\_id'].isin(ids)]

# Find most frequent cluster

clusters = favorites['kmeans\_cluster'].value\_counts()

user\_favorite\_cluster = clusters.idxmax()

print('\nFavorite cluster:', user\_favorite\_cluster, '\n')

# Suggest songs

suggestions = tracks\_df[tracks\_df['kmeans\_cluster'] == user\_favorite\_cluster]

print("Top 5 song suggestions:")

print(suggestions[['track\_name', 'artists', 'album\_name']].head())

# Function to get recommendations from different clusters

def get\_different\_recommendations(tracks, favorite\_cluster, corr):

if favorite\_cluster in corr.index:

different\_clusters = corr.loc[favorite\_cluster].sort\_values(ascending=True).index.tolist()

different\_clusters.remove(favorite\_cluster)

different\_suggestions = tracks[tracks['kmeans\_cluster'].isin(different\_clusters)].head(10)

return different\_suggestions

else:

print(f"Cluster {favorite\_cluster} not found in the correlation matrix.")

return pd.DataFrame() # Return an empty DataFrame if the cluster is not found

# Collect feedback

def collect\_feedback(tracks, user\_favorite\_cluster, corr):

feedback = input("\nDid you like these recommendations? (yes/no):\n> ").strip().lower()

if feedback == 'yes':

print("\nGreat!")

elif feedback == 'no':

print("\nSorry to hear that. Try these songs:")

different\_suggestions = get\_different\_recommendations(tracks, user\_favorite\_cluster, corr)

if not different\_suggestions.empty:

print(different\_suggestions[['track\_name', 'artists', 'album\_name']])

else:

print("No alternative recommendations available.")

else:

print("\nInvalid input. Please enter 'yes' or 'no'.")

# Use the feedback function

collect\_feedback(tracks\_df, user\_favorite\_cluster, corr)

**Edited Part 3 (Embedded part 4, changed song ID input -> song title and author input)**

Idk why the background is black i wrote it in pycharm and i can’t get rid of it 😭

I think it becomes normal again if you paste it into a jupyter kernel tho so its ok

# Ensure the data is numeric

numeric\_columns = ['acousticness', 'danceability', 'duration\_ms', 'energy',

'instrumentalness', 'key', 'liveness', 'loudness', 'mode', 'popularity',

'speechiness', 'tempo', 'time\_signature', 'valence'] # Adjust this list based on your actual numeric columns

numeric\_df = tracks\_df[numeric\_columns]

# Handle NaN

numeric\_df = numeric\_df.dropna(subset=numeric\_columns)

# Z-score method to get rid of outliers

def RemoveOutliersZScore(df, threshold=3):

z\_scores = stats.zscore(df)

abs\_z\_scores = abs(z\_scores)

filtered\_entries = (abs\_z\_scores < threshold).all(axis=1)

return df[filtered\_entries]

# Applying method to df

Trimmed\_df = RemoveOutliersZScore(numeric\_df)

# Showing before and after of df

print(f"Original df:\n{numeric\_df.shape}")

print(f"Updated df:\n{Trimmed\_df.shape}")

# ELBOW Method

def ElbowMethod(df):

Sum\_Of\_Distance = []

for i in range(1, 11):

kmeans = KMeans(n\_clusters=i, random\_state=42)

kmeans.fit(df)

Sum\_Of\_Distance.append(kmeans.inertia\_)

plt.figure(figsize=(10, 6))

plt.plot(range(1, 11), Sum\_Of\_Distance, marker='o')

plt.title('Elbow Method')

plt.xlabel('Number of clusters')

plt.ylabel('Error')

plt.show()

#Functions to plot clusters for reusability

def plot\_clusters(data, labels, title):

plt.figure(figsize=(10, 6))

sns.scatterplot(x='energy', y='acousticness', hue=labels, palette='viridis', data=data, s=50)

plt.title(title)

plt.xlabel('Energy')

plt.ylabel('Acousticness')

plt.legend(loc='best')

plt.show()

# Optimal amount of clusters

ElbowMethod(Trimmed\_df)

kmeans = KMeans(n\_clusters=5, random\_state=42)

kmeans.fit(Trimmed\_df)

# Ensure indices and lengths match before predictions

if len(kmeans.labels\_) == len(Trimmed\_df.index):

tracks\_df.loc[Trimmed\_df.index, 'kmeans\_cluster'] = kmeans.labels\_

tracks\_df['kmeans\_cluster'] = tracks\_df['kmeans\_cluster'].fillna(-1)

tracks\_df['kmeans\_cluster'] = tracks\_df['kmeans\_cluster'].astype(int) # Ensure the cluster labels are integers

else:

print("Error: Mismatch in length of indices and KMeans labels.")

plot\_clusters(tracks\_df, 'kmeans\_cluster', 'KMeans Clusters')

# Ensure only numeric columns are used for correlation calculation

numeric\_df\_with\_clusters = tracks\_df[numeric\_columns + ['kmeans\_cluster']]

# Calculate and plot the correlation matrix

corr = numeric\_df\_with\_clusters.corr()

plt.figure(figsize=(20,15))

sns.heatmap(corr, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5, square=True)

plt.title('Data with Clusters Correlation Matrix Heatmap')

# plt.show()

# Remove duplicate songs

tracks\_df = tracks\_df.drop\_duplicates(subset=["artists", "track\_name"])

# Prompt favorite songs' names and artists(ensure they are string-based and not case sensitive)

songs = input('Enter comma-separated song titles and artists of your favorite songs: (e.g: Let It Be - Remastered 2015:The Beatles)\n> ').strip().split(',')

songs = [song.strip().lower().split(':') for song in songs]

song\_names = []

song\_artists = []

for i in range(len(songs)):

for j in range(2):

songs[i][j] = songs[i][j].strip()

song\_names.append(songs[i][0])

song\_artists.append(songs[i][1])

favorites = tracks\_df[tracks\_df['track\_name'].str.lower().str.strip().isin(song\_names) &

tracks\_df['artists'].str.lower().str.strip().isin(song\_artists)]

# Find most frequent cluster

clusters = favorites['kmeans\_cluster'].value\_counts()

user\_favorite\_cluster = clusters.idxmax()

print('\nFavorite cluster:', user\_favorite\_cluster, '\n')

# Suggest songs

suggestions = tracks\_df[tracks\_df['kmeans\_cluster'] == user\_favorite\_cluster]

#Function to collect user's mood and generate range

def ask\_mood():

mood = input('How would you rate your mood today on a scale of 1-5, where 1 is the worst and 5 is the best?\n>')

mood\_range = [0, 0]

mood=int(mood)

if(mood not in range(1,6)):

print("Invalid input, try again:\n>")

else:

mood = int(mood)

mood\_range[0] = (mood-1)\*(0.6)

mood\_range[1] = mood\_range[0]+0.6

return mood\_range

#Ask a Y/N question for a situation

def ask\_situation(question, yes='Y', no='N'):

situation = input(question)

if(situation==yes):

return True

elif(situation==no):

return False

else:

print("Invalid input, try again:")

return ask\_situation(question)

#Filter for specific situations

def filter\_situation():

track\_sf = suggestions

wanted = ask\_situation("Do you want further filtering? Y/N\n>")

if (not wanted):

return track\_sf

pg = ask\_situation("Are you with young children/family? Y/N\n>")

explicity = lambda x : track\_sf[track\_sf['explicit']==False] if x else track\_sf

track\_sf = explicity(pg)

sing = ask\_situation("Are you looking for songs to sing along to? Y/N\n>")

singalong = lambda x : track\_sf[track\_sf['popularity']>=75] if x else track\_sf

track\_sf = singalong(sing)

dance = ask\_situation("What about dance? Y/N\n>")

dancealong = lambda x : track\_sf[track\_sf['danceability']>0.8] if x else track\_sf

track\_sf = dancealong(dance)

background = ask\_situation("Are you looking for background music, or lyrical music? B/L\n>", 'B', 'L')

instrumental = lambda x : track\_sf[track\_sf['speechiness']<0.2] if x else track\_sf[track\_sf['speechiness']>0.2]

track\_sf = instrumental(background)

return track\_sf

mood\_range = ask\_mood()

sugggestions = suggestions[(suggestions['valence']+suggestions['energy']+suggestions['liveness']>mood\_range[0]) &

(suggestions['valence']+suggestions['energy']+suggestions['liveness']<mood\_range[1])]

suggestions = filter\_situation()

# Sort by popularity

suggestions = suggestions.sort\_values(by='popularity', ascending=False)

print("Top 5 song suggestions:")

print(suggestions[['track\_name', 'artists', 'album\_name']].head())

# Function to get recommendations from different clusters

def get\_different\_recommendations(tracks, favorite\_cluster, corr):

if favorite\_cluster in corr.index:

different\_clusters = corr.loc[favorite\_cluster].sort\_values(ascending=True).index.tolist()

different\_clusters.remove(favorite\_cluster)

different\_suggestions = tracks[tracks['kmeans\_cluster'].isin(different\_clusters)].head(10)

return different\_suggestions

else:

print(f"Cluster {favorite\_cluster} not found in the correlation matrix.")

return pd.DataFrame() # Return an empty DataFrame if the cluster is not found

# Feedback

def collect\_feedback(tracks, user\_favorite\_cluster, corr):

feedback = ask\_situation("Did you like these recommendations? Y/N \n")

if feedback:

print("\nGreat!")

elif not feedback:

print("\nSorry to hear that. Try these songs:")

different\_suggestions = get\_different\_recommendations(tracks, user\_favorite\_cluster, corr)

if not different\_suggestions.empty:

print(different\_suggestions[['track\_name', 'artists', 'album\_name']])

else:

print("No alternative recommendations available.")

else:

print("\nInvalid input. Please enter 'yes' or 'no'.")

# Prompt feedback

collect\_feedback(tracks\_df, user\_favorite\_cluster, corr)

# Collect initial feedback

collect\_feedback()