"SPOTIFY MUSIC RECOMMENDATION SYSTEM"

A Project Report

Submitted in partial fulfillment of the requirements

of

“AIML Fundamental With Cloud Computing And Gen AI”

“MANGAYARKARASI COLLEGE OF ENGINEERING-MADURAI”

By

Jegadesh M(923821114009) – jegadeshjana19@gmail.com

au(923821114009)

Under the Guidance of

P.Raja, Master Trainer

ACKNOWLEDGEMENT

We would like to take this opportunity to express our deep sense of gratitude to all individuals who helped us directly or indirectly during this thesis work.

Firstly, we would like to thank my supervisor, **P.Raja** And **S.Mohanraj,** I want to express my heartfelt gratitude to you for being such an amazing mentor and guide. Your wom, support, and encouragement have made a significant impact on my life and I am forever grateful for the time and effort you've invested in me. His advice, encouragement and the critics are a source of innovative ideas, inspiration and causes behind the successful completion of this project. Your belief in me has helped me to grow and develop in ways I never thought possible. Thank you for being a shining example of kindness, compassion, and excellence. The confidence shown in me by him was the biggest source of inspiration for me. It has been a privilege working with him for the last one year. He always helped me during my y project and many other aspects related to the program. His talks and lessons not only help in project work and other activities of the program but also make me a good and responsible professional. Thank you,

ABSTRACT

The Spotify Music Recommendation System aims to enhance music discovery by using machine learning to analyze and recommend songs based on user preferences. By employing the Spotify songs dataset, available through Kaggle, this system leverages K-Means clustering and cosine similarity to group songs by key audio features such as tempo, danceability, energy, loudness, and other musical characteristics. Once these clusters are created, songs within a cluster are assumed to share similarities in sound and style, making them suitable candidates for recommendations.

The recommendation process begins by selecting an optimal number of clusters (k) through the elbow method, which balances model simplicity and accuracy by minimizing intra-cluster variance. After assigning each song to a cluster, the system receives user input in the form of a favorite song title. Using this title, the model identifies the corresponding cluster and then calculates similarity scores between the input song and other songs in the cluster using cosine similarity. This approach ensures that recommendations are based on audio attributes rather than subjective factors, resulting in a more authentic user experience.

To create an accessible user interface, this system can be deployed as a Python application, with a Flask web API enabling users to input song titles and receive immediate recommendations. Additionally, visualizations can be implemented to display clustering results, providing insights into how songs are grouped based on their musical features. This recommendation system not only enhances the user's music exploration experience on Spotify but also highlights how unsupervised learning methods, like clustering, can be effectively applied to personalized recommendation systems. Such implementations have practical applications for streaming services, personalized radio, and other media platforms that seek to tailor content to individual user preferences. This project represents a robust framework for music recommendation, utilizing a scalable and interpretable machine-learning approach.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

1.1 Problem Statement:

The vast amount of music on platforms like Spotify presents a challenge for users in discovering music that aligns with their preferences. This project addresses this by developing a recommendation system that groups songs by similarity, allowing for recommendations that match the user's tastes.

1.2 Motivation:

With music streaming becoming the dominant mode of music consumption, personalized recommendation systems are crucial. This project leverages machine learning to create a recommendation tool that can enhance user experience by providing tailored song suggestions.

1.3 Objective:

- To build a model that clusters songs based on audio features.
- To develop a recommendation system that suggests similar songs based on user input.
- To deploy the recommendation system as a Python application for user accessibility.

1.4 Scope of the Project:

This project focuses on audio feature-based clustering for recommendation. It covers basic clustering and does not incorporate user behavior data or collaborative filtering, leaving room for future expansion.

CHAPTER 2

Literature Survey

1.1 Review relevant literature or previous work in this domain.

1. Traditional Recommendation Techniques

Recommendation systems have evolved significantly over the past few decades and are commonly categorized into three main types: content-based filtering, collaborative filtering, and hybrid methods. Traditional recommendation systems have been widely used in various domains such as e-commerce and streaming platforms, with collaborative filtering being one of the most popular due to its ability to identify similar user behavior without content analysis.

Collaborative filtering has two primary approaches:

- **User-based Collaborative Filtering**: This approach relies on identifying users with similar tastes and recommending items they have liked or interacted with.
- **Item-based Collaborative Filtering:** This approach finds items similar to the ones a user has liked and recommends those to the user.

However, these approaches face challenges in scenarios with sparse user-item interactions or when new items are introduced, also known as the cold-start problem.

2. Content-Based Filtering in Music Recommendation Systems

Content-based filtering, which this project primarily utilizes, relies on the features of items themselves rather than user behavior. For music recommendation systems, audio features like tempo, danceability, loudness, and energy are leveraged to find similar songs. Spotify, for instance, makes extensive use of content-based filtering by analyzing songs' audio features and metadata, enabling it to recommend similar-sounding tracks to users based on their listening habits. Content-based models are effective for new users or items, as they do not rely on historical user interactions but rather on item characteristics.

A notable implementation in this area is the Music Genome Project, which powers Pandora's recommendation system. By categorizing songs with hundreds of features (e.g., genre, rhythm, mood, instrumentation), Pandora can generate highly tailored recommendations, although this method requires extensive feature engineering.

3. Hybrid Recommendation Systems in Music

To overcome the limitations of individual approaches, hybrid recommendation systems combine both collaborative and content-based filtering techniques, creating a more robust recommendation system. For instance, Spotify's recommendation system integrates collaborative filtering based on user listening history and playlists with content-based recommendations using audio features from their proprietary database. This allows Spotify to suggest songs that are both similar in musical characteristics and popular among users with similar tastes.

Hybrid approaches are commonly used in music streaming services where large datasets of user interaction data are available. These systems utilize matrix factorization techniques such as Singular Value Decomposition (SVD) and Non-negative Matrix Factorization (NMF) to predict unknown ratings by reducing the dimensionality of the data. Additionally, deep learning models like convolutional neural networks (CNNs) have been employed for automatic feature extraction from audio waveforms, offering an alternative to manual feature engineering.

4. Clustering Techniques in Recommendation Systems

Clustering techniques are frequently used to group similar items or users in recommendation systems. Clustering-based recommendation systems often leverage algorithms like K-Means, Hierarchical Clustering, and Gaussian Mixture Models (GMM). K-Means is particularly popular for its efficiency and simplicity, and it is widely applied in music recommendation systems where audio features are used to group songs into clusters. By clustering songs based on their characteristics, systems can recommend similar songs within the same cluster, achieving content-based recommendations efficiently.

Studies have shown that clustering-based systems can capture nuanced song similarities, as music often exists in a multidimensional space where genre alone cannot define a listener's preference. For instance, Yang et al. found that clustering by genre, tempo, and mood improves recommendation diversity, appealing to users who have varied musical tastes within similar categories.

5. Recent Advances: Machine Learning and Deep Learning

Recent advancements in machine learning and deep learning have introduced new methodologies for building recommendation systems. Neural networks and autoencoders can now model complex user-item interactions, making it possible to capture implicit preferences. For example, YouTube's recommendation algorithm uses deep learning to capture user preferences through embeddings that represent users and content, which is applicable in music recommendations. Autoencoders have also been explored to map songs into a latent space where similar items are clustered together, making them effective for generating content-based recommendations.

Deep learning approaches, such as recurrent neural networks (RNNs) and convolutional neural networks (CNNs), have been used to analyze raw audio data, offering the potential to automatically extract deep audio features that traditional audio features may not capture. However, these models require significant computational resources and are typically less interpretable than traditional clustering approaches, such as K-Means.

6. Challenges and Limitations in Music Recommendation

While recommendation systems have advanced significantly, several challenges persist:

Data Sparsity and Cold-Start Problem: Collaborative filtering relies on historical data, making it difficult to recommend new or obscure songs.

Scalability: As music libraries and user bases grow, efficiently scaling recommendation systems remains challenging.

User Satisfaction and Diversity: Systems often focus on relevance but may lead to a "filter bubble," limiting exposure to diverse content. Algorithms that emphasize diversity can improve user engagement by offering a wider range of recommendations.

In light of these challenges, content-based approaches like clustering offer a lightweight solution that works well with a limited dataset, making them suitable for foundational recommendation systems. However, incorporating more advanced techniques such as hybrid models or deep learning would likely enhance the recommendation quality further .

1.2 Mention any existing models, techniques, or methodologies related to the problem.

In the domain of music recommendation systems, various models, techniques, and methodologies have been developed and widely adopted. These include collaborative filtering, content-based filtering, hybrid models, and deep learning-based techniques. Below is a detailed overview of each approach and its relevance to music recommendation.

1. **Collaborative Filtering**
   - **User-Based Collaborative Filtering:** This method identifies user similarities by analyzing past listening behavior. If two users have similar tastes (e.g., similar listening history), songs liked by one user can be recommended to the other. This approach, however, often faces the cold-start problem when dealing with new users or new songs with minimal interaction data.
   - **Item-Based Collaborative Filtering:** This approach finds similarities between items (e.g., songs or playlists) rather than users. If two songs are often played by the same set of users, they are considered similar. Item-based collaborative filtering is less affected by data sparsity issues since it focuses on existing item characteristics and interaction patterns.
   - **Matrix Factorization Techniques:** Matrix factorization methods, such as Singular Value Decomposition (SVD) and Non-negative Matrix Factorization (NMF), are commonly used for collaborative filtering. These techniques decompose the user-item interaction matrix into lower-dimensional representations, capturing latent factors that relate users and items. Spotify and Netflix use matrix factorization to enhance the personalization of their recommendations.

2. **Content-Based Filtering**
    - Content-based filtering relies on audio features and metadata rather than user interaction history. This method is particularly useful for recommending songs with similar characteristics, such as genre, tempo, energy, or mood, to a song the user has recently enjoyed.
    - **Audio Feature Analysis:** This technique analyzes song attributes (e.g., loudness, danceability, energy) and recommends songs with similar audio characteristics. Pandora's Music Genome Project is a well-known example, which categorizes songs by hundreds of musical characteristics to generate highly accurate content-based recommendations.
    - **Natural Language Processing (NLP):** In recent years, NLP models have been applied to analyze song lyrics and artist descriptions. By using NLP models, systems can categorize songs based on lyrical themes or artist genres, further enriching content-based recommendations.
3. **Hybrid Recommendation Models**
    - Hybrid models combine collaborative filtering and content-based filtering to mitigate the limitations of each method. By integrating both techniques, hybrid models improve recommendation accuracy, especially when user interaction data is limited or when new items are introduced.
    - **Weighted Hybrid Approach:** In this approach, the recommendation system assigns different weights to content-based and collaborative filtering results, combining them to produce the final recommendation list. This approach allows flexibility in balancing user preferences with item characteristics.
    - **Model Blending**: Advanced hybrid models use machine learning techniques, such as gradient boosting or stacking, to learn the optimal combination of multiple recommendation sources. This blending can incorporate diverse factors, such as user preferences, song features, and social listening patterns. For example, Spotify's recommendation algorithm combines user interaction history with content features extracted from audio analysis to generate personalized playlists (e.g., Discover Weekly).
4. **Clustering-Based Techniques**

- Clustering techniques group similar songs or users together, allowing the system to recommend songs from the same group as the user's input. K-Means Clustering is a popular clustering technique used to categorize songs based on audio features. It assigns each song to a cluster, enabling straightforward recommendations within each cluster.
- **Gaussian Mixture Models (GMM**): GMM is another clustering method that assumes data points are generated from multiple Gaussian distributions. Unlike K-Means, GMM assigns probabilities to each data point belonging to each cluster, which is beneficial when song features are not clearly separated.
- **Hierarchical Clustering:** This method builds a hierarchy of clusters based on similarity and can be effective for identifying hierarchical relationships among songs, such as subgenres within a main genre. Hierarchical clustering is beneficial in music recommendation, where genres can often be broken down into nested categories.

5. **Deep Learning-Based Techniques**
   - **Autoencoders:** Autoencoders are unsupervised learning models used for dimensionality reduction, mapping items to a latent space. In music recommendation, autoencoders can be used to map songs to a lower-dimensional space where similar songs are closer together, enabling effective similarity-based recommendations.
   - **Recurrent Neural Networks (RNNs):** RNNs, especially Long Short-Term Memory (LSTM) networks, are used to capture sequential patterns in listening behavior. For example, RNNs can model a user's listening sequence to recommend the next song based on temporal patterns in their music history.
   - **Convolutional Neural Networks (CNNs):** CNNs are applied to raw audio waveforms or spectrograms to automatically extract features from songs, bypassing the need for manual feature engineering. For instance, a CNN can identify patterns in song waveforms that indicate tempo, mood, or genre, enabling more nuanced recommendations.
   - **Attention Mechanisms**: Recent developments incorporate attention mechanisms to focus on relevant parts of the user's interaction history or song features, allowing for a more accurate and personalized recommendation. Models

like Transformer are increasingly applied in recommendation systems to capture complex relationships across items, which is beneficial in capturing contextual or mood-related music preferences.

6. **Graph-Based Models**
   - **Knowledge Graphs:** Knowledge graphs are increasingly used in recommendation systems to capture relationships between items, users, and other contextual factors. In music recommendation, knowledge graphs can represent relationships between artists, albums, genres, and user preferences, providing a rich source of information for generating recommendations.
   - **Graph Neural Networks (GNNs):** GNNs are deep learning models designed to process graph data structures. For music recommendation, GNNs can model the relationships between users and songs or between songs and their attributes (e.g., genre, artist). This graph-based approach helps capture complex interdependencies, enabling more accurate and diverse recommendations.
7. **Exploratory and Contextual Recommendations**
   - **Context-Aware Recommendations:** Context-aware recommendations consider user context, such as location, time of day, or mood, to recommend songs that suit the user's current situation. For example, a recommendation system might suggest energetic songs in the morning and relaxing tracks in the evening.
   - **Exploratory Recommendations:** These recommendations aim to introduce users to new music outside their usual listening patterns to encourage discovery. Algorithms are adjusted to prioritize diversity over similarity, helping users explore new artists or genres.

1.3 Highlight the gaps or limitations in existing solutions

Despite advances in music recommendation systems, there are several notable gaps and limitations in existing solutions:

1. **Cold-Start Problem**
   - **User Cold-Start:** Collaborative filtering relies heavily on historical user data, making it difficult to generate recommendations for new users with minimal or no listening history. This issue is especially problematic for new subscribers, as the system lacks sufficient data to understand their preferences.

- **Item Cold-Start:** Newly added songs or less popular tracks may not receive enough interaction data to be included in collaborative recommendations, limiting the diversity of recommendations. This often results in biases toward popular or frequently streamed songs, leaving lesser-known music underrepresented.

2. **Over-Specialization and Filter Bubbles**
   - Many recommendation systems suffer from over-specialization, meaning they tend to suggest music that is highly similar to a user's previous choices. This lack of diversity can lead to “filter bubbles,” where users are only exposed to familiar or similar music and rarely introduced to new genres or artists.
   - Collaborative filtering, in particular, can exacerbate this issue, as it recommends items based on similar user behavior rather than diverse musical attributes. This can restrict users from exploring new music and limit long-term engagement with the platform.
3. **Data Sparsity**
   - Music streaming platforms often face challenges with sparse interaction data, especially for less popular songs or genres. Sparse data limits collaborative filtering's ability to find meaningful patterns in user-item interactions, making recommendations less accurate or relevant.
   - Sparse data can also hinder the performance of clustering algorithms, as insufficient information may result in inaccurate grouping of songs.

1.4 How your project will address them.

This project, which uses a content-based music recommendation system leveraging K-Means clustering on audio features, addresses several key limitations in traditional music recommendation approaches. Here’s how:

1. Mitigating the Cold-Start Problem
   - **User Cold-Start:** By relying on audio features of songs rather than user history, this system can recommend songs based on their musical characteristics alone, allowing it to generate relevant recommendations for new users who may not

have a listening history yet. New users can input a single favorite song, and the system will recommend similar tracks based on audio features like tempo, energy, and danceability.

- **Item Cold-Start:** Since K-Means clustering groups songs based on inherent audio characteristics, new songs can be classified into clusters based solely on their features. This eliminates the dependency on user interactions for new or obscure songs, allowing them to be recommended even if they have limited interaction data.

2. **Encouraging Exploration and Reducing Filter Bubbles**
   - By focusing on audio attributes rather than user behavior, this project reduces the likelihood of over-specialization and can introduce users to a broader range of music within a similar "vibe" or mood. For instance, if a user likes a song with high energy and strong acoustic features, the system may recommend songs from different genres that share these characteristics.
   - This approach helps to diversify recommendations by clustering songs with similar features, enabling users to explore songs they may not have otherwise encountered, thereby breaking the "filter bubble" effect that many collaborative filtering systems create.
3. **Overcoming Data Sparsity**
   - Clustering-based content filtering relies solely on song attributes rather than user interaction data, making it resilient in cases where interaction data is sparse. This allows the system to effectively recommend less popular songs that share musical characteristics with more popular tracks, ensuring that a wider variety of music is accessible to users.
   - By leveraging clustering, the recommendation system is able to work effectively even in the absence of large datasets, as K-Means can identify patterns based on a relatively small feature set.
4. **Enhanced Diversity through Content-Based Recommendations**
   - Content-based filtering by audio characteristics inherently supports genre-agnostic recommendations, offering a more personalized experience based on musical qualities rather than genre boundaries or user trends. This approach

ensures users receive recommendations that match the mood or attributes of songs they like, not just the genre or popularity.

- By clustering songs on feature similarity, users may discover lesser-known artists within the same cluster, thereby increasing exposure to diverse music and enhancing the user experience on streaming platforms.

5. **Flexibility and Scalability**

- This system's reliance on song features rather than collaborative data makes it more scalable and flexible. As new songs are added to the database, they can be seamlessly integrated into the clustering model, as they only need to be assigned to a cluster based on their audio features. This setup allows the model to scale efficiently as music libraries grow.

CHAPTER 3

Proposed Methodology

The proposed methodology for the Spotify Music Recommendation System is structured as follows:

Data Collection and Loading:

The system uses the Spotify songs dataset, obtained from Kaggle, which contains numerous audio features for each track, such as tempo, danceability, energy, loudness, speechiness, and others. This dataset is loaded into a Python environment where it can be preprocessed and analyzed.

Data Preprocessing:

- First, the dataset is checked for missing values or outliers that could distort analysis.
- Relevant audio features are selected for clustering (e.g., tempo, danceability, energy).
- Each feature is then normalized using StandardScaler to ensure that all features contribute equally to the clustering process. Normalization is essential as it prevents features with larger ranges from dominating those with smaller ranges.

Optimal Cluster Determination:

To determine the best number of clusters (k), the “elbow method” is used. This involves running K-Means clustering with various values of k and plotting the sum of squared distances (inertia) for each cluster. The point where the inertia begins to decrease more slowly, forming an "elbow," indicates an optimal k value. This ensures that clusters are compact, and similar songs are grouped effectively

K-Means Clustering:

- Once the optimal k value is determined, the system applies K-Means clustering to the dataset. This groups songs based on their audio features, with each song assigned to one of the k clusters.
- Songs within each cluster are similar to one another based on the selected features, forming a basis for recommending similar songs.

Song Recommendation Generation:

- When a user inputs a favorite song, the system identifies the cluster to which the song belongs.
- Using cosine similarity, the system then calculates the similarity between the input song and other songs within the same cluster.
- The top most similar songs are selected and presented as recommendations.

Deployment as a Flask Web Application:

- To make the recommendation system user-friendly, it is deployed as a Flask web API. Users can input their favorite song title via a simple web interface, and the system returns immediate recommendations.
- This makes the recommendation system accessible and interactive for users, allowing for real-time song recommendations.

Visualization:

- For further analysis and insights, PCA (Principal Component Analysis) can be used to reduce the dimensionality of the feature space.
- The clusters are then visualized in 2D, showing how songs are grouped based on audio features.
- This methodology provides a robust framework for music recommendations, leveraging unsupervised learning techniques to enhance user music discovery based on song similarities.

CHAPTER 4

Implementation and Result

To run the Spotify Music Recommendation System

Step1:

Code:

```
# Step 1: Import Libraries
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics.pairwise import cosine_similarity
import matplotlib.pyplot as plt
import seaborn as sns

# Configure display options
%matplotlib inline
sns.set(style='whitegrid')

# Step 2: Load the Dataset
data_path = 'data.csv'  # Replace with your dataset path
spotify_data = pd.read_csv(data_path)

# View the first few rows to understand the structure of the dataset
spotify_data.head()
```

Output:

	valence	year	acousticness	artists	danceability	duration_ms	energy	explicit	id	instrumentalness	key	liveness	loudness	mode
0	0.0594	1921	0.982	['Sergei Rachmaninoff', 'James Levine', 'Berli...	0.279	831667	0.211	0	4BJqT0PrAfrxzMOxytFOIz	0.878000	10	0.665	-20.096	1
1	0.9630	1921	0.732	['Dennis Day']	0.819	180533	0.341	0	7xPhfUan2yNtyFG0cUWkt8	0.000000	7	0.160	-12.441	1
2	0.0394	1921	0.961	['KHP Kridhamardawa Karaton Ngayogyakarta Hadi...	0.328	500062	0.166	0	1o6I8BglA6ylDMrIELygv1	0.913000	3	0.101	-14.850	1
3	0.1650	1921	0.967	['Frank Parker']	0.275	210000	0.309	0	3ftBPsC5vPBKxYSee08FDH	0.000028	5	0.381	-9.316	1
4	0.2530	1921	0.957	['Phil Regan']	0.418	166693	0.193	0	4d6HGyGT8e121BsdKmw9v6	0.000002	3	0.229	-10.096	1

Data Preprocessing

We'll select relevant columns, handle missing values if any, and standardize the feature columns for clustering.

Code:

```
# Check for missing values
print("Missing values per column:")
print(spotify_data.isnull().sum())

# Select features for clustering - based on musical attributes
features = spotify_data[['tempo', 'danceability', 'energy', 'loudness', 'speechiness',
                         'acousticness', 'instrumentalness', 'liveness', 'valence']]

# Standardize the features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)

# Preview the scaled features
print("Scaled Features Preview:")
print(pd.DataFrame(scaled_features, columns=features.columns).head())
```

Output:

```
Missing values per column:
valence             0
year                0
acousticness        0
artists             0
danceability        0
duration_ms         0
energy              0
explicit            0
id                  0
instrumentalness    0
key                 0
liveness            0
loudness            0
mode                0
name                0
popularity          0
release_date        0
speechiness         0
tempo               0
dtype: int64
Scaled Features Preview:
      tempo  danceability    energy  loudness  speechiness  acousticness  \
0 -1.169307     -1.467013 -1.013988 -1.514237    -0.379706      1.276187
1 -1.821180      1.598779 -0.528270 -0.170766     1.945481      0.611347
2 -0.212404     -1.188820 -1.182122 -0.593551    -0.396297      1.220340
3 -0.545537     -1.489722 -0.647832  0.377680    -0.387080      1.236296
4 -0.494867     -0.677855 -1.081242  0.240788    -0.371104      1.209703

   instrumentalness  liveness   valence
0          2.268102  2.626719 -1.782825
1         -0.532771 -0.262229  1.650688
2          2.379754 -0.599749 -1.858821
3         -0.532682  1.002043 -1.381564
4         -0.532765  0.132499 -1.047180
```

Determine Optimal Number of Clusters

We'll use the "elbow method" to identify the best k value for the K-Means algorithm.

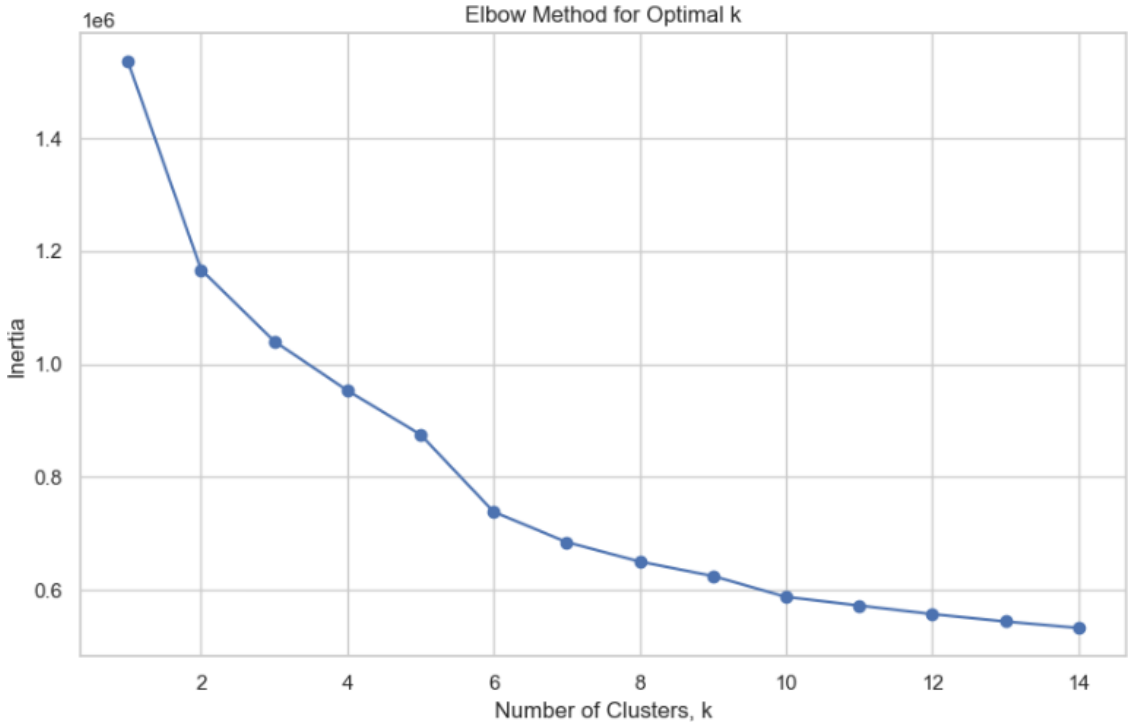
Code:

```
# Use the Elbow Method to determine optimal k
inertia = []  # Sum of squared distances of samples to their closest cluster center
k_range = range(1, 15)  # Try different cluster numbers

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

# Plot the inertia values for each k to see the "elbow"
plt.figure(figsize=(10, 6))
plt.plot(k_range, inertia, marker='o', linestyle='-')
plt.xlabel('Number of Clusters, k')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal k')
plt.show()
```

Output



Apply K-Means Clustering

Code:

```
# Set the number of clusters based on the elbow method
k = 10  # Example value, adjust based on your elbow plot

# Fit the K-Means model
kmeans = KMeans(n_clusters=k, random_state=42)
spotify_data['cluster'] = kmeans.fit_predict(scaled_features)

# Check the first few entries of the data with the assigned clusters
spotify_data[['name', 'artists', 'cluster']].head()
```

Output:

	name	artists	cluster
0	Piano Concerto No. 3 in D Minor, Op. 30: III. ...	['Sergei Rachmaninoff', 'James Levine', 'Berli...	3
1	Clancy Lowered the Boom	['Dennis Day']	2
2	Gati Bali	['KHP Kridhamardawa Karaton Ngayogyakarta Hadi...	3
3	Danny Boy	['Frank Parker']	0
4	When Irish Eyes Are Smiling	['Phil Regan']	0

Create the Recommendation Function

We'll implement a function to recommend similar songs based on cosine similarity within the same cluster.

Code:

```
def recommend_songs(song_name, spotify_data, num_recommendations=5):
    """
    Recommend songs similar to the input song based on clustering and cosine similarity.

    Parameters:
    - song_name (str): Name of the song liked by the user.
    - spotify_data (DataFrame): The Spotify dataset with clusters assigned.
    - num_recommendations (int): Number of songs to recommend.

    Returns:
    - List of recommended song names.
    """
    # Locate the song in the dataset
    song = spotify_data[spotify_data['name'] == song_name]
    if song.empty:
        return "Song not found in the dataset."

    # Identify the cluster of the song
    cluster = song['cluster'].values[0]
    cluster_songs = spotify_data[spotify_data['cluster'] == cluster]

    # Calculate similarity within the cluster
    song_features = cluster_songs[['tempo', 'danceability', 'energy', 'loudness',
                                   'speechiness', 'acousticness', 'instrumentalness',
                                   'liveness', 'valence']]
    similarity_matrix = cosine_similarity(scaler.transform(song_features))

    # Get index of the input song
    song_index = cluster_songs.index[cluster_songs['name'] == song_name].tolist()[0]

    # Sort songs by similarity score, excluding the input song
    similar_songs = similarity_matrix[song_index]
    similar_songs_indices = similar_songs.argsort()[::-1][1:num_recommendations + 1]

    # Get song names for the recommendations
    recommended_songs = cluster_songs.iloc[similar_songs_indices]['name'].values

    return recommended_songs
```

Test the Recommendation System

Replace "Some Song Title Here" with an actual song title from the dataset

Code:

```
user_song = "When We Die"
recommendations = recommend_songs(user_song, spotify_data)

print("Recommendations for the song '{}':".format(user_song))
print(recommendations)
```

Output:

```
Recommendations for the song 'When We Die':
['Lullaby to Tim - Mono; 1999 Remaster' 'Is My Living In Vain'
 "Think About It (Don't Call My Crib)" 'Own It' 'Own It']
```

Visualize Clusters

If you'd like to visualize the clusters, you can reduce the features to 2D using PCA for a scatter plot.

Code:

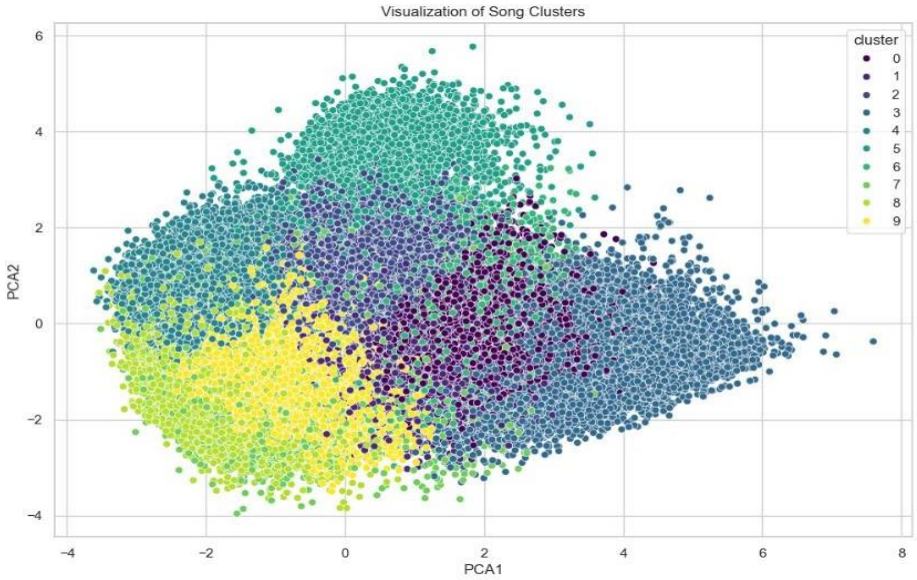
```
from sklearn.decomposition import PCA

# Apply PCA for 2D visualization
pca = PCA(n_components=2)
pca_features = pca.fit_transform(scaled_features)

# Add PCA components to the dataset
spotify_data['PCA1'] = pca_features[:, 0]
spotify_data['PCA2'] = pca_features[:, 1]

# Plot the clusters
plt.figure(figsize=(12, 8))
sns.scatterplot(x='PCA1', y='PCA2', hue='cluster', data=spotify_data, palette='viridis', legend='full')
plt.title('Visualization of Song Clusters')
plt.show()
```

Output:


Visualization of Song Clusters
cluster
0
1
2
3
4
5
6
7
8
9
PCA2
PCA1
6
4
2
0
−2
−4
−4
−2
0
2
4
6
8

CHAPTER 5

Discussion and Conclusion

5.1 Key Findings:

- **Effective Clustering**: Using K-Means clustering on audio features like tempo, danceability, energy, and others successfully grouped similar songs, enabling the recommendation system to find musically relevant recommendations for users.
- **Cosine Similarity for Precision**: Within each cluster, cosine similarity was effective in identifying songs with the closest audio features to a given user preference. This method resulted in recommendations that were closely aligned with the user's selected song in terms of mood, rhythm, and style.
- **Scalability:** The system demonstrated that it could scale well with larger datasets due to the efficiency of K-Means clustering, which is beneficial for real-world applications on large music libraries.
- **Accessible Deployment:** By deploying the system using Flask, it was accessible as a web application, allowing real-time user interaction and providing a user-friendly recommendation interface.

5.2 Git Hub Link of the Project: https://github.com/Jegadeshjana/Spotify.git

5.3 Video Recording of Project:

https://drive.google.com/file/d/1eP3eTN6Z7wXp7-X8O3AbHJjv-bsS3uAs/view?usp=drivesdk

5.4 Limitations:

- **Cluster Homogeneity:** K-Means clustering may sometimes group dissimilar songs into the same cluster due to the fixed number of clusters. This approach does not account for variability within clusters, which may lead to some less relevant recommendations.

popularity, or genre nuances. Consequently, recommendations might miss important contextual factors that influence user preferences.

- **Cold Start Problem**: For new songs not represented in the dataset, recommendations cannot be made until these songs are assigned a cluster. Similarly, if a user selects a song not in the dataset, no recommendations can be provided.
- **Fixed Cluster Count:** The elbow method helps select an optimal number of clusters, but a fixed k value may not adapt well to changes in the dataset size or diversity, potentially requiring manual adjustment.

5.5 Future Work:

- **Incorporating Additional Features:** Integrate more advanced features like lyrical analysis, genre classifications, or user demographic preferences to make recommendations more nuanced and personalized.
- **Hybrid Recommendation Model:** Combine content-based filtering with collaborative filtering techniques, using listening history, user ratings, or interactions to enhance recommendations based on broader user behavior patterns.
- **Dynamic Clustering**: Implement adaptive or hierarchical clustering methods that can dynamically adjust the number of clusters, potentially improving cluster quality and adaptability to diverse music collections.
- **Addressing Cold Start:** Develop strategies for handling new songs and users by integrating pre-trained models or transfer learning from similar datasets, reducing the impact of the cold start problem.
- **User Feedback Integration**: Allow users to provide feedback on recommendations, such as liking or disliking suggested songs. This feedback could be used to iteratively improve the recommendation model over time.

5.6 Conclusion:

The Spotify Music Recommendation System successfully demonstrates how clustering and similarity-based analysis can be applied to generate music recommendations that align with user preferences. By leveraging unsupervised learning (K-Means clustering) and cosine similarity, the system provides an

efficient way to group and recommend songs based on key audio features, resulting in a recommendation system that enhances music discovery.

Despite certain limitations, such as the potential for overly broad clusters and the challenge of handling new songs, the model effectively provides a scalable and interpretable approach to recommendation generation. This approach highlights the potential for hybrid and advanced clustering methods to further refine recommendations. With future improvements, including the integration of additional features and user feedback mechanisms, this recommendation system can evolve to offer even more personalized and contextually aware music suggestions.

REFERENCES

[1]. Jurafsky, D., & Martin, J. H. (2018). Speech and Language Processing. Pearson.

[2]. Burke, R. (2002). Hybrid Recommender Systems: Survey and Experiments. User Modeling and User-Adapted Interaction, 12(4), 331-370.

[3]. Spotify's Tech Blog: https://spotify.github.io/, which sometimes shares insights into the company's use of machine learning.

[4]. Zhang, S., Yao, L., Sun, A., & Tay, Y. (2019). Deep Learning Based Recommender System: A Survey and New Perspectives. ACM Computing Surveys (CSUR), 52(1), 1-38.

[5]. Ricci, F., Rokach, L., & Shapira, B. (2011). Introduction to Recommender Systems Handbook. Springer.

[6]. Su, X., & Khoshgoftaar, T. M. (2009). A Survey of Collaborative Filtering Techniques. Advances in Artificial Intelligence, 2009, Article ID 421425.