Project: Song Recommendation System

Introduction
This project aims to use Python libraries such as pandas, numpy, Scikit-learn, and SciPy to create a song recommendation system that suggests songs based on the input song. Relevant columns that contribute uniquely to a song's characteristics such as track ID, name, artist, popularity, album name, playlist genre, and name are chosen to train the system.

Data Source
The dataset was selected from Kaggle (https://www.kaggle.com/datasets/joebeachcapital/30000-spotify-songs), including about 30000 songs from Spotify API.

Data Cleaning and Wrangling
First, the relevant columns selected from the dataset seem sensible to train the system. The dataset was checked for null values, rows with null track names, artist or album names were dropped and then the dataset was checked for duplicate values.
To solve this issue, I decided to divide the dataset into two parts: one with track popularity values and playlist genre and the other with rest column values(name, id, artist, album name). The division is based on the fact that in the first group all the duplicate values matter whereas in the second group we can just use the column values form the first row that occurs.

I found the average value of track popularity and mode for playlist genre value and for the rest column values, the first value of the duplicates was kept. Finally, both groups were merged and we have a dataset to work with.

Method Used

To build the recommendation system, the dataset was first preprocessed using functions from the `sklearn` library. The string-based columns (e.g., `track_artist_name`, `playlist_genre`) were vectorized using the `TfidfVectorizer()` from `sklearn.feature_extraction.text`. This method transforms the text data into numerical vectors by calculating the Term Frequency-Inverse Document Frequency (TF-IDF) scores, ensuring that important and unique terms contribute more to the resulting vectors.

The dataset also included a numerical column (`track_popularity`), which was standardized using the `StandardScaler()` from `sklearn.preprocessing`. Standardization was necessary to bring this numerical feature onto the same scale as the vectorized text data, preventing it from dominating the similarity computations.

Next, the vectorized text data and the standardized numerical column were combined using `hstack()` from the `scipy.sparse` module to create a single matrix representing all features of each song. This matrix was then used to compute a similarity matrix using `cosine_similarity()` from `sklearn.metrics.pairwise`. Cosine similarity measures the angle between two vectors, making it an ideal metric for determining the closeness between songs based on both their genre and popularity.

For generating recommendations, the input song name was first matched to the closest entry in the dataset using the `.tolist()` method, which created a list of all song names. Once the closest match was identified, the similarity matrix was used to retrieve and rank all songs based on their similarity score to the input song. The songs were then sorted by their similarity scores, and the top 30 most similar songs were selected. Finally, a simple `for` loop was used to print the song titles and their corresponding artists.

Conclusion

This project provided valuable insight into the machine learning functionality available through Python, specifically using libraries like `sklearn` and `scipy`. It allowed me to explore how various preprocessing techniques and similarity measures can be applied to a recommendation system. In the future, I plan to test methods for evaluating the accuracy of this system and explore ways to further optimize and enhance the recommendation algorithm to develop more robust systems.