

**Project Title**

**Music Recommendation System**

**Team members:**

* NIRUPAM ABHIJITH [CB.EN.U4CSE21243] (Emotion Based)
* PARTHIV CHANDRAN[CB.EN.U4CSE21245](Spotify Playlist Based)

**Introduction to Hybrid Data Structures:**

The term "hybrid data structures" refers to the data structures that are created by combining the traits and capabilities of many data structures to address the needs of the present issue. They are made to minimize their downsides while maximizing the benefits of different data formats. The hybrid data structures provide us more adaptability by enabling us to handle difficulties effectively in dynamic (real-time) contexts, and they minimize complexity since each of these problems is dispersed and handled more efficiently. These hybrid data structures are tailored for specialized situations.

**Project’s objective:**

The primary objective of this study on a music recommendation system is to employ hybrid data structures to provide consumers personalized song recommendations. The system intends to analyze a user's playlist and extract important properties like popularity, danceability, energy, and pace using machine learning and the Spotify Web API. After being normalized, these characteristics are utilized to train a Nearest Neighbour model. When a user chooses a music, the algorithm uses the track's attributes to determine the closest Neighbour and provides a carefully selected selection of songs that are similar. This project aims to improve the music listening experience by recommending relevant and enjoyable music that match the user's taste and preferences by utilizing hybrid data structures and taking into account user preferences.

**Overview of the Hybrid Data Structure:**

In this music recommendation system project, a pandas DataFrame and a nearest neighbour model are combined to form a hybrid data structure. The DataFrame, which was imported from the 'tracks.csv' file, functions as a tabular structure to store and manage the music track data. It has several qualities that are utilised to propose songs, including popularity, danceability, energy, etc. The normalised track characteristics from the DataFrame are used by the Nearest Neighbours model, which was created using the sklearn package, to locate the closest neighbours based on how similar they are. This hybrid data structure enables effective track data storage, retrieval, analysis, and comparable track discovery based on feature proximity.

**Implementation Details:**

* **Importing Required Libraries:** The first lines of code imports the required libraries. Sklearn is used for data preparation and the NearestNeighbors algorithm, pandas is used for data manipulation, spotipy is a Python library for the Spotify Web API, and webbrowser is used to open a web browser to authenticate the Spotify user.
* **Setting Up Spotify API Credentials:** They need to log into their sptoify and access it in developer mode. After that, they need to create an API which will generate clientid, clientsecret, redirecturi.
* **Loading and Preparing Data:** The code reads a CSV file called 'tracks.csv' and inserts it into a pandas DataFrame. It chooses particular columns as the features to recommend. The characteristics are then normalized with MinMaxScaler to bring them within a certain range.
* **Training the Nearest Neighbors Model:** With n\_neighbors set to 5 (you may change this value as desired) and the algorithm set to "auto," a NearestNeighbors model is created. Then, using the normalized characteristics, the model is trained.
* **Authenticating and Initializing Spotify API:** The SpotifyClientCredentials object, which accepts the client ID and client secret as inputs, is used to authenticate users of the Spotify API. Spotipy generates a Spotify API object.Spotify, supplying the manager of client credentials.
* **Recommending Songs from a Playlist:** 'YOUR\_PLAYLIST\_ID' must be changed to the ID of the Spotify playlist from which you wish to propose songs. Using Spotify.playlist\_tracks(), the code fetches the tracks from the provided playlist. It uses the recommend\_songs() method to get a list of suggested songs for each track. Recommendations are printed and added to the recommended\_songs\_list if they are available.

**The trade-offs that were made during the implementation phase are:**

**Normalization**: In order to maintain the connections between the features while bringing them inside a certain range, it was decided to normalize the features using the MinMaxScaler. While normalization guarantees that each characteristic contributes fairly to the recommendations, it may also result in the loss of certain fine-grained information.

**Configuration of the Nearest Neighbours Model**: There was a trade-off between computational efficiency and recommendation accuracy when deciding on the number of neighbours (n\_neighbors) for the Nearest Neighbours Model. A lesser number of neighbours could produce suggestions that are less varied, but a higher number might make computing the nearest neighbours more expensive.

**User Interface**: To concentrate on functionality and proof of concept, the implementation may have reduced the user interface. To make the implementation reasonable within the provided scope, more complex functionality like user feedback integration, real-time updates, or advanced filtering options may have been traded off.

**GitHub Repo link**: github.com/parthivvv/PlaylistRecommender

**Practical Applications:**

Numerous real-world uses for the music recommendation system project exist that might improve consumers' musical listening experiences. Several potential uses include:

**Platforms for personalised music streaming:** The project may be linked with already-existing services for personalised music streaming, like Spotify or Apple Music, to provide customers personalised song recommendations based on their listening habits, interests, and chosen songs. Users may utilise this to find new music and artists that suit their personal likes.

**Playlist Curation:** Users may create and edit playlists with the help of the recommendation algorithm. Users may quickly add music that fit their preferred mood, genre, or topic to their playlists by receiving suggestions for comparable songs based on a chosen tune.

**Radio Stations and DJ Sets:** Using the system, radio stations or DJs can make playlists or sets that adhere to a certain aesthetic or tone. A music may be chosen by the DJ, and the system can then suggest more tracks that go well with it, ensuring that the audience has a seamless and pleasurable listening experience.

**Music Selection for Events or locations:** The recommendation system may be used to create playlists that are tailored to the desired ambience or audience preferences for events, parties, or locations where music selection is important. It may provide recommendations for songs that fit the desired speed, intensity level, or genre to produce a personalized and captivating musical environment.

**Music analysis and research:** The recommendation system may be used as a tool by musicologists, researchers, or data analysts to examine musical patterns, genre classifications, or musical resemblance. It can aid in the study and analysis of music by utilizing the system's capacity to locate comparable recordings based on attributes.

**Performance Analysis:**

**Preprocessing and Data Loading**

**Time Complexity**: Using pd.read\_csv() to load the tracks data from a CSV file normally takes O(n) time, where n is the number of rows in the file. The temporal complexity of preprocessing operations like feature selection and normalization is O(m), where m is the number of features that were picked.

**Space Complexity**: Because the data is kept in a pandas DataFrame and additional memory is needed for the normalized features, the space complexity for loading and preparing the data is O(n \* m).

**Training for the Nearest Neighbours Model:**

**Time Complexity**: Using a model, train the Nearest Neighbours algorithm.When k is the number of neighbours, d is the number of dimensions (chosen features), and n is the number of data points (tracks), fit() has an average time complexity of O(k \* d \* log(n)). With higher values of k and d, the temporal complexity may rise.

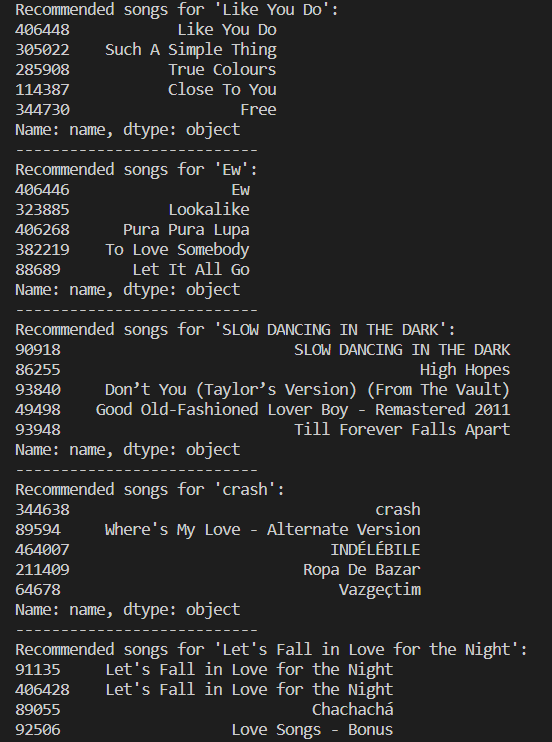
**Space Complexity**: Because the Nearest Neighbours model must keep the normalised feature vectors for each track in memory, its space complexity is O(n \* d).

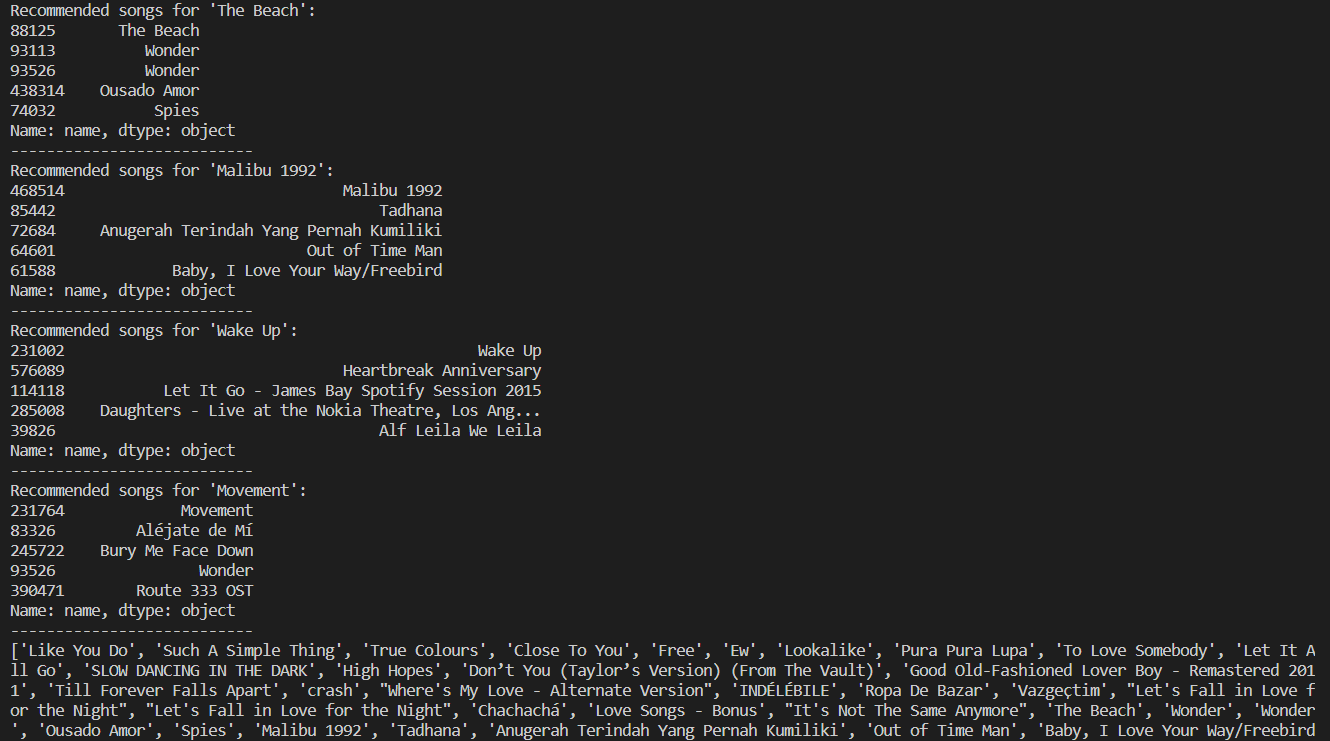
**Songs to Recommend for a Track:**

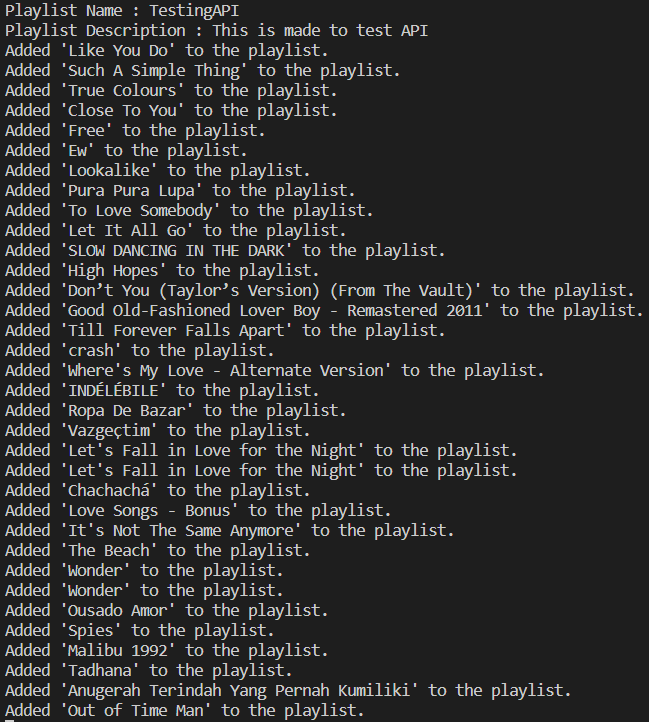
**Time Complexity:** Model.kneighbors()'s nearest neighbours search is used to get suggestions for a given track. To locate the k closest neighbours based on the track's characteristics, it takes O(k \* log(n)) of time. Additionally, the temporal complexity of getting the names of the suggested songs from the DataFrame is O(k), where k is the total number of suggested tracks.

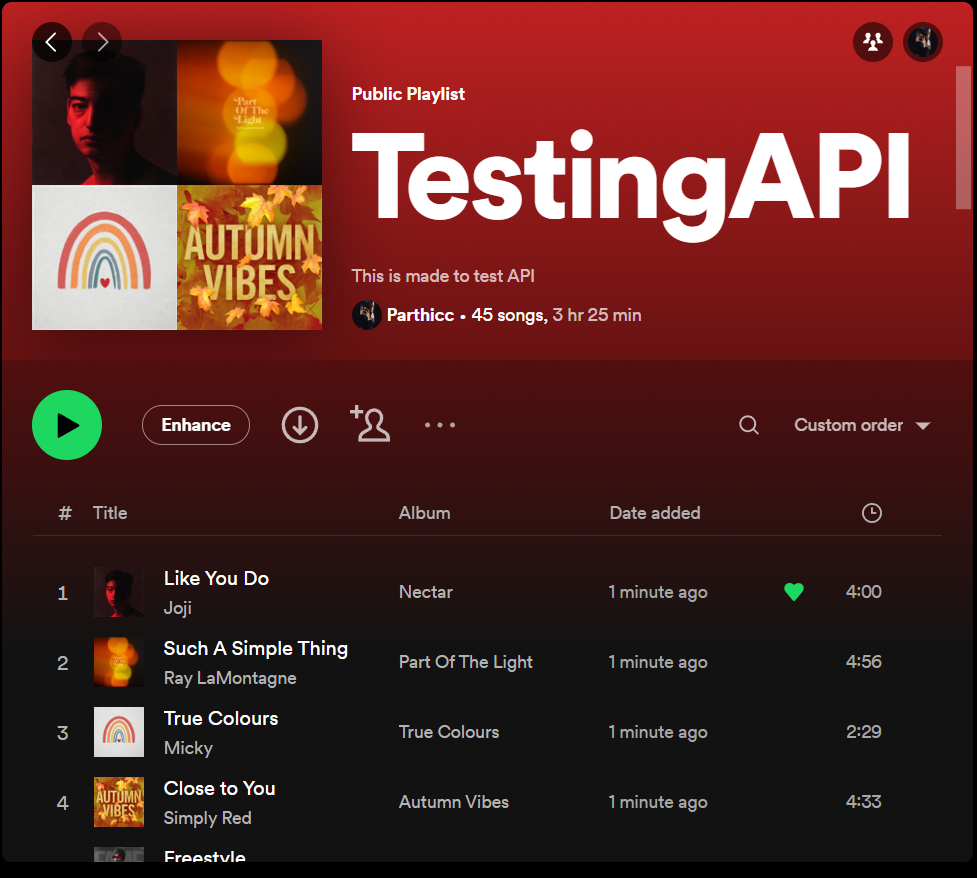
**Space Complexity:** Due to the suggested track IDs being maintained in a list, the space complexity for song recommendations is O(k).

**Experimental Evaluation:**









**Discussion:**

To provide individualized song recommendations, the music recommendation system project makes use of hybrid data structures, machine learning, and the Spotify API. The system detects related tracks based on feature closeness using a pandas DataFrame and a nearest neighbours model. It includes effective data pretreatment, loading, and interaction with the Spotify API for playlist construction and track information retrieval. Accuracy and complexity trade-offs were made to ensure prompt suggestions that accommodate a range of user preferences. The system's goal is to improve music listening by providing individualized recommendations that are in line with user preferences, encouraging music discovery, and increasing user pleasure.

**Conclusion:**

The music recommendation system project uses hybrid data structures, machine learning, and the Spotify API to deliver personalized song suggestions. The system uses a pandas DataFrame and a nearest neighbours model to identify similar tracks based on feature proximity. It provides efficient data pretreatment, loading, and Spotify API interface for creating playlists and retrieving track information. To guarantee timely suggestions that take into account a variety of user preferences, accuracy and complexity trade-offs were made. The system's objective is to enhance music listening by offering tailored suggestions based on user preferences, promoting music discovery, and raising user satisfaction.

**Improvements:** Considering adding user comments, collaborative filtering, and contextual suggestions to further improve the music recommendation algorithm. Integrate social characteristics for a customised experience, investigate hybrid recommendation strategies, and keep feeding fresh data into the model. Create algorithms for creating customised playlists and make use of cutting-edge machine learning methods. To evaluate the efficacy of various algorithms and features, do A/B testing. For access while on the go, expand the system to a mobile application. With these upgrades, the system will be more precise, interesting, and personalized, giving consumers a wonderful music discovery and listening experience.