Name:Isha Poojary
Githublink:https://github.com/isha9380/Project-Music-Recommendaton-System.git

Music Recommendation System

Objective: The goal of this project is to suggest songs similar to the one entered by the user by using machine learning (clustering) on audio features from a Spotify dataset.

This is a Streamlit-based web application that allows users to search for songs and get personalized recommendations based on song clustering. It uses a pre-clustered dataset of music tracks, which helps in finding songs that are similar in style, mood, or features.

- Build a user-friendly web app that takes a song name as input
- Search for the input song in a pre-clustered dataset.
- Recommend other songs from the same cluster, assuming they are similar in style or sound.
- Use Streamlit for the frontend, and a machine learning clustering model in the backend to group similar songs together.

Project Workflow:

1. Data Collection:

- A dataset is collected from Spotify, containing songs and their audio features such as:danceability, energy, tempo, valence, etc.
- This dataset forms the foundation for clustering.

2. Data Preprocessing:

- Handle missing or duplicate values.
- Normalize or scale numeric features.
- Convert categorical values if needed (e.g., genres).
- Select important features for clustering.

3. Clustering (Machine Learning):

- Apply unsupervised learning (e.g., KMeans) to group songs into clusters based on their audio characteristics.
- Add a new column cluster to label each song's group.

- Save the result to a file, e.g., clustered_df.csv.

4. Web App Development (Streamlit):

- Create a web interface using Streamlit (`app.py`):
- Loads the clustered dataset.
- Takes user input (a song name).
- Searches for the song in the dataset.
- Identifies the cluster of the matched song.
- Recommends other songs from the same cluster (similar audio features).
- If no matching song is found, it warns the user. If the dataset is missing the expected columns, it shows appropriate error messages.

5. User Interaction Flow:

- User opens the web app.
- Input a song name.
- App finds matches and displays them.
- App recommends similar songs based on clustermatch.

6. Output:

Display:

- Matching songs.
- Recommended songs (from the same cluster).

Handle cases where:

- No song is found.
- Required columns are missing.

About the Dataset:

The dataset appears to be based on Spotify's music data, which includes both song metadata and audio features extracted using the Spotify Web API. The specific file used in your project is named:clustered_df.csv

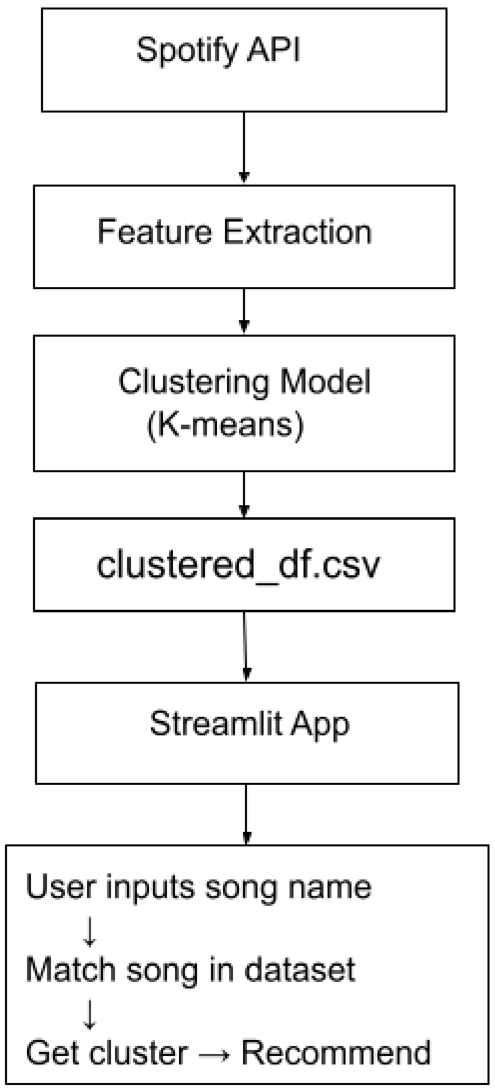
The dataset contains metadata of various songs and has been pre-processed and clustered to support recommendation tasks. Each record in the dataset represents a single song with relevant features extracted for clustering and matching.

This dataset is a preprocessed version where songs have been grouped into clusters based on their audio characteristics.

Contents of the Dataset:

- **name**: The title of the song.
- **artists**: The name(s) of the artist(s) associated with the song.
- **cluster** (or similarly named): Represents the cluster assigned to a song based on its audio features (e.g., tempo, energy, valence, etc.). Songs in the same cluster are considered similar and are used for recommendations.

System Flowchart:

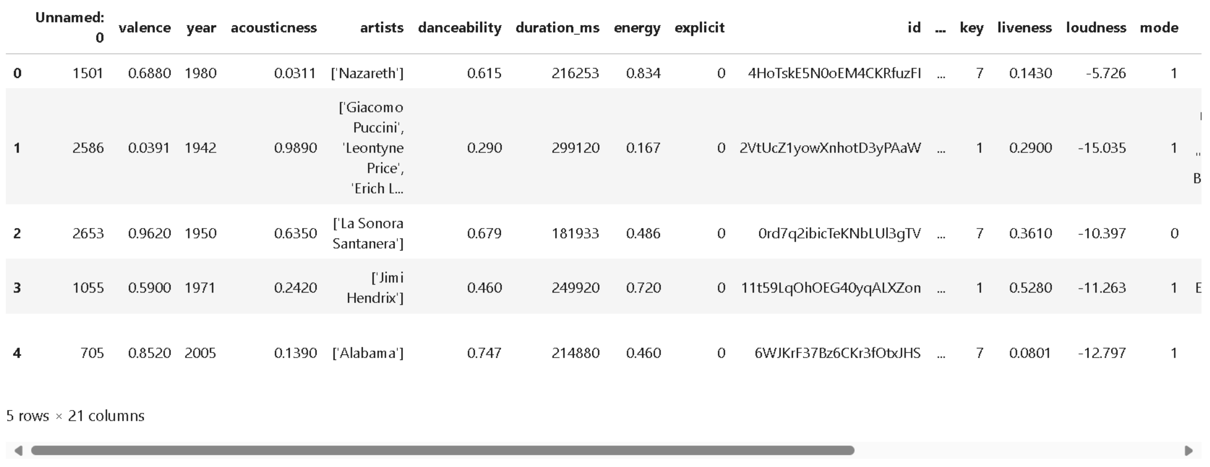


Code of the project:

Jupyter Notebook Code(.ipynb) :

Importing all the necessary libraries for data handling (Pandas) and load the Spotify Dataset.

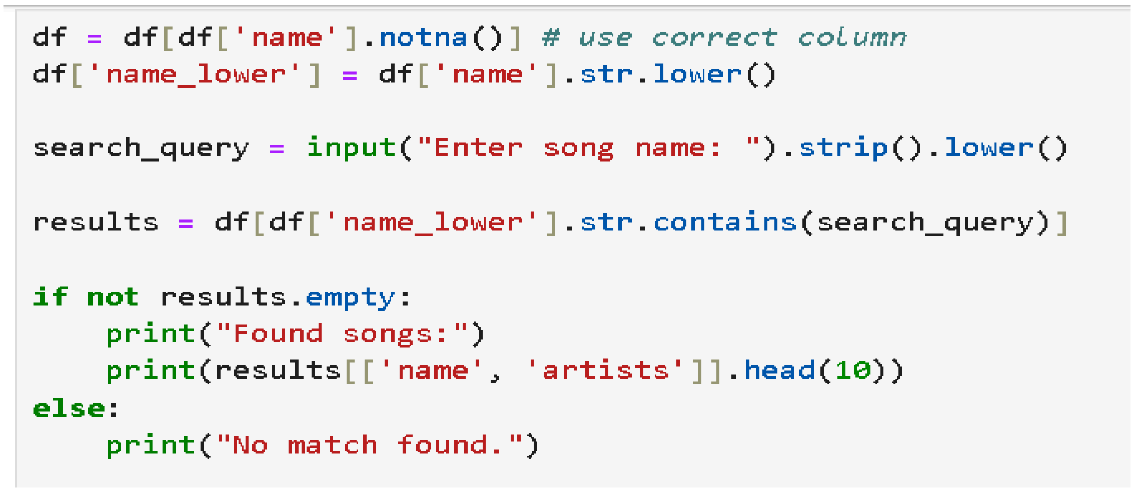
```python
# Load the dataset
import pandas as pd
df = pd.read_csv("clustered_df.csv")
df = df.sample(n=5000, random_state=42).reset_index(drop=True)
df.head()  # or df.head() for default 5 rows
```



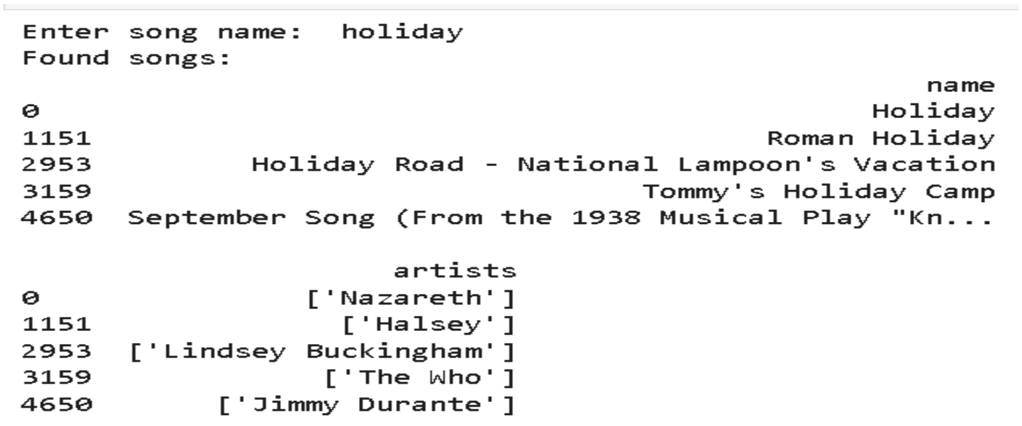
	Unnamed: 0	valence	year	acousticness	artists	danceability	duration_ms	energy	explicit	id	...	key	liveness	loudness	mode
0	1501	0.6880	1980	0.0311	['Nazareth']	0.615	216253	0.834	0	4HoTskE5N0oEM4CKRfuzFI	...	7	0.1430	-5.726	1
1	2586	0.0391	1942	0.9890	['Giacomo Puccini', 'Leontyne Price', 'Erich L...	0.290	299120	0.167	0	2VtUcZ1yowXnhotD3yPAaW	...	1	0.2900	-15.035	1
2	2653	0.9620	1950	0.6350	['La Sonora Santanera']	0.679	181933	0.486	0	0rd7q2ibicTeKNbLUl3gTV	...	7	0.3610	-10.397	0
3	1055	0.5900	1971	0.2420	['Jimi Hendrix']	0.460	249920	0.720	0	11t59LqOhOEG40yqALXZon	...	1	0.5280	-11.263	1
4	705	0.8520	2005	0.1390	['Alabama']	0.747	214880	0.460	0	6WJKrF37Bz6CKr3fOtxJHS	...	7	0.0801	-12.797	1

5 rows × 21 columns

Remove Missing Song Names,Create a Lowercase Version of Song Names,Take User Input for a Song Name,Find Songs that Match the Search,Display Results in notebook



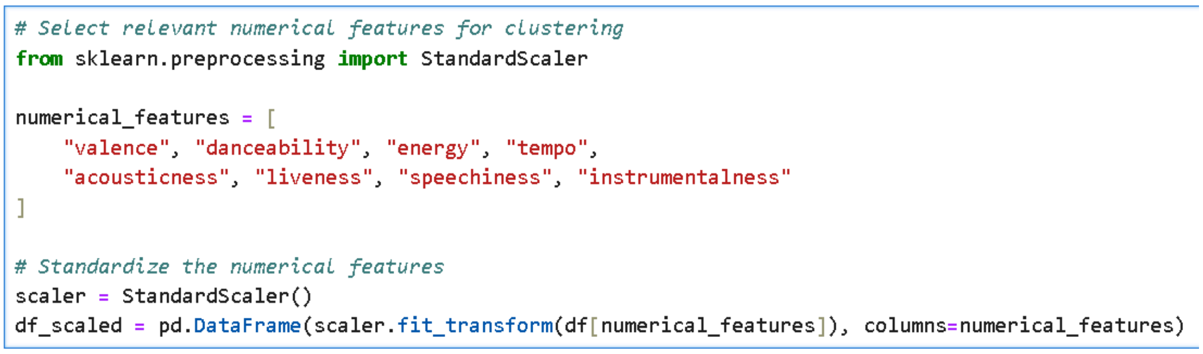
```python
df = df[df['name'].notna()] # use correct column
df['name_lower'] = df['name'].str.lower()

search_query = input("Enter song name: ").strip().lower()

results = df[df['name_lower'].str.contains(search_query)]

if not results.empty:
    print("Found songs:")
    print(results[['name', 'artists']].head(10))
else:
    print("No match found.")
```



```
Enter song name:  holiday
Found songs:
                                                    name
0                                                Holiday
1151                                       Roman Holiday
2953          Holiday Road - National Lampoon's Vacation
3159                                Tommy's Holiday Camp
4650  September Song (From the 1938 Musical Play "Kn...

                     artists
0               ['Nazareth']
1151              ['Halsey']
2953  ['Lindsey Buckingham']
3159             ['The Who']
4650       ['Jimmy Durante']
```

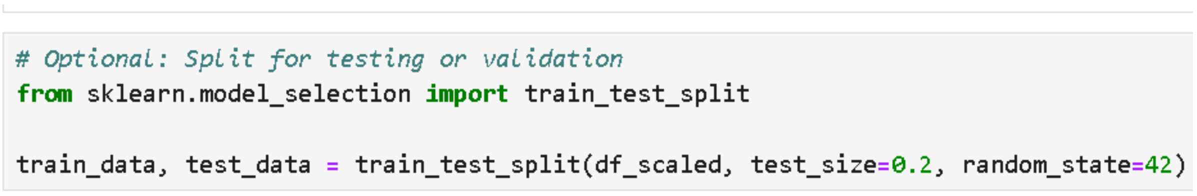
This code selects key numerical audio features like energy, danceability, and tempo, which describe the characteristics of each song. It then uses StandardScaler to standardize these features so they have a mean of 0 and standard deviation of 1, ensuring fair contribution during clustering.

Features and Scaling



```python
# Select relevant numerical features for clustering
from sklearn.preprocessing import StandardScaler

numerical_features = [
    "valence", "danceability", "energy", "tempo",
    "acousticness", "liveness", "speechiness", "instrumentalness"
]

# Standardize the numerical features
scaler = StandardScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df[numerical_features]), columns=numerical_features)
```

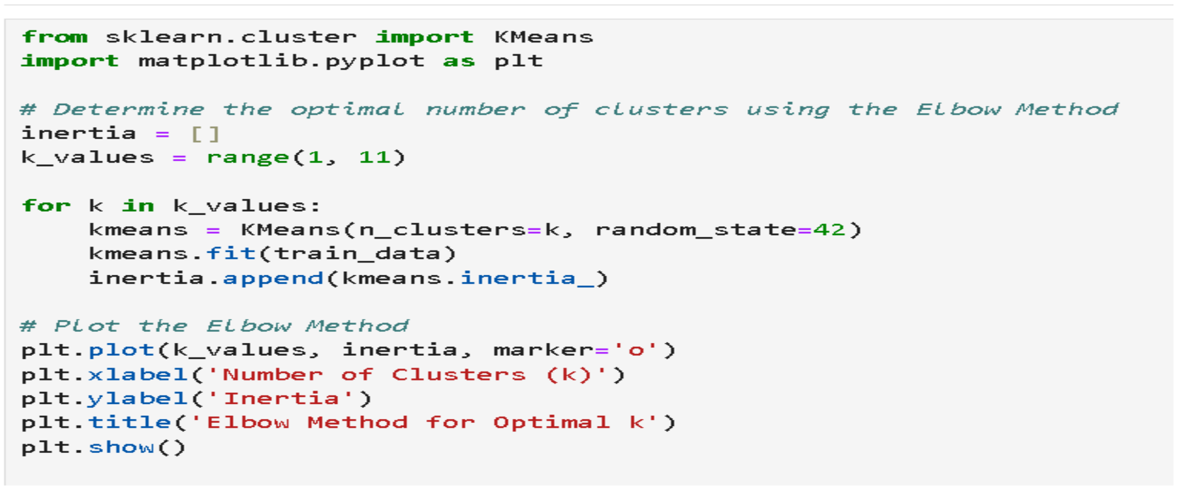
This code uses train_test_split from sklearn.model_selection to randomly split the scaled dataset into training and testing sets. It assigns 80% of the data to train_data and 20% to test_data, with random_state=42 to ensure the split is reproducible.



```
# Optional: Split for testing or validation
from sklearn.model_selection import train_test_split

train_data, test_data = train_test_split(df_scaled, test_size=0.2, random_state=42)
```

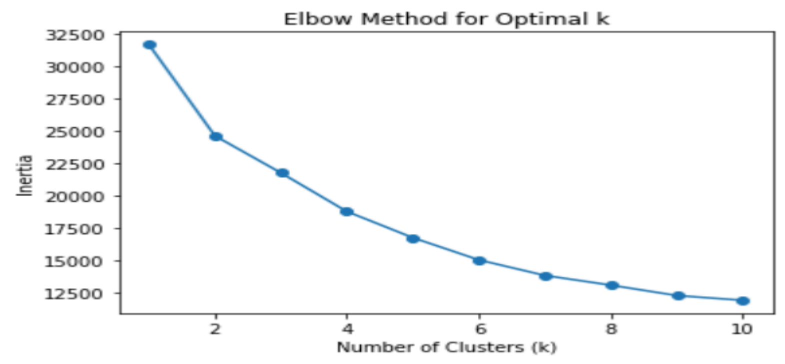
This code uses the Elbow Method to find the optimal number of clusters (k) for KMeans clustering:

1. It loops through values of k from 1 to 10, fits a KMeans model for each, and records the inertia (a measure of how tightly grouped the clusters are).
2. It then plots k vs. inertia — the "elbow point" (where the curve bends) suggests the ideal number of clusters, balancing between low inertia and not overfitting with too many clusters.



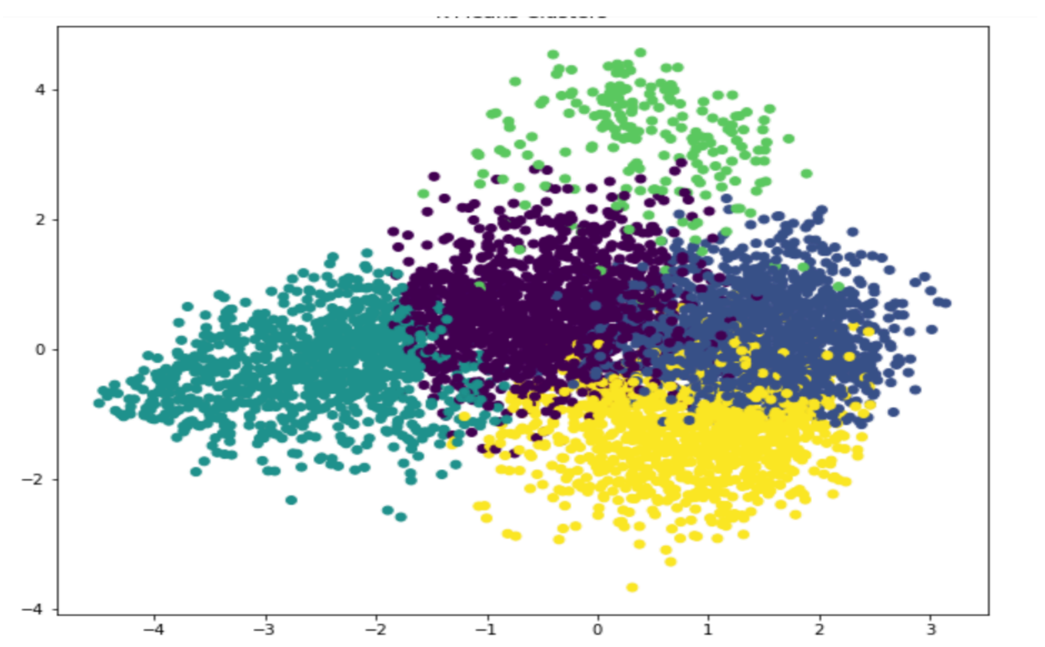
```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Determine the optimal number of clusters using the Elbow Method
inertia = []
k_values = range(1, 11)

for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(train_data)
    inertia.append(kmeans.inertia_)

# Plot the Elbow Method
plt.plot(k_values, inertia, marker='o')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal k')
plt.show()
```

In this graph, the "elbow" or point of maximum curvature appears around k = 4.
That means 4 clusters is likely a good choice, as adding more clusters beyond that gives diminishing returns in reducing inertia.

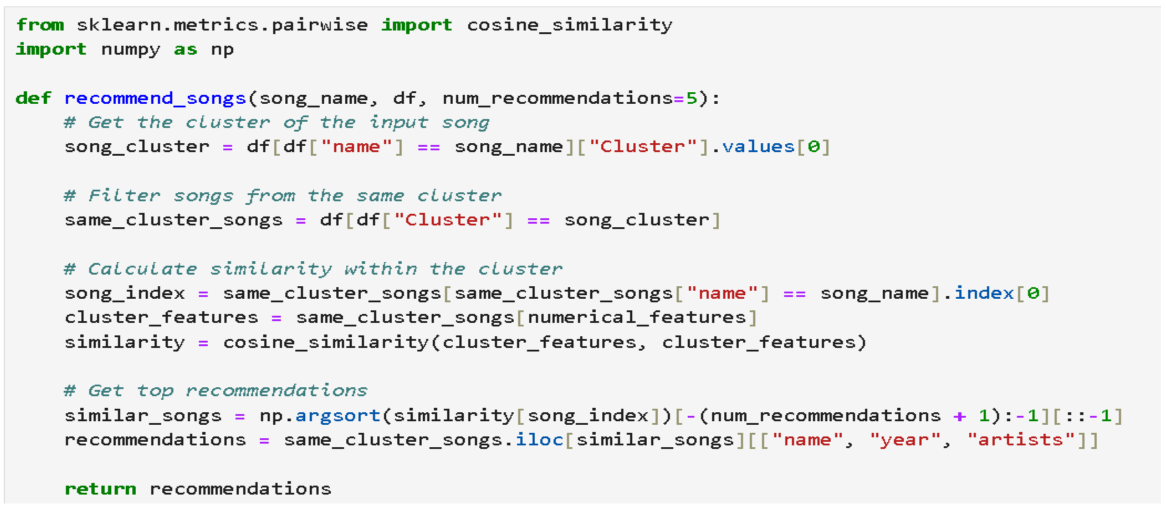


This code performs K-Means clustering and visualizes the clusters using PCA (Principal Component Analysis).PCA reduces the high-dimensional data to 2D for easy visualization, and plt.scatter() plots the points colored by cluster using the "viridis" color map.The result is a 2D scatter plot that visually separates songs into distinct clusters based on their audio features.

```
# Apply K-Means clustering with optimal k (e.g., k=5)
optimal_k = 5
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
df["Cluster"] = kmeans.fit_predict(df_scaled)

# Visualize clusters using PCA
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
pca_result = pca.fit_transform(df_scaled)

plt.figure(figsize=(10,8))
plt.scatter(pca_result[:, 0], pca_result[:, 1], c=df["Cluster"], cmap="viridis")
plt.title("K-Means Clusters")
plt.show()
```

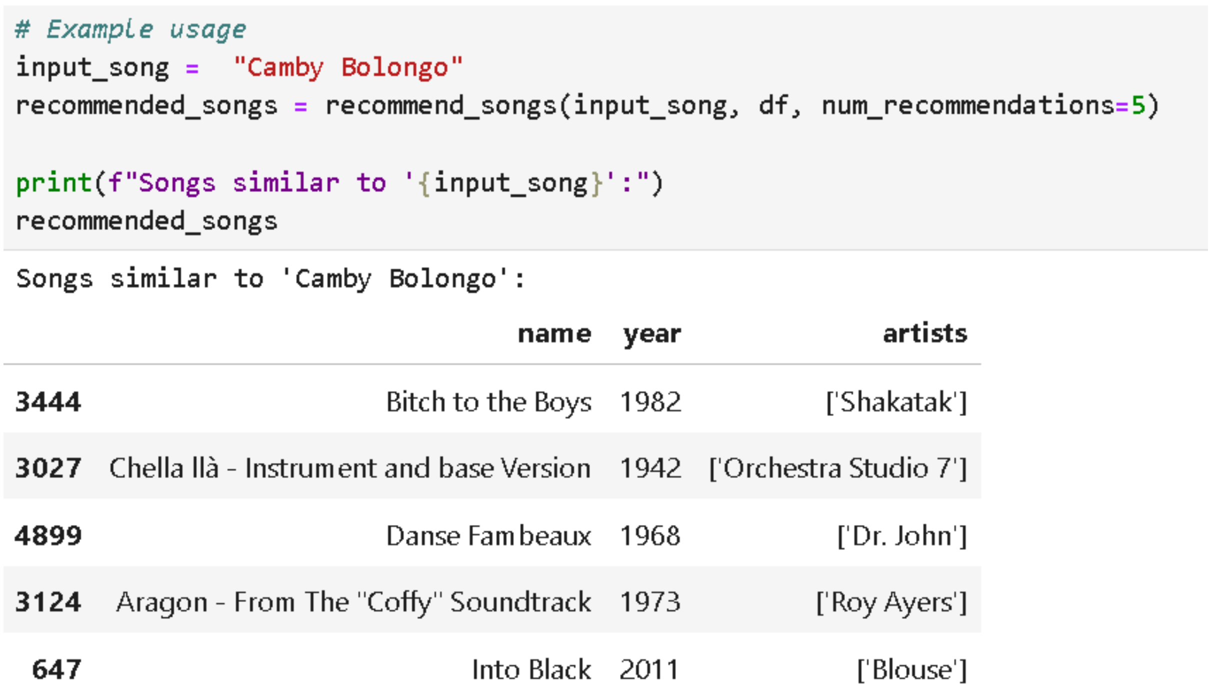


Recommendation system with K-means: It finds the cluster of a given song, filters other songs from that cluster, calculates similarity using numerical features, and returns the top most similar songs as recommendations.



```python
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np

def recommend_songs(song_name, df, num_recommendations=5):
    # Get the cluster of the input song
    song_cluster = df[df["name"] == song_name]["Cluster"].values[0]

    # Filter songs from the same cluster
    same_cluster_songs = df[df["Cluster"] == song_cluster]

    # Calculate similarity within the cluster
    song_index = same_cluster_songs[same_cluster_songs["name"] == song_name].index[0]
    cluster_features = same_cluster_songs[numerical_features]
    similarity = cosine_similarity(cluster_features, cluster_features)

    # Get top recommendations
    similar_songs = np.argsort(similarity[song_index])[-(num_recommendations + 1):-1][::-1]
    recommendations = same_cluster_songs.iloc[similar_songs][["name", "year", "artists"]]

    return recommendations
```

For exaple:This final output shows a successful music recommendation system based on clustering.Camby Bolongo" is the reference track for which we want similar recommendations.
K-Means groups songs by feature similarity (e.g., tempo, energy, etc.).
PCA reduces dimensionality for visualization.
Cosine similarity finds the most similar tracks within the same cluster.
The top 5 are returned (excluding the input).



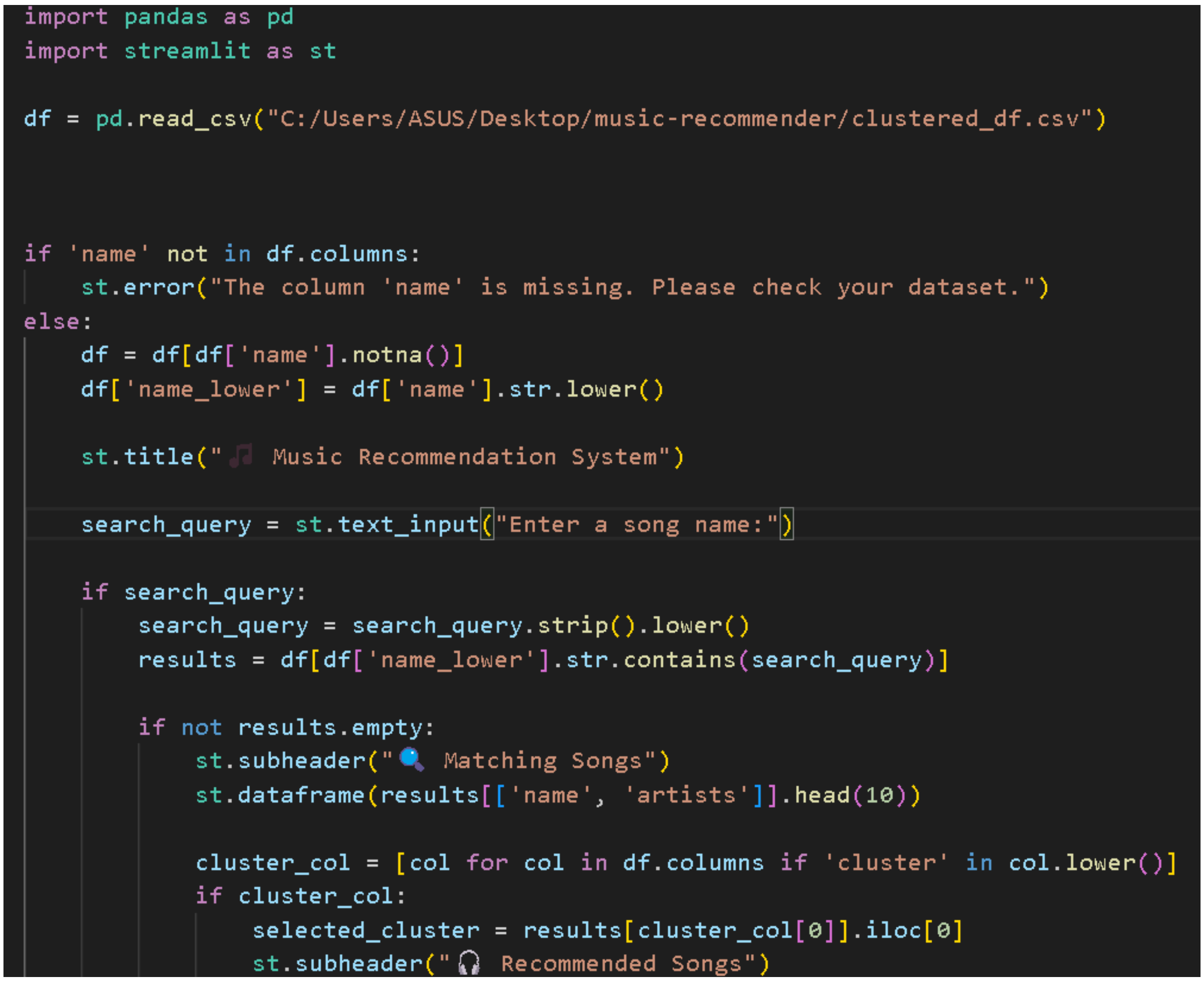
```python
# Example usage
input_song =  "Camby Bolongo"
recommended_songs = recommend_songs(input_song, df, num_recommendations=5)

print(f"Songs similar to '{input_song}':")
recommended_songs
```

```
Songs similar to 'Camby Bolongo':
```

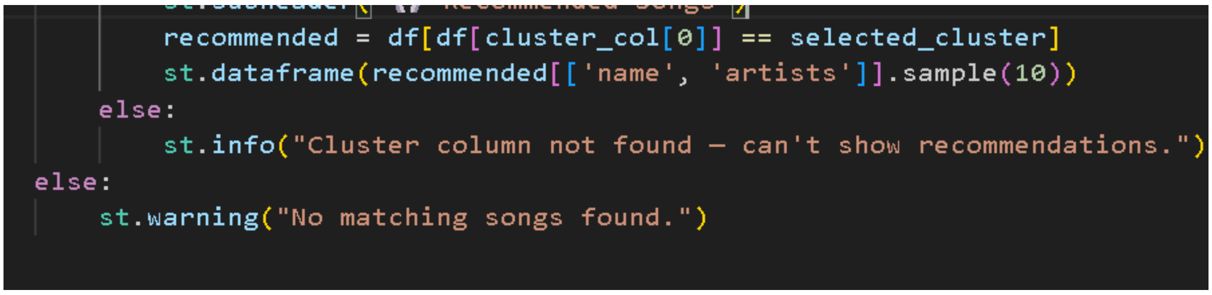
	name	year	artists
3444	Bitch to the Boys	1982	['Shakatak']
3027	Chella llà - Instrument and base Version	1942	['Orchestra Studio 7']
4899	Danse Fambeaux	1968	['Dr. John']
3124	Aragon - From The "Coffy" Soundtrack	1973	['Roy Ayers']
647	Into Black	2011	['Blouse']

Stramlit app for deployment:

This Streamlit app lets users:Search for a song by name.Find similar songs from the same cluster (based on precomputed K-means clustering).
Displays:Matching songs,10 recommended songs from the same cluster.It uses a CSV file with song data and clusters, and provides a simple, interactive music recommendation system.



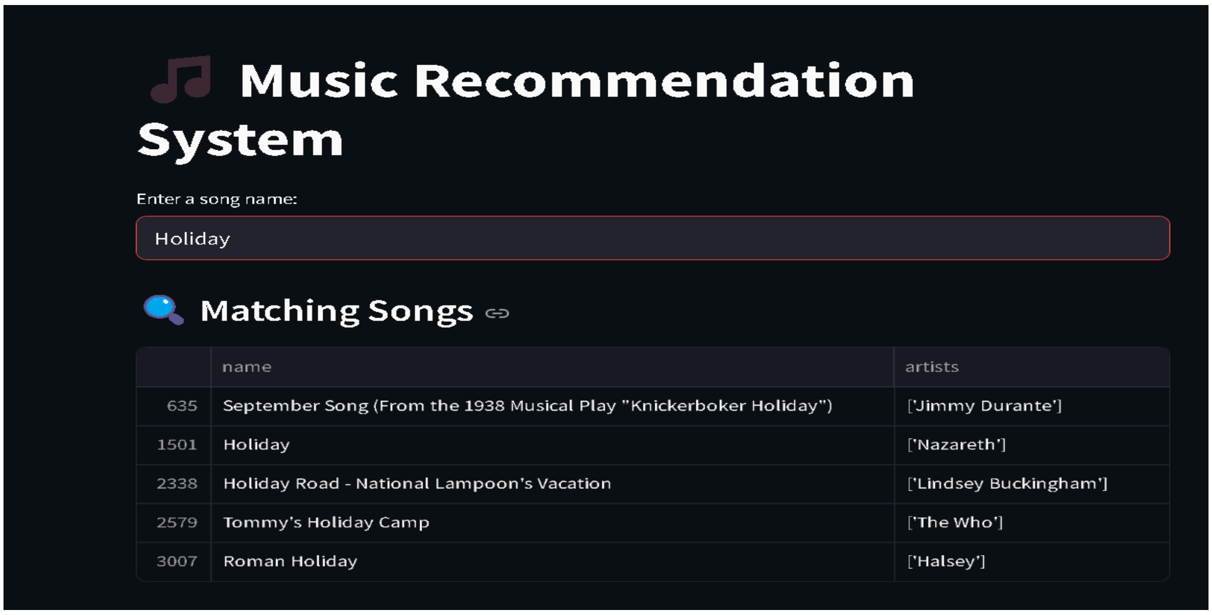
```python
import pandas as pd
import streamlit as st

df = pd.read_csv("C:/Users/ASUS/Desktop/music-recommender/clustered_df.csv")



if 'name' not in df.columns:
    st.error("The column 'name' is missing. Please check your dataset.")
else:
    df = df[df['name'].notna()]
    df['name_lower'] = df['name'].str.lower()

    st.title("🎵 Music Recommendation System")

    search_query = st.text_input("Enter a song name:")

    if search_query:
        search_query = search_query.strip().lower()
        results = df[df['name_lower'].str.contains(search_query)]

        if not results.empty:
            st.subheader("🔍 Matching Songs")
            st.dataframe(results[['name', 'artists']].head(10))

            cluster_col = [col for col in df.columns if 'cluster' in col.lower()]
            if cluster_col:
                selected_cluster = results[cluster_col[0]].iloc[0]
                st.subheader("🎧 Recommended Songs")
```



```python
                recommended = df[df[cluster_col[0]] == selected_cluster]
                st.dataframe(recommended[['name', 'artists']].sample(10))
            else:
                st.info("Cluster column not found — can't show recommendations.")
        else:
            st.warning("No matching songs found.")
```

<u>Result:</u>

- User Input: You entered a song title ("Holiday") in the input box.
- Search Results: The app found multiple songs with "Holiday" in their name (like "Holiday", "Roman Holiday", etc.).
- Matching Display: It lists matching songs with their name and artist(s) in a clean, scrollable table.



Music Recommendation System

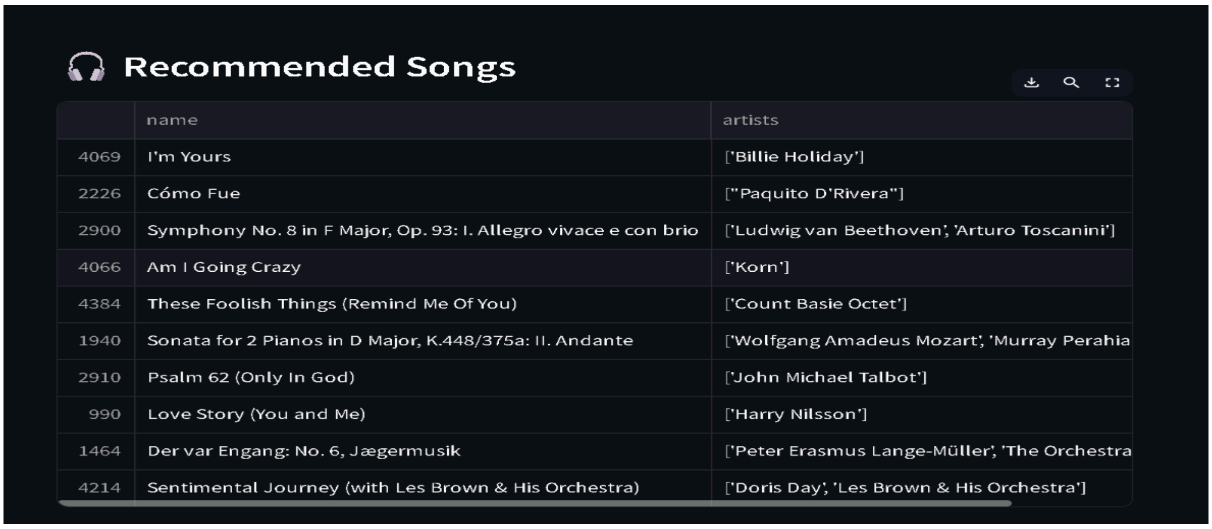
Enter a song name:

Holiday

Matching Songs

	name	artists
635	September Song (From the 1938 Musical Play "Knickerboker Holiday")	['Jimmy Durante']
1501	Holiday	['Nazareth']
2338	Holiday Road - National Lampoon's Vacation	['Lindsey Buckingham']
2579	Tommy's Holiday Camp	['The Who']
3007	Roman Holiday	['Halsey']

Recommended Songs:These aren't necessarily matching your search, but are suggested based on the input you gave or possibly your listening preferences or behavior.



Recommended Songs

	name	artists
4069	I'm Yours	['Billie Holiday']
2226	Cómo Fue	["Paquito D'Rivera"]
2900	Symphony No. 8 in F Major, Op. 93: I. Allegro vivace e con brio	['Ludwig van Beethoven', 'Arturo Toscanini']
4066	Am I Going Crazy	['Korn']
4384	These Foolish Things (Remind Me Of You)	['Count Basie Octet']
1940	Sonata for 2 Pianos in D Major, K.448/375a: II. Andante	['Wolfgang Amadeus Mozart', 'Murray Perahia
2910	Psalm 62 (Only In God)	['John Michael Talbot']
990	Love Story (You and Me)	['Harry Nilsson']
1464	Der var Engang: No. 6, Jægermusik	['Peter Erasmus Lange-Müller', 'The Orchestra
4214	Sentimental Journey (with Les Brown & His Orchestra)	['Doris Day', 'Les Brown & His Orchestra']