

 **HINDUSTHAN INSTITUTE OF TECHNOLOGY**

**An Autonomous Institution**

**(Accredited with “A” Grade by NAAC) (Approved by AICTE, New Delhi, Affiliated to Anna University, Chennai,**

Valley Campus, Pollachi Main Road, Coimbatore 641 032

# SPOTIFY MUSIC RECOMMENDATION

**A MINI PROJECT REPORT**

***Submitted by***

**720822103118 – PANDINILA M**

**720822103101 – MEERADHARSHINI S**

**22AD405 – MACHINE LEARNING**

**HINDUSTHAN INSTITUTE OF TECHNOLOGY**

NOVEMBER 2024

## BONAFIDE CERTIFICATE

Certified that this project report **“SPOTIFY MUSIC RECOMMENDATION”** is the bonafide work of **720822103118**-**PANDINILA M, 720822103101- MEERADHARSHINI S** who carried out the project work as a part of **22AD405 Machine Learning Laboratory**.

**SIGNATURE SIGNATURE**

Dr.A.JAMEER BASHA Ms.P.JEEVITHA

#### HEAD OF THE DEPARTMENT FACULTY INCHARGE

**Submitted for the viva voce held on \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Examiner 1**

### ABSTRACT

This project presents a music recommendation system based on K-Means clustering, which groups songs with similar musical features such as tempo, popularity, loudness, and genre. The dataset includes attributes like song\_id, genre, tempo, popularity, and loudness. To prepare the data, the categorical genre feature is encoded into numerical values, and numerical features are normalized using StandardScaler. The K-Means algorithm is applied to cluster the songs into five distinct groups based on their characteristics.

The system provides song recommendations by identifying songs from the same cluster as a given song. By using clustering techniques, the model suggests songs with similar audio features, offering users personalized recommendations without requiring historical data or explicit feedback. The results are visualized through pairplots, which highlight the relationships between the features and the clustering process. This approach offers an innovative, data-driven solution to personalized music recommendations, enhancing user experience on streaming platforms like Spotify. This clustering-based approach enables efficient, genre-agnostic music recommendations by analyzing intrinsic audio features, making it adaptable to a wide variety of music tastes

**INTRODUCTION**

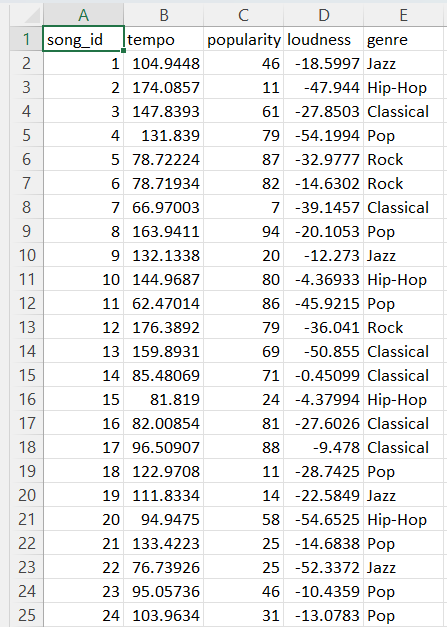
With the ever-increasing popularity of music streaming platforms like Spotify, providing users with personalized music recommendations has become a crucial feature. Traditional recommendation systems often rely on collaborative filtering or contentbased filtering methods. However, this project aims to explore the use of unsupervised learning, specifically clustering, to group songs based on their audio characteristics and generate music recommendations. K-Means clustering is employed to partition songs into clusters based on attributes such as tempo, loudness, popularity, and genre. These clusters are used to recommend songs similar to the one being played by a user.

### DATASET DESCRIPTION

The dataset used for this project contains various attributes of music tracks. The key columns include:

* song\_id: Unique identifier for each song.
* genre: The musical genre of the song (e.g., Pop, Rock, Classical).
* tempo: The tempo of the song in beats per minute (BPM).
* popularity: Popularity score, indicating how widely the song is liked by listeners.
* loudness: The loudness of the song in decibels (dB).

This dataset contains thousands of songs across multiple genres, and the features tempo, popularity, and loudness provide the necessary information to cluster the songs into similar groups.



### DATA PREPROCESSING TECHNIQUE

The raw dataset requires several preprocessing steps before applying clustering algorithms:

**Categorical Encoding**: The genre column, which is categorical, is converted into numerical values using the cat.codes method. This step allows us to treat the genre as a numerical feature for clustering.

**Code**: songs\_data['genre'] = songs\_data['genre'].astype('category').cat.codes

**Normalization**: The numerical features such as tempo, popularity, and loudness are normalized using the StandardScaler. Normalization ensures that the features are on a similar scale, which is important for the performance of the K-Means algorithm.

**Code** : scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

The preprocessing steps are crucial for the success of the clustering algorithm, as they prepare the data in a form that can be fed into the model.

**ALGORITHM USED**

The K-Means clustering algorithm is used for grouping similar songs based on their audio features. K-Means is an unsupervised machine learning algorithm that partitions data into a predefined number of clusters. It minimizes the variance within each cluster by assigning each data point (song) to the nearest centroid and then adjusting the centroids iteratively.

The steps followed in this project are:

**Feature Selection**: The features tempo, popularity, loudness, and genre are selected for clustering

**Data Normalization**: The data is normalized using StandardScaler to ensure each feature contributes equally.

**K-Means Clustering**: The K-Means algorithm is applied to partition the songs into 5 clusters.

**Cluster Visualization**: The clusters are visualized using a pairplot to show the relationship between the features.

**Song Recommendation**: A function is implemented to recommend songs from the same cluster as a given song.

### CODE IMPLEMENTATION

# Import required libraries

import pandas as pd

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

import seaborn as sns

import matplotlib.pyplot as plt

# Load the dataset

songs\_data = pd.read\_csv('/content/songs\_data.csv')

# Convert genre to numerical format for clustering

songs\_data['genre'] = songs\_data['genre'].astype('category').cat.codes

# Select features for clustering

X = songs\_data[['tempo', 'popularity', 'loudness', 'genre']]

# Normalize the data scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Initialize and fit KMeans

kmeans = KMeans(n\_clusters=5, random\_state=42) songs\_data['cluster'] = kmeans.fit\_predict(X\_scaled)

# Visualize clusters using a pairplot

sns.pairplot(songs\_data, hue='cluster', vars=['tempo', 'popularity', 'loudness']) plt.show()

# Print centroids of each cluster

print("Cluster Centers (centroids):") print(kmeans.cluster\_centers\_)

# Recommendation Function

def recommend\_songs(song\_id, n\_recommendations=5):

# Find the cluster of the given song\_id

song\_cluster = songs\_data.loc[songs\_data['song\_id'] == song\_id, 'cluster'].values[0]

# Recommend songs from the same cluster

recommended\_songs=songs\_data[songs\_data['cluster']==song\_cluster].

sample(n\_recommendations)

print(f"Recommendations based on SongID {song\_id}:")

print(recommended\_songs[['song\_id','tempo','popularity','loudness','genre'])

# Example: Get recommendations for song\_id 10

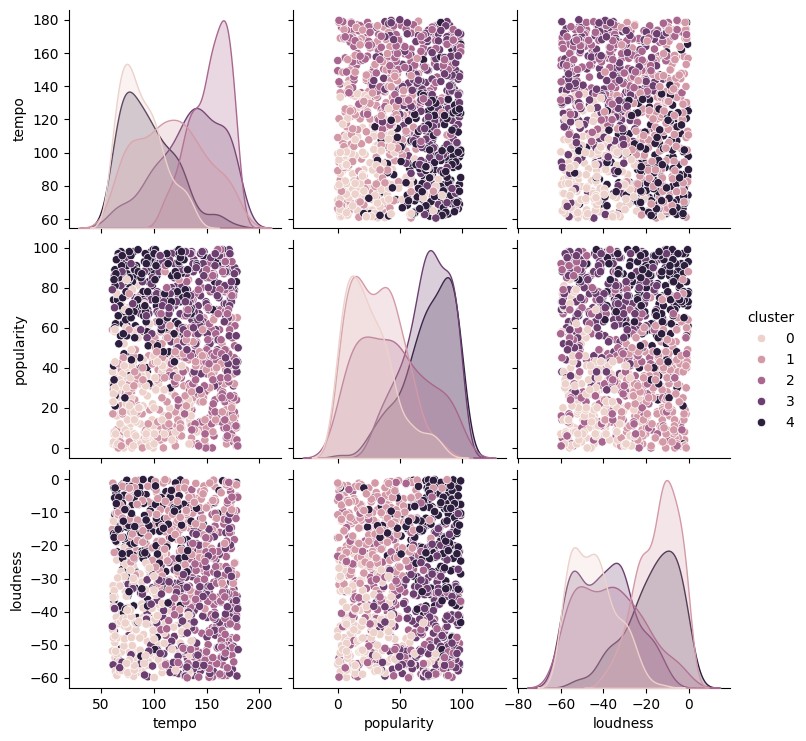
recommend\_songs(10)

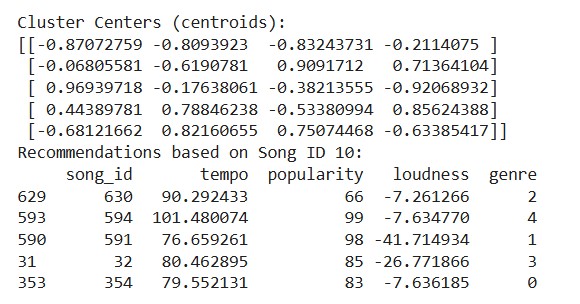
**RESULT**

The results of the clustering are visualized using a pairplot. The pairplot shows the relationships between the features tempo, popularity, and loudness, with each cluster represented by a different color.

**Cluster Visualization**:

This pairplot reveals that songs within the same cluster share similar values for the selected features. The K-Means algorithm successfully groups songs into distinct clusters, with each cluster representing a group of songs with similar musical characteristics.





### CONCLUSION

The K-Means clustering algorithm successfully clustered songs based on their audio features, allowing for the generation of song recommendations. The visualization of the clusters through pairplots revealed clear patterns in how the songs were grouped, with each cluster containing songs with similar tempo, loudness, and popularity. By recommending songs from the same cluster, the system provides personalized music suggestions to users based on their musical preferences.

Although this approach offers an efficient recommendation system, future improvements could involve experimenting with other clustering techniques or incorporating additional features like artist, mood, or user behavior data.

**REFERENCES**

1.Jolliffe, I.T. (2002) Principal Component Analysis. Springer-Verlag.

2.Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research, 12, 2825-2830.