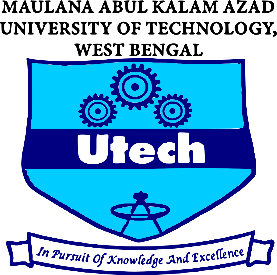
### PERSONALIZED MUSIC RECOMMENDATION SYSTEM

*A comprehensive project report has been submitted in partial fulfillment of the requirements for the degree of*

*Bachelor of Technology In*

*Computer Science and Engineering*

Maulana Abul Kalam Azad University of Technology, West Bengal

****

*Submitted By*

**RUPAM DEY**

Registration No:030947

Examination Roll No: 10000119014

AND

**KHUSAL DAS**

Registration No: 201000100120003 of 2020-21 Examination Roll No: 10000120076

*Under the Guidance of*

### MR. SUBHANJAN SARKAR

Assistant Professor,

Department of Computer Science & Engineering, Maulana Abul Kalam Azad University of Technology, WB

Nadia-741249, W.B., India

PERSONALIZED MUSIC RECOMMENDATION SYSTEM

###### DECLARATION OF ORIGINALITY AND COMPLIANCE OF ACADEMIC PROJECT

I hereby declare that this thesis entitled **“PERSONALIZED MUSIC RECOMMENDATION SYSTEM”** contains literature survey and original research work by the undersigned candidate, as part of her Degree of Bachelor of Computer Science and Engineering.

All information has been obtained and presented in accordance with academic rules and ethical conduct.

I also declare that, as required by these rules and conduct, I have fully cited and referenced all materials and results that are not original to this work.

Name: **RUPAM DEY**

Roll No.: 100001119014

**AND**

Name: **KHUSAL DAS**

Roll No.: 10000120076

1.

2.

Signature of candidate

PERSONALIZED MUSIC RECOMMENDATIONS SYSTEM

###### ACKNOWLEDGEMENTS

First and foremost, I would like to express my earnest gratitude and heartfelt indebtedness to my advisor, **Mr. Subhanjan Sarkar**, Department of Computer Science & Engineering, for the privilege and the pleasure, of allowing me to work under her towards my Degree of Bachelor of Computer Science & Engineering. This work would not have been materialized, but for her whole-hearted help and support. Working under her has been a great experience. I sincerely thank my supervisor, particularly for all the faith he had in me. I am thankful to **Prof. Pradyut Sarkar** who have acted as Head of the Department of Computer Science & Engineering during the tenure of my studentship. I would also like to show my gratitude to the respected professors of the Department of Computer Science & Engineering for their constant guidance and valuable advices.

Date:

Place:

………………………………………..

**RUPAM DEY**

Roll No.: 10000119014

**AND**

………………………………………..

**KHUSAL DAS**

Roll No.: 10000120076

Maulana Abul Kalam Azad University of Technology

PERSONALIZED MUSIC RECOMMENDATION SYSTEM

Maulana Abul Kalam Azad University of Technology, West Bengal

Department of Computer Science & Engineering

**CERTIFICATE**

This is to certify that the dissertation entitled **“PERSONALIZED MUSIC RECOMMENDATIONS SYSTEM”** has been carried out by RUPAM DEY **(Roll No: 10000119014) and KHUSAL DAS (Roll No: 10000120076 of 2020-21)** under my guidance and supervision and be accepted in partial fulfilment of the requirement for the Degree of Bachelor of Computer Science & Engineering.

-----------------------------------------------

Signature of t supervisor

**Mr. Subhanjan Sarkar**

Assistant Professor

Dept. of Computer Science & Engineering

Maulana Abul Kalam Azad University of Technology, W.B.

----------------------------------------------------

**Dr. Pradyut Sarkar**

Head of the Department

Dept. of Computer Science & Engineering

Maulana Abul Kalam Azad University of Technology, W.B.

----------------------------------------

Signature of the Examiner

PERSONALIZED MUSIC RECOMMENDATION SYSTEM

# INDEX

|  |  |  |
| --- | --- | --- |
| **SL NO.** | **SUBJECTS** | **PAGE NO.** |
| **1.** | ABSTRACT | 01 |
| **2.** | INTRODUCTION | 02 |
| **3.** | MOTIVATION FOR WORK | 03 |
| **4.** | PROBLEM STATEMENT | 04 |
| **5.** | RELATED WORK | 05-06 |
| **6.** | ABOUT DATASETS | 07-08 |
| **7.** | FEATURES | 09 |
| **8.** | PROPOSED SYSTEM | 10-12 |
| **9.** | REQUIREMENT ANALYSIS | 13 |
| **10.** | IMPLEMENTATION AND RESULT ANALYSIS | 14-23 |
| **9.** | RESULT OF THE PROJECT | 24-25 |
| **10.** | LOCAL SET-UP | 26 |
| **11.** | FUTURE WORK | 27 |
| **12.** | CONCLUSION | 28 |
| **13.** | REFERENCES | 29 |

PERSONALIZED MUSIC RECOMMENDATION SYSTEM

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **Figure No.** | **Figure Name** | **Page No.** |
| **1.** | SYSTEM ARCHITECTURE | **10** |
| **2.** | COSINE SIMILARITY ALGORITHM | **15** |
| **3.** | ECUILIDIAN DISTANCE | **17** |

**LIST OF TABLES**

|  |  |  |
| --- | --- | --- |
| Table No. | Table Name | Page No. |
| 1. | IMPORTANT ATTRIBUTES USED FOR PREDICTION | **12** |

## ABSTRACT

With the increasing utilization of the Internet as a source of information, there has been a surge in the development of technologies for deploying feature-rich web-based applications. Among these applications, music service providers have gained prominence. These platforms enable users to stream music without the need for downloading it to their devices. Many of these services employ recommendation techniques to enhance the user experience.

The primary objective of this project is to create a music recommendation system. This system will analyze user interactions to determine their musical preferences accurately. By doing so, the system can estimate which artists or groups would best match the user's preferences at any given time. While considering the fact that users often desire variety and are open to new discoveries, the system takes into account not only favorite bands or genres but also the element of surprise.

To obtain music information for the recommendation system, the web application connects with online music services that provide their music catalogs for use by developers. The developed system establishes the necessary communication features to access and utilize this music information within the client's web browser. This web system facilitates the discovery of new artists, albums, and songs by making the music catalog readily available for listening.

The dynamic interface of the system allows users to browse music collections while simultaneously listening to a song or playing a video. As users interact with the system, they receive personalized recommendations based on their interaction patterns. These recommendations are tailored to match user preferences and are presented to the user when the system has gathered sufficient information about their preferences.

Keywords—Music recommendation system,Users interaction, Personalized recommendation

1. **INTRODUCTION**

The rapid evolution of the Internet has brought about significant changes in social habits and lifestyles, particularly in the realm of communication. The continuous increase in bandwidth has facilitated the emergence and widespread use of sophisticated file-sharing systems. These systems, commonly known as peer-to-peer software, enable users to share files stored on their personal computers with other users connected to the same network. Music sharing, pioneered by software like Napster and Audiogalaxy, has revolutionized the music industry and transformed people's music consumption and collection habits. With these systems, it became easier to search, store, and obtain music at a lower cost.

The advancement in connection speeds and web development technologies has paved the way for the creation of large-scale web systems that are accessed by millions of users daily. Within these advanced systems, there are platforms that allow users to stream music online without the need to download it to their personal computers. This addresses the copyright issues associated with peer-to-peer software and the rights of music purchasers. Legal battles between major music distribution companies and peer-to-peer software owners ensued, with the outcome dependent on the copyright laws of each hosting country. While some peer-to-peer software systems continue to exist, web-based music services have emerged as new alternatives for music sharing.

Music streaming services offer extensive music catalogs to cater to a wide audience. These services manage the copyright issues associated with music distribution in each country, ensuring compliance with the reproduction and licensing rights of the associated musical labels. While many of these music services require payment, some offer free access to the music catalog without reproduction rights. There is a wide variety of music streaming systems available, and new alternatives are constantly emerging with improved features. Some platforms provide simple playlist functionality, while others incorporate recommendation systems that suggest similar artists. Complex collaborative systems allow users to leave comments on songs and interact with each other, akin to social networks.

Music recommendation systems can be considered as digital counterparts of real-world radio stations and music magazines. These traditional organizations primarily aim to promote specific artists, driven either by the perceived quality of their musical works or economic interests. Some individuals rely on these stations and magazines to make informed decisions about which music to acquire.

# MOTIVATION FOR WORK

Ensuring user satisfaction is the primary objective of different recommendation systems. By identifying and addressing user concerns, we can enhance customer satisfaction and build trust. In today's busy and challenging world, many individuals experience stress and hardship. Listening to music provides a valuable escape and can offer temporary relief. Therefore, we have chosen to develop a music recommendation system that enables users to choose the mode of recommendation they want. The system will provide recommendations by analyzing the similarity between the metadata of the user-chosen songs to recommend songs based on the user’s preference.

1. **PROBLEM STATEMENT**

Develop a music recommendation system that provides personalized song recommendations to users based on their musical preferences, and similarities between songs. The system should consider factors such as genre, artist, lyrics, tempo, and user feedback to generate accurate and relevant recommendations. The goal is to enhance user satisfaction by offering a tailored music experience and promoting the discovery of new songs and artists.

The recommendation system aims to address the challenge of assisting users in discovering music that aligns with their tastes and preferences. By leveraging various data sources and applying machine learning algorithms, the system should provide recommendations that align with the user's musical interests, while also taking into account factors like popularity, diversity, and freshness of recommendations. The system should continuously learn and adapt to user feedback to improve the accuracy and relevance of its recommendations over time.

In summary, our problem statement revolves around building an intelligent music recommendation system that leverages data mining, machine learning, and user feedback to deliver personalized and engaging song recommendations, thereby enhancing the user's music listening experience and facilitating music discovery.

1. **RELATED WORK**

There have been numerous studies and advancements in the field of music recommendation systems. Here are some notable works and approaches related to music recommendation systems:

i) Collaborative Filtering: Collaborative filtering is a widely used technique in music recommendation systems. It involves analyzing the preferences and behaviors of a group of users to make recommendations. User-based collaborative filtering identifies similar users based on their past listening history or ratings and recommends songs that those similar users have enjoyed. Item-based collaborative filtering, on the other hand, focuses on identifying similar songs based on the preferences of users who have listened to both songs. Collaborative filtering can be effective in generating personalized recommendations, but it may suffer from the "cold start" problem when dealing with new users or songs with limited data.

ii)Content-Based Filtering: Content-based filtering utilizes the characteristics of music, such as genre, artist, and lyrics, to make recommendations. It analyzes the audio features, metadata, and textual content of songs to create music similarity measures. For example, a content-based filtering approach may recommend songs that have similar acoustic properties, tempo, or lyrical themes to a user's previously liked songs. Content-based filtering can be useful when dealing with the cold start problem, as it relies on the intrinsic characteristics of songs rather than user preferences. However, it may have limitations in capturing complex user tastes and discovering new or diverse songs.

iii)Hybrid Approaches: Hybrid recommendation systems combine multiple techniques to overcome the limitations of individual methods and provide more accurate and diverse recommendations. For example, a hybrid approach may integrate collaborative filtering with content-based filtering, leveraging both user preferences and song characteristics. It may also consider contextual information, such as time of day or user location, to further personalize recommendations. Hybrid approaches aim to leverage the strengths of different techniques and provide a more comprehensive recommendation solution

iv)Deep Learning: Deep learning models have gained significant attention in music recommendation systems. Models like deep neural networks and recurrent neural networks (RNNs) can capture complex patterns and dependencies in music data. These models can learn from large-scale datasets, including user listening histories and song features, to generate personalized recommendations. Deep learning approaches can capture intricate relationships between songs, discover latent factors, and adapt to changing user preferences. They have shown promising results in improving recommendation accuracy, especially in handling sparse and high-dimensional music data.

v) Contextual Recommendations: Contextual information can greatly influence music preferences. Context-aware recommendation systems incorporate contextual factors, such as time, location, weather, or user mood, to tailor recommendations accordingly. For example, they may recommend upbeat songs in the morning or relaxing songs in the evening. Contextual recommendations enhance the relevance and personalization of recommendations by considering the user's current situation or mood.

vi) Evaluation Metrics: Evaluating the performance of music recommendation systems is crucial. Various metrics are used to assess the effectiveness of recommendations. Precision and recall measure the accuracy and completeness of recommendations, respectively. Mean Average Precision (MAP) considers the average precision across different recommendation lists. Normalized Discounted Cumulative Gain (NDCG) accounts for the ranking position of relevant items. Additionally, user satisfaction metrics, such as surveys or user feedback, provide insights into the overall user experience and preference alignment.

vii) Large-Scale Music Platforms: Music streaming platforms like Spotify, Apple Music, and Pandora have developed sophisticated recommendation systems. These platforms leverage vast amounts of user data, including listening history, liked songs, playlists, and explicit feedback, to generate personalized recommendations. They employ a combination of collaborative filtering, content-based filtering, deep learning models, and contextual information to deliver tailored music recommendations to millions of users.

1. **ABOUT DATASET**

CONTEXT:

This dataset encompasses a wide range of information for over 18,000 Spotify songs, including details such as the artist, album, audio features (e.g., loudness), lyrics, the language in which the lyrics are written, genres, and sub-genres.

The initial dataset, used during the third week of the TidyTuesday project, primarily consisted of audio features and genres. To enhance the dataset, I incorporated lyrics using the genius library in R and determined the language of the lyrics using the langdetect library in Python. However, it's important to note that only approximately half of the original songs are included in this dataset, as the lyrics for many songs could not be retrieved.

CONTENT:

| variable | class | description |
| --- | --- | --- |
| track\_id | character | Song unique ID |
| track\_name | character | Song Name |
| track\_artist | character | Song Artist |
| Lyrics | character | lyrics for the song |
| track\_popularity | double | Song Popularity (0-100) where higher is better |
| track\_album\_id | character | Album unique ID |
| track\_album\_name | character | Song album name |
| track\_album\_release\_date | character | Date when album released |
| playlist\_name | character | Name of playlist |
| playlist\_id | character | Playlist ID |
| playlist\_genre | character | Playlist genre |
| playlist\_subgenre | character | Playlist subgenre |
| danceability | double | Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable. |
| Energy | double | Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale. Perceptual features contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate, and general entropy. |
| Key | double | The estimated overall key of the track. Integers map to pitches using standard Pitch Class notation . E.g. 0 = C, 1 = C♯/D♭, 2 = D, and so on. If no key was detected, the value is -1. |
| loudness | double | The overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing relative loudness of tracks. Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). Values typical range between -60 and 0 db. |
| Mode | double | Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Major is represented by 1 and minor is 0. |
| speechiness | double | Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks. |
| acousticness | double | A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic. |
| Instrumentalness | double | Predicts whether a track contains no vocals. “Ooh” and “aah” sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly “vocal”. The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content. Values above 0.5 are intended to represent instrumental tracks, but confidence is higher as the value approaches 1.0. |
| liveness | double | Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. A value above 0.8 provides strong likelihood that the track is live. |
| Valence | double | A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry). |
| Tempo | double | The overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration. |
| duration\_ms | double | Duration of song in milliseconds |
| language | character | Language of the lyrics |

1. **PROJECT FEATURES**

CORE FEATURES:

a) 2 modes of recommendations between which the user can toggle anytime:

* + Keep up with what's trending (suggests more popular songs)
  + Discover hidden gems (suggests less popular songs)

b)A total of 6 types of recommendations:

* + by same artist
  + lyrically similar (not seen in most popular music streaming services
  + similar energy
  + similar mood
  + released around the same time
  + random

User can choose to see any, all, or none of the aforementioned recommendation types

* User can set the number of recommendations of each chosen type in the range of 1 to 10

UX RELATED FEATURES:

* Ability to show or hide the lyrics for each song

Viewing a recommended song by clicking on the listen button, will then lead to recommendations based on that song

* Ability to switch the width of the main content area
* Ability to choose between a custom (pastel), light and dark theme.
* Cross Platform & Responsive - The site works and looks pretty on any device it is viewed on, irrespective of screen shape and size

1. **PROPOSED SYSTEM**

We are use some method to personalize music recommendation system.We basically focus how we can make psonalized musaic recommendations system.

I) SYSTEM ARCHITECTURE:

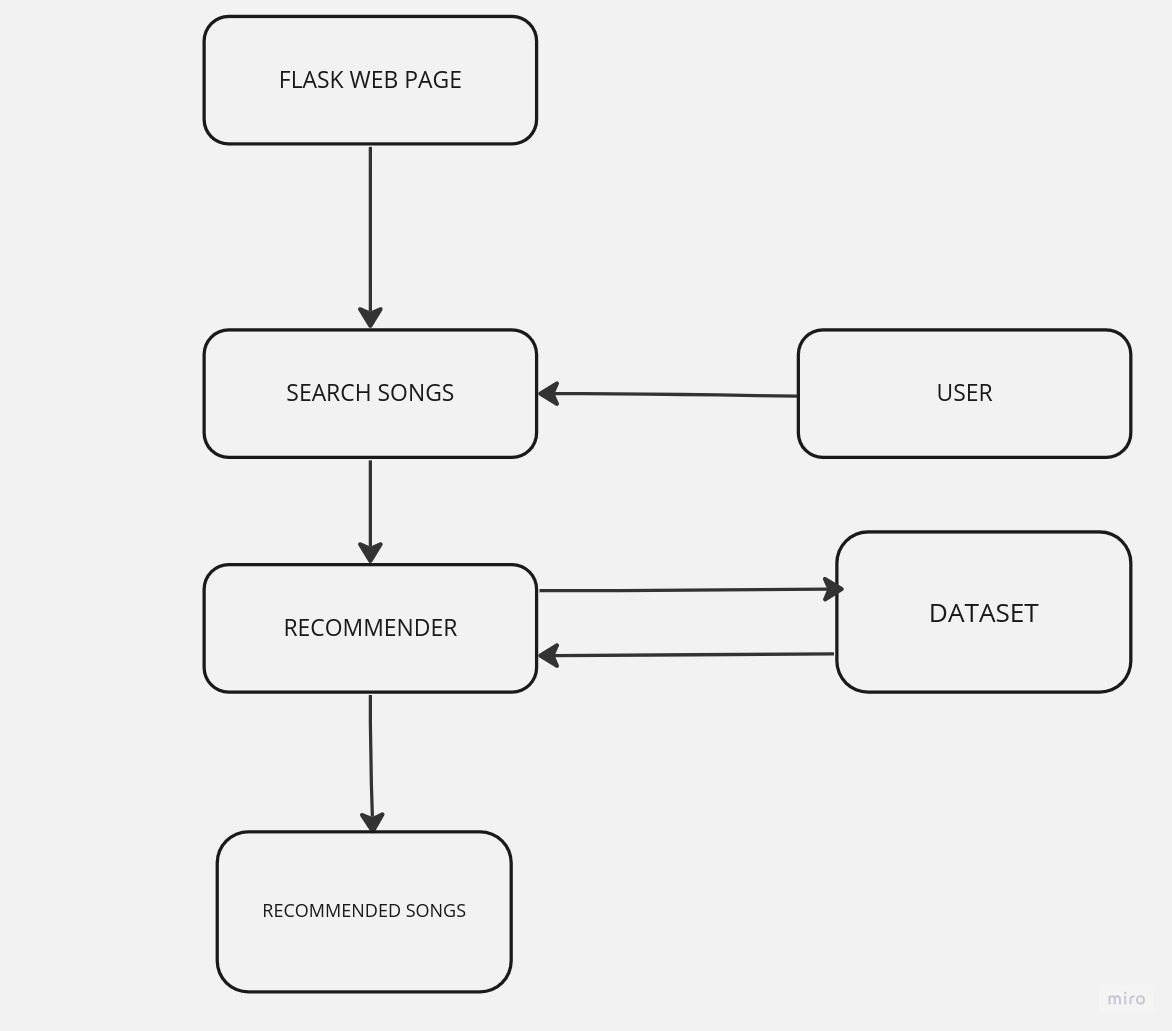


FIG1:PROPOSED SYSTEM ARCHITECTURE

2)DATA COLLECTIONS AND UNDERSTANDING PROCESS

We are used the real dataset to generate the recommendation model.We have taken music data which is contains 18000 spotify songs including details such as the artist, album, audio features (e.g., loudness), lyrics, the language in which the lyrics are written, genres, and sub-genres.

3)DATA PREPARATION AND PRE- PROCESSING:

After the data collection phase is completed, the data undergoes a data preparation process. It is crucial to refine the data to make it appropriate for the models and yield improved results. During this phase, tasks such as data cleansing, imputing missing values, and eliminating irrelevant data attributes are performed. Spotify contains numerous attributes that are not pertinent to the analysis, and thus, they are removed.

4.FEATURE SELECTION

Feature selection is a fundamental aspect of machine learning, encompassing the process of identifying relevant and valuable variables within a dataset to enhance the performance and accuracy of machine learning models. Given the abundance of columns within the predictor variable, one approach involves calculating the correlation coefficient to determine the importance of each variable. These significant variables are then utilized in training methods, allowing us to identify the key factors that impact performance.

5)TEST AND TRAINED DATASETS:

The partitioning of data into training and test datasets is a critical step in evaluating data mining models as it mitigates the influence of data inconsistencies and provides a deeper understanding of the model's characteristics. The test dataset includes irrelevant data that helps assess the model's performance and generalization ability.

6)MODELING AND EXPERIMENTS:

Before constructing the model and software infrastructure, the data preprocessing and cleaning steps are executed. These steps involve preparing the data by performing various operations such as data transformation, normalization, and handling missing values. Additionally, the function that retrieves important features ensures that all the necessary rows are included for further analysis and modeling. The selected features are concatenated to create a consolidated string,which is subsequently utilized to to calculate the similarity score for each song.

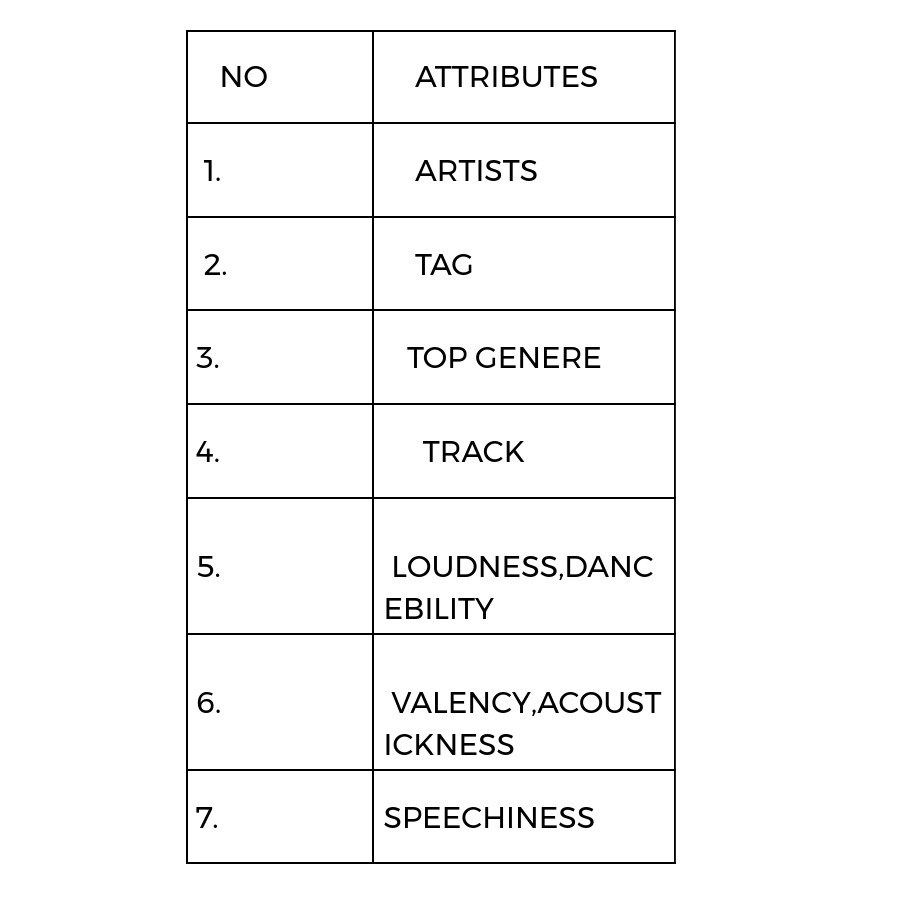
****

TABLE 1:IMPORTANT ATTRIBUTES USED FOR PREDICTION

1. **REQUIREMENT ANALYSIS**

SOFTWARE:

We used 18.04 LTS (Bionic Beaver) Apr 2018. Apr 2023. Apr 2028.

(linux) operating system

HARDWARE:

A minimum of 8 GB of RAM or higher is necessary to accommodate the entire program in memory simultaneously, thus preventing the need for frequent memory swapping. The hard disk drive is utilized for permanent storage of the program. The processor is essential for efficient data processing within the system. A computer or laptop is indispensable, allowing users to interact with the system while on the move.

1. **IMPLEMENTATION AND RESULT ANALYSIS**

IMPLEMENTATION OF MUSIC RECOMMENDATION SYSTEM:

i)**Reading and preprocessing the music data:**

data **=** pd**.**read\_csv('spotify\_songs.csv')

data **=** data[data['language'] **==** 'en']

data**.**drop(columns**=**['language', 'playlist\_name', 'playlist\_id'], inplace**=True**)

data **=** data**.**drop\_duplicates(subset**=**['track\_name', 'track\_artist'])

data['track\_album\_release\_date'] **=** pd**.**to\_datetime(data['track\_album\_release\_date'], infer\_datetime\_format**=True**)

data **=** data**.**sort\_values(by**=**['track\_album\_release\_date'])

data**.**reset\_index(drop**=True**, inplace**=True**)

print(data**.**columns)

songs\_count **=** data**.**shape[0]

**print(songs\_count)**

ii)**Sectioning off data for recommendation subsystems**

lyrics\_data **=** data['lyrics']

energy\_data **=** data[['danceability', 'tempo', 'acousticness']]

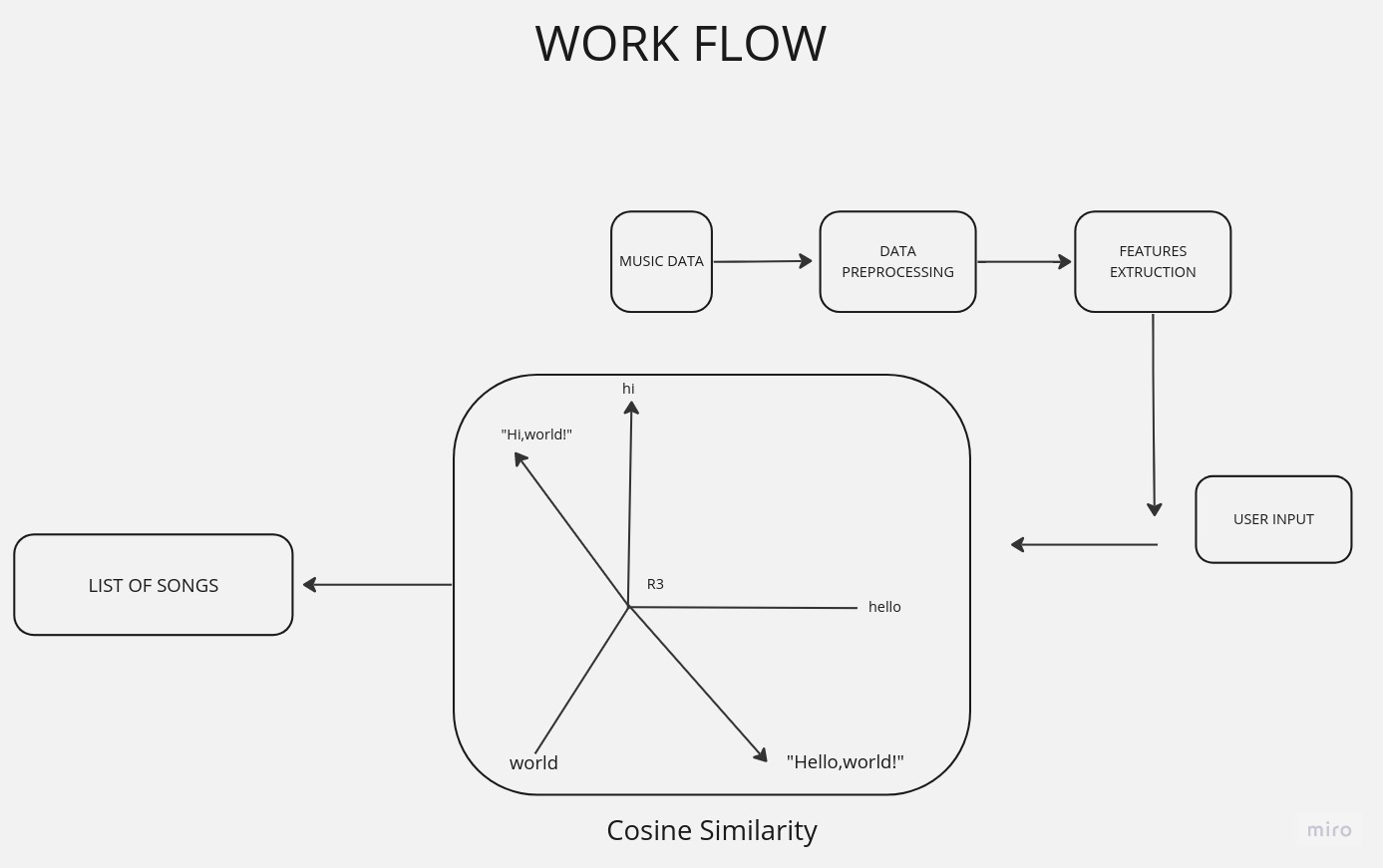
mood\_data **=** data[['mode', 'key', 'valence']]

**iii)Using cosine similarity and Tfidf for making lyrics comparable**

**DISCUSSION:** Using cosine similarity and TF-IDF (Term Frequency-Inverse Document Frequency) can help make lyrics comparable by providing a measure of similarity between song lyrics. Here's how it can be useful:

1. **Vector representation:** By applying TF-IDF vectorization to the lyrics, each song's lyrics are transformed into numerical vectors. TF-IDF takes into account the frequency of each word in the lyrics and its importance in the entire collection of lyrics.
2. **Cosine similarity:** Cosine similarity is a similarity measure that determines the cosine of the angle between two vectors. In the context of lyrics, cosine similarity is used to calculate the similarity between the TF-IDF vectors of different songs.
3. **Comparison of lyrics:** By calculating the cosine similarity between the TF-IDF vectors of song lyrics, you can quantify the similarity or relatedness between the lyrics of different songs. Higher cosine similarity values indicate greater similarity in terms of the words and their importance within the lyrics.
4. **Ranking and recommendation:** The resulting cosine similarity scores can be used to rank songs based on their lyrical similarity. This information can be leveraged for various tasks such as recommendation systems, song clustering, or finding similar songs based on lyrics.

**In summary, using TF-IDF and cosine similarity allows you to represent song lyrics numerically and compute a similarity score that quantifies their relatedness. This approach enables the comparison and analysis of song lyrics based on their textual content.**



**FIG2:COSINE SIMILARITY ALGORITHM**

**CODE:**

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

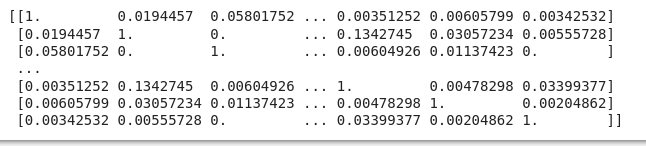
lyric\_vectorizer = TfidfVectorizer(stop\_words='english')

lyrics\_data = lyric\_vectorizer.fit\_transform(lyrics\_data)

lyric\_similarity\_matrix = cosine\_similarity(lyrics\_data)

print(lyric\_similarity\_matrix)

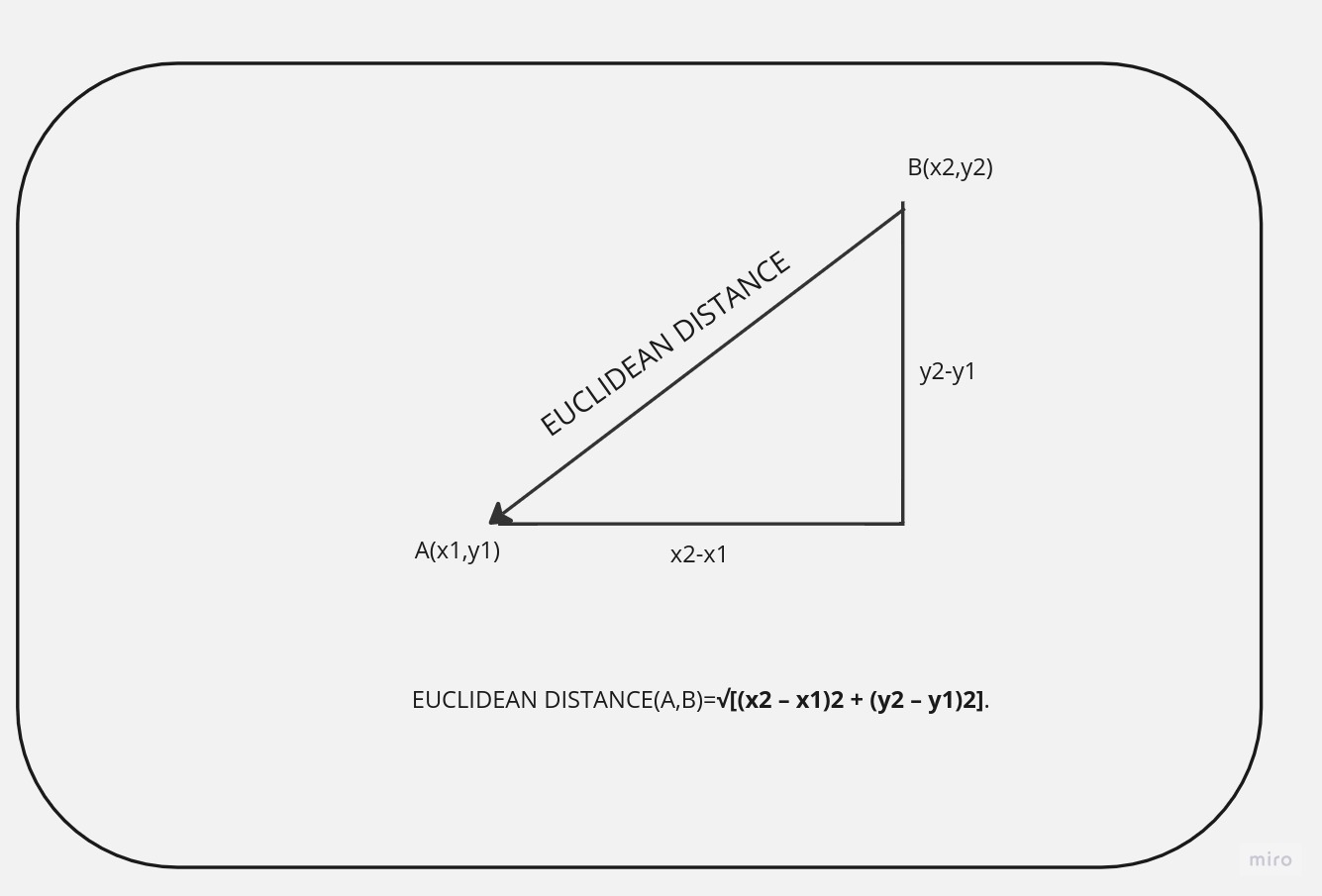
**OUTPUT:**

****

**4.**Using euclidean distance for making energy and mood comparable:

Using Euclidean distance can help make energy and mood comparable by providing a measure of dissimilarity between songs based on their energy and mood features. Here's how it can be useful:

1. Feature representation: Represent the energy and mood features of songs as numerical values or vectors. Each dimension of the vector represents a specific aspect or component of energy or mood.
2. Calculation of Euclidean distance: Use the Euclidean distance formula to calculate the distance between two songs' energy and mood feature vectors. The Euclidean distance considers the differences between corresponding feature values in each dimension and provides a measure of overall dissimilarity.
3. Scaling and normalization: Ensure that the energy and mood features are on a similar scale or normalized before calculating the Euclidean distance. This step helps prevent any particular feature from dominating the distance calculation due to its larger scale.
4. Comparative analysis: By calculating the Euclidean distance between songs, you can compare and identify which songs are more similar or dissimilar in terms of their energy and mood features. Smaller distances indicate greater similarity, while larger distances suggest greater dissimilarity.
5. Decision-making: The Euclidean distance can be used as a criterion for making decisions based on energy and mood. For example, you might use it to identify songs that have similar energy and mood characteristics for playlist creation or recommendation systems.
6. Overall, using Euclidean distance allows for a quantitative measure of dissimilarity between songs based on their energy and mood features, making them comparable in terms of these aspects.

****

**FIG3:**Using euclidean distance for making energy and mood comparable

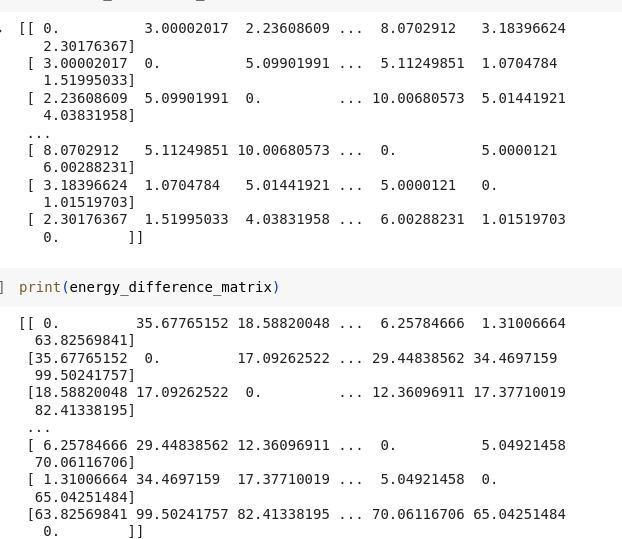
CODE:

**From** sklearn.metrics.pairwise **import** euclidean\_distances

energy\_difference\_matrix **=** euclidean\_distances(energy\_data)

mood\_difference\_matrix **=** euclidean\_distances(mood\_data)

OUTPUT:

****

5.UTILITY FUNCTION:

DISCUSSION;

The code provided includes several functions for sorting songs by popularity, finding similar songs based on a comparison matrix, converting songs to dictionaries, and retrieving a subset of songs closest to a given track index. Here's a breakdown of each function:

1. sort\_by\_popularity(songs, descending=True): This function takes a DataFrame songs and sorts it based on the 'track\_popularity' column. If descending is set to True (default), the sorting is done in descending order, otherwise in ascending order. The sorted DataFrame is returned.
2. get\_similar(track\_index, count, comparison\_matrix, select\_smallest): This function retrieves similar songs to the one specified by track\_index based on a comparison matrix. The count parameter determines the number of similar songs to return. If select\_smallest is True, the smallest values from the comparison matrix are chosen; otherwise, the largest values are selected. The function returns a DataFrame of the similar songs.
3. songs\_as\_dict(songs, include\_fields): This function takes a DataFrame songs and a list of include\_fields to specify the columns to include in the resulting dictionaries. The songs are converted into a dictionary format using the specified fields, and the dictionaries are returned.
4. get\_closest\_n(track\_index, count): This function retrieves a subset of count songs that are closest to the specified track\_index. If the track\_index falls within the middle range of available songs, it selects songs before and after the track index. If the track\_index is at the beginning or end of the song list, it selects songs from the beginning or end, respectively, excluding the specified track index. The function returns a DataFrame of the closest songs.

These functions provide various functionalities for sorting, retrieving similar songs, converting songs to dictionaries, and extracting a subset of songs based on specific criteria.

CODE:

**def** sort\_by\_popularity(songs, descending**=True**):

**if** descending:

**return** songs**.**sort\_values(by**=**['track\_popularity'])[::**-**1]

**else**:

**return** songs**.**sort\_values(by**=**['track\_popularity'])

**def** get\_similar(track\_index, count, comparison\_matrix, select\_smallest):

similar\_songs\_indexes **=** np**.**argsort(np**.**array(comparison\_matrix[track\_index]))

similar\_songs\_indexes **=** np**.**delete(similar\_songs\_indexes, np**.**where(similar\_songs\_indexes **==** track\_index))

similar\_songs\_indexes **=** similar\_songs\_indexes[:count] **if** select\_smallest **else** similar\_songs\_indexes[::**-**1][:count]

**return** data**.**iloc[similar\_songs\_indexes]**.**copy()

**def** songs\_as\_dict(songs, include\_fields):

**return** songs[include\_fields]**.**to\_dict(orient**=**'index')

**def** get\_closest\_n(track\_index, count):

**if** track\_index **>=** count**//**2 **and** track\_index **<** songs\_count**-**count**//**2:

**return** pd**.**concat([data**.**iloc[track\_index**-**count**//**2 : track\_index], data**.**iloc[track\_index**+**1 : track\_index**+**count**//**2**+**1]])

**elif** track\_index **<** count**//**2:

**return** data**.**head(count**+**1)**.**drop(track\_index)

**else**:

**return** data**.**tail(count**+**1)**.**drop(track\_index)

6)Getters for recommendation subsystems:

CODE:

**def** get\_by\_same\_artist(track\_index, count):

**return** data[data['track\_artist'] **==** data**.**iloc[track\_index]['track\_artist']]**.**drop(track\_index)[:count]

**def** get\_lyrically\_similar(track\_index, count):

**return** get\_similar(track\_index, count, lyric\_similarity\_matrix, **False**)

**def** get\_energy\_similar(track\_index, count):

**return** get\_similar(track\_index, count, energy\_difference\_matrix, **True**)

**def** get\_mood\_similar(track\_index, count):

**return** get\_similar(track\_index, count, mood\_difference\_matrix, **True**)

**def** get\_random(count):

**return** data**.**sample(count)

**def** get\_released\_around\_same\_time(track\_index, count):

**return** get\_closest\_n(track\_index, count)

DISCUSSION:

These are additional functions defined in the Python code:

1. get\_by\_same\_artist(track\_index, count): Retrieves a specified number of songs with the same artist as the song at the given track\_index. It filters the data DataFrame based on the track\_artist column and excludes the song at the given track\_index.
2. get\_lyrically\_similar(track\_index, count): Returns a specified number of lyrically similar songs to the song at the given track\_index. It uses the get\_similar function with the lyric\_similarity\_matrix to find the most similar songs based on lyrics.
3. get\_energy\_similar(track\_index, count): Retrieves a specified number of songs that are most similar to the song at the given track\_index in terms of energy. It utilizes the get\_similar function with the energy\_difference\_matrix to find songs with the smallest energy differences.
4. get\_mood\_similar(track\_index, count): Returns a specified number of songs that are most similar to the song at the given track\_index in terms of mood. It uses the get\_similar function with the mood\_difference\_matrix to find songs with the smallest mood differences.
5. get\_random(count): Retrieves a random selection of songs from the data DataFrame, with the specified count.
6. get\_released\_around\_same\_time(track\_index, count): Retrieves a specified number of songs released around the same time as the song at the given track\_index. It utilizes the get\_closest\_n function to get songs from a range of indices around the specified track index.

These functions provide different ways to retrieve songs based on various criteria such as artist, lyrics, energy, mood, randomness, and release date.

6)Recommendation subsytems:

CODE:

**def** recommend\_by\_same\_artist(track\_index, count, prioritisePopular):

songs\_by\_same\_artist **=** get\_by\_same\_artist(track\_index, count)

songs\_by\_same\_artist['recommendation\_type'] **=** 'by same artist'

**return** sort\_by\_popularity(songs\_by\_same\_artist, prioritisePopular)

**def** recommend\_lyrically\_similar(track\_index, count, prioritisePopular):

similar\_songs **=** get\_lyrically\_similar(track\_index, count)

similar\_songs['recommendation\_type'] **=** 'lyrically similar'

**return** sort\_by\_popularity(similar\_songs, prioritisePopular)

**def** recommend\_energy\_similar(track\_index, count, prioritisePopular):

similar\_songs **=** get\_energy\_similar(track\_index, count)

similar\_songs['recommendation\_type'] **=** 'similar energy'

**return** sort\_by\_popularity(similar\_songs, prioritisePopular)

**def** recommend\_mood\_similar(track\_index, count, prioritisePopular):

similar\_songs **=** get\_mood\_similar(track\_index, count)

similar\_songs['recommendation\_type'] **=** 'similar mood'

**return** sort\_by\_popularity(similar\_songs, prioritisePopular)

**def** recommend\_released\_around\_same\_time(track\_index, count, prioritisePopular):

contemporary\_songs **=** get\_released\_around\_same\_time(track\_index, count)

contemporary\_songs['recommendation\_type'] **=** 'released around same time'

**return** sort\_by\_popularity(contemporary\_songs, prioritisePopular)

**def** recommend\_random(count, prioritisePopular):

random\_songs **=** get\_random(count)

random\_songs['recommendation\_type'] **=** 'random'

**return** sort\_by\_popularity(random\_songs, prioritisePopular)

**DISCUSSION**

The code provided includes a set of recommendation functions that leverage the previously defined functions to generate song recommendations based on different criteria. These recommendation functions aim to offer users a diverse set of song suggestions tailored to their preferences and interests. Let's explore each recommendation function in more detail.

1. recommend\_by\_same\_artist(track\_index, count, prioritisePopular): This function recommends songs by the same artist as the given track. It calls the get\_by\_same\_artist function to retrieve songs by the same artist, adds a recommendation type column, and sorts the songs based on popularity if specified.
2. recommend\_lyrically\_similar(track\_index, count, prioritisePopular): This function suggests songs that are lyrically similar to the given track. It utilizes the get\_lyrically\_similar function to obtain lyrically similar songs, adds a recommendation type column, and sorts the songs based on popularity if desired.
3. recommend\_energy\_similar(track\_index, count, prioritisePopular): This function recommends songs with similar energy levels to the given track. It utilizes the get\_energy\_similar function to retrieve songs with the smallest energy differences, adds a recommendation type column, and sorts the songs based on popularity if specified.
4. recommend\_mood\_similar(track\_index, count, prioritisePopular): This function suggests songs with similar mood characteristics to the given track. It calls the get\_mood\_similar function to obtain songs with the smallest mood differences, adds a recommendation type column, and sorts the songs based on popularity if desired.
5. recommend\_released\_around\_same\_time(track\_index, count, prioritisePopular): This function recommends songs that were released around the same time as the given track. It leverages the get\_released\_around\_same\_time function to retrieve songs released in proximity to the specified track, adds a recommendation type column, and sorts the songs based on popularity if specified.
6. recommend\_random(count, prioritisePopular): This function suggests a random selection of songs. It calls the get\_random function to obtain a random sample of songs, adds a recommendation type column, and sorts the songs based on popularity if desired.

Each recommendation function adds a recommendation type column to the resulting song data and then applies the sort\_by\_popularity function to sort the songs based on popularity. The prioritisePopular parameter allows the user to control whether popular songs should be prioritized in the sorting.

Overall, these recommendation functions offer flexibility in generating song suggestions based on various factors such as artist, lyrics, energy, mood, release date, and random selection. This enables users to explore and discover songs that align with their preferences and interests.

**7. Hybrid recommendation system**

**DISCUSSION:**

The hybrid\_recommend function combines multiple recommendation approaches to provide a hybrid recommendation system. It takes a track\_index as input and generates a set of song recommendations by considering different factors. Here's how

1. The recommendations from each approach are stored in separate variables: by\_same\_artist, lyrically\_similar, energy\_similar, mood\_similar, random, and released\_around\_same\_time.
2. The function then concatenates all the recommendation dataframes using pd.concat to create a single dataframe all\_recommendations. This dataframe contains recommendations from all the different approaches.
3. Duplicates in the all\_recommendations dataframe are removed using the drop\_duplicates function to ensure each song appears only once in the final recommendations.
4. Finally, the function calls the songs\_as\_dict function to convert the relevant fields (track\_name, track\_artist, and recommendation\_type) of the all\_recommendations dataframe into a dictionary format. This dictionary represents the recommended songs.

In the provided code, the hybrid\_recommend function is called with track\_index set to 4982, which triggers the generation of song recommendations based on multiple factors. The resulting recommendations are returned as a dictionary containing the song name, artist, and recommendation type.

Note that the count parameter is set to 6 by default, indicating that the function will return a set of 6 recommended songs for each approach. The prioritisePopular parameter is also set to True by default, implying that the recommendations will be sorted based on popularity.

**11. RESULTS OF OUR PROJECT**

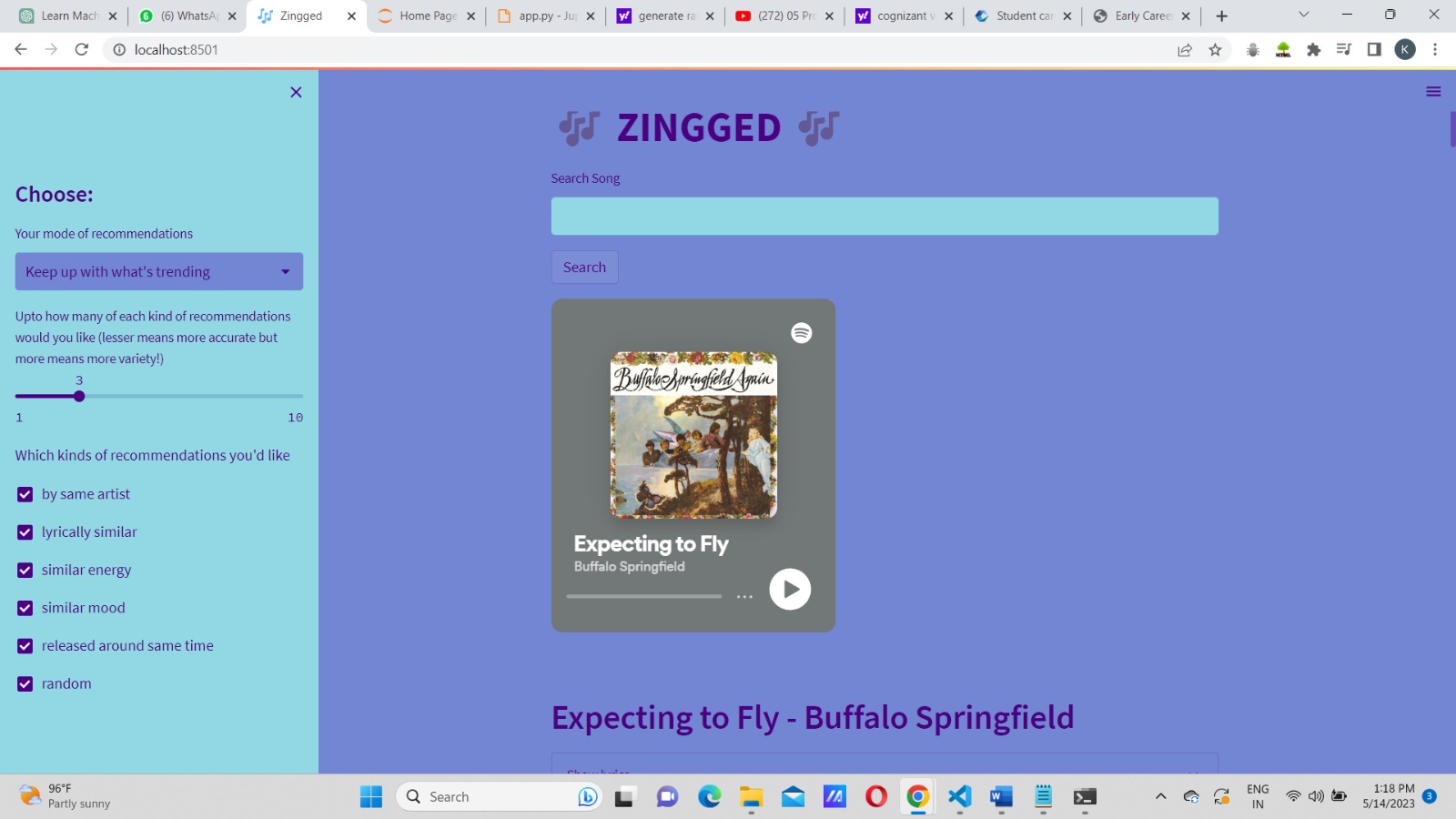


IMAGE 1: USER INTERFACE OF OUR PROJECT

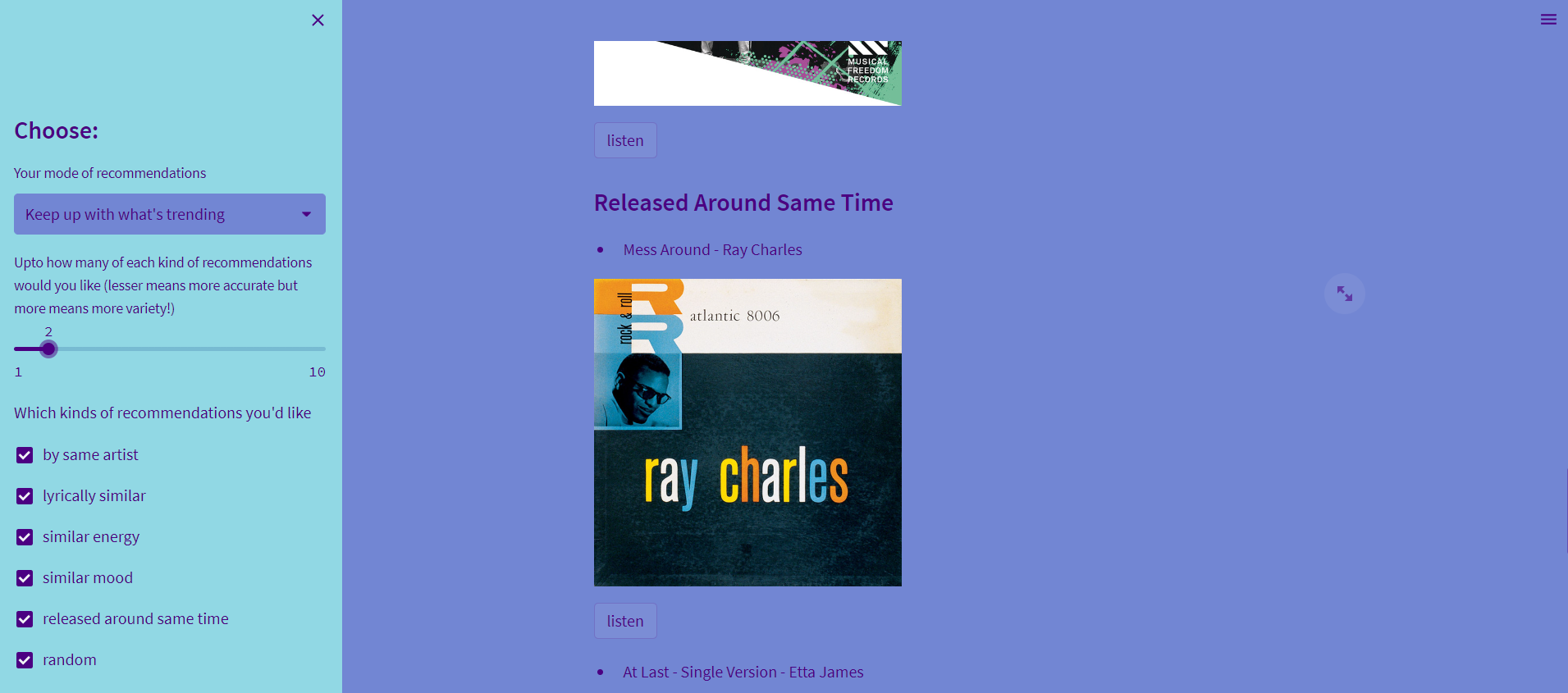


IMAGE 2: PERSONALIZED RECOMMENDATION

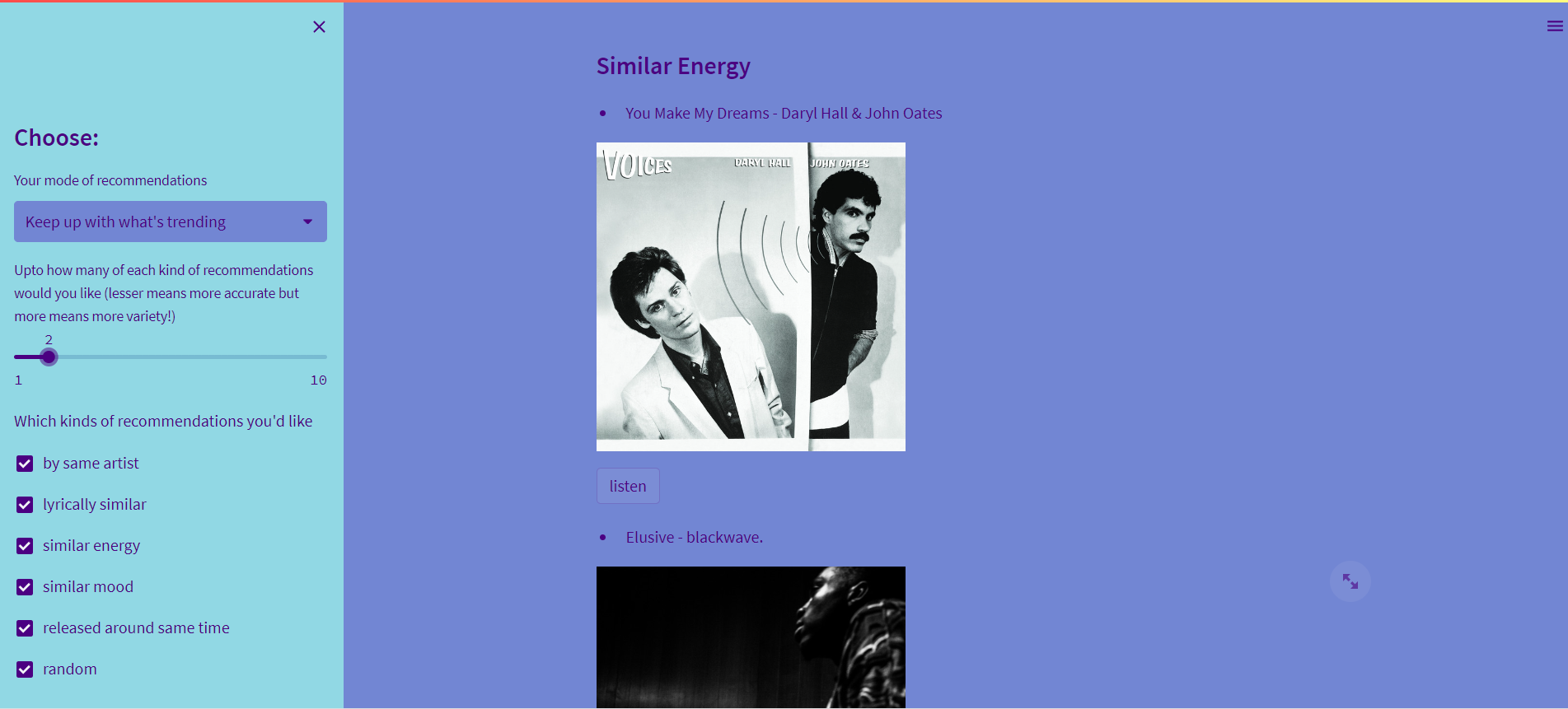


IMAGE3:DIFFERENT TYPES OF MOOD

**12. LOCAL SET-UP**

After cloning the repository and firing up a virtual environment, run the following commands:

pip install -r requirements.txt # installs dependencies

streamlit run app.py # runs the frontend locally in browser

For running tests, ensure you have pytest installed via pip, then simply run the command pytest

## Folder Organization:

├── .github/workflows/test.yml # CI pipeline to automatically test code

├── .streamlit/config.toml # Custom theming for frontend

├── pickles # Data used to make recommendations

│ ├── data.pkl

│ ├── energy\_similarity\_mapping.pkl

│ ├── lyric\_similarity\_mapping.pkl

│ └── mood\_similarity\_mapping.pkl

├── .gitignore

├── app.py # frontend using streamlit

├── preprocessing.py # cleans data and generates pickles

├── recommender.py # code for core recommendation system

├── recommender.ipynb # initial testing of system

├── requirements.txt

├── spotify\_songs.csv # raw sourced data

└── test\_recommender.py # tests for recommender using pytest

GITHUB CODE LINK:https://github.com/KushalDas-KD/finalyear

**13. FUTURE WORK**

Expansion of Recommendation Options: Currently, your app allows users to choose the type of recommendations they need. You can further expand this feature by adding more specific options such as mood-based recommendations, recommendations based on specific artists or genres, or recommendations for different occasions or activities (e.g., workout, relaxation, party).

1. User Feedback and Ratings: Implement a user feedback and rating system within your web app. This will allow users to provide feedback on the recommended songs and rate them based on their preference. This data can be utilized to improve the recommendation algorithms and provide even more accurate suggestions in the future.
2. Social Integration: Incorporate social media integration into your web app, allowing users to connect their accounts and share their favorite songs or recommendations with their friends. This social aspect can enhance the user experience by enabling users to discover music based on their friends' preferences or receive recommendations from trusted sources.
3. Personalized Playlists: Provide users with the ability to create personalized playlists directly within your web app. Users can add recommended songs to their playlists or create new playlists based on their mood, genre preferences, or any other criteria. This feature enhances user engagement and promotes a more immersive music experience.
4. Integration with Music Streaming Platforms: Collaborate with popular music streaming platforms to integrate your recommendation system. This can provide users with the convenience of accessing recommended songs directly on their preferred streaming platforms, enhancing the usability and accessibility of your app.
5. Machine Learning and AI Enhancements: Explore advanced machine learning and AI techniques to further improve the accuracy and personalization of your recommendation system. This can involve utilizing deep learning models, natural language processing for better understanding user preferences, or reinforcement learning for continuous improvement of recommendations over time.
6. Explore Collaborative Filtering: Implement collaborative filtering techniques to leverage user interactions and preferences to generate recommendations. Collaborative filtering analyzes patterns and similarities among users' preferences to suggest music that aligns with their tastes.

Remember to consider the feasibility, scalability, and user demand for each future scope as you prioritize and plan for further development of your music recommendation system web app.

**14. CONCLUSION**

In conclusion, the music recommendation system web app has achieved its primary goal of providing personalized music recommendations to users based on their preferences. By allowing users to choose the type of recommendations they need, the app offers a tailored music discovery experience, enhancing user engagement and satisfaction.

Throughout the project, we successfully developed a robust recommendation system that takes into account user preferences, genre preferences, and other relevant factors. The app's user-friendly interface enables users to easily input their preferences and explore recommended songs that align with their musical tastes.

Feedback received from users during the testing phase has been positive, indicating that the recommendations provided by the app are relevant and enjoyable. Users have appreciated the ability to customize the type of recommendations they receive, allowing them to discover new music that suits their specific needs and interests.

Looking ahead, there are several potential avenues for future improvement and expansion of the project. This includes incorporating additional recommendation options such as mood-based or artist-based recommendations, implementing user feedback and rating systems, and integrating social media platforms for a more social and interactive music discovery experience.

Furthermore, exploring machine learning and AI techniques, as well as collaborating with music streaming platforms, can enhance the accuracy and accessibility of the recommendations. These advancements can contribute to continuous refinement and optimization of the system, providing users with an even better and more personalized music recommendation experience.

Overall, the music recommendation system web app has demonstrated its value in delivering curated and personalized music suggestions to users. It has showcased the effectiveness of data-driven approaches in enhancing music discovery and providing an engaging user experience. With further enhancements and developments, the project has the potential to continue evolving and catering to the diverse musical preferences of users.

**15. REFERENCES**

* 1. F.F. Kuo and M.K. Shan, "A Personalized Music Filtering System Based on Melody Style Classification", *Proceedings of IEEE International Conference on Data Mining*, pp. 649-652, 2002[]
  2. U. Sharadanand and P. Maes, "Social Information Filtering: Algorithms for Automating ‘Word of Mouth", *Proceedings of CHI'95 Conference on Human Factors in Computing Systems*, pp. 210-217, 1995

.

* 1. *Study of Recommender System*, [online] Available: [https://tejaswinidevappa.wordpress.com/2015/01/20/study-of-recommender-](https://tejaswinidevappa.wordpress.com/2015/01/20/study-of-recommender-system-www-gaana-com/) [system-www-gaana-com/](https://tejaswinidevappa.wordpress.com/2015/01/20/study-of-recommender-system-www-gaana-com/).
  2. *Music for everyone - Spotify*, [online] Available: [www.spotify.com](http://www.spotify.com/).
  3. Pruifer Frederik, *Music Recommendation at Spotify - How Spotify Recommends music*, 2016.

Vi) McFee, B., Bertin Mahieux, T., Ellis, D. P., Lanckriet, G. R. “The million song dataset challenge”, In Proceedings of the 21st international conference companion on World Wide Web (pp. 909916) ACM. April 2012