Welcome to DA331 Mid term Presentation

Spotify inspired Song Recommendation System

Song Recommendation System using the Spotify Million Playlist Dataset. It employs content-based filtering, analyzing song features and metadata to provide personalized music recommendations.

By Team 5
Ankita Anand (210150006) and
Devahuti Talukder (200121015)

Breaking Down The Making Of Project

Dataset Preparation by extracting features using Spotify web API

Using MongoDB Querying and Modeling

Developing the website to deploy model.

By Ankita Anand

Feature Extraction

Reading1M_dataset_extracting_URL.ipynb

1

Here we read JSON data, extract playlists, transform the data using PySpark, and define a UDF to process URIs. It also counts the total number of playlists and playlist entries in the data.

Reading the 1 million playlists and keeping only the unique track URIs for the content-based recommendation system.
After extraction csv contain these three columns ['track_uri', 'artist_uri', 'album_uri']

A Spotify URI (Uniform Resource Identifier) is a unique identifier used by the Spotify music streaming service to refer specific content, such as songs, albums, playlists, artists, and more. URIs are used to provide a direct link to a particular piece of content within the Spotify platform.

By Ankita Anand

Feature Extraction

extracting_track_audio_album_features_csv.ipynb

Using spotify API we can extract a lot of information about the audio like (Audio Features, Track Release Date, Track Popularity, Artist Popularity, Artist Genres)

Artist, Audio and Track features Extraction.

1. Acousticness : measuring if a track is acoustic
2. Danceability : A value of 0.0 is least danceable and 1.0 is most danceable.
3. Speechiness : detecting spoken words
4. Valence : from 0.0 to 1.0 describing the musical positiveness conveyed by a track.
5. Mode, etc.

These features are very powerful in understanding the sentiments of songs.

By Ankita Anand

Pre-Processed Dataset

Columns Present in the final processed Dataset

['track_uri', 'artist_uri', 'album_uri', 'danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness','liveness', 'valence', 'tempo', 'duration_ms', 'time_signature', 'Track_release_date', 'Track_pop', 'Artist_pop', 'Artist_genres']

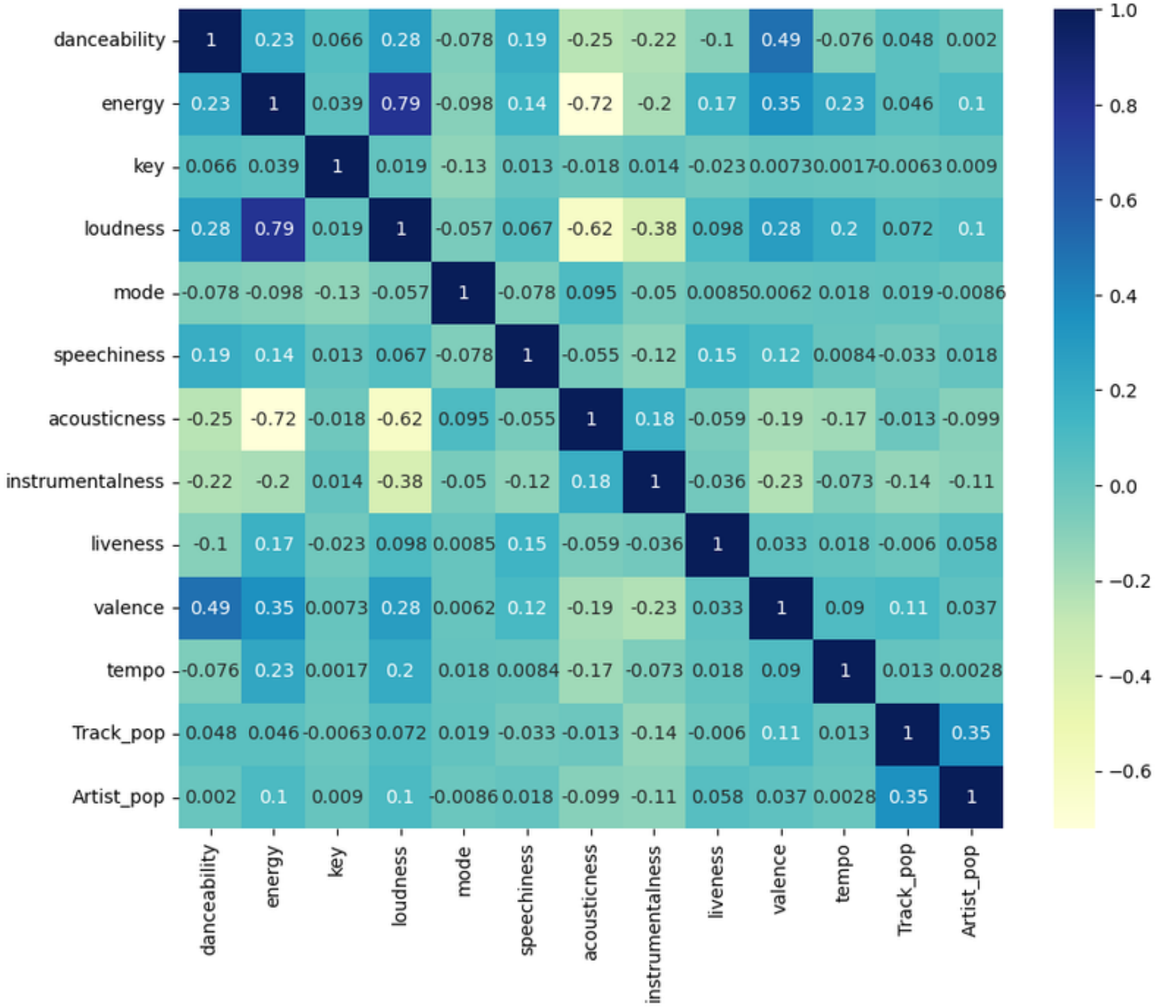
Dataframe Summary

```
In [23]: df_new.describe()
```

Out[23]:

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	vale
count	199492.000000	199492.000000	199492.000000	199492.000000	199492.000000	199492.000000	199492.000000	199492.000000	199492.000000	199492.000
mean	0.567603	0.618908	5.249799	-8.313687	0.659395	0.089608	0.290860	0.130381	0.203058	0.469
std	0.173274	0.241795	3.576157	4.612043	0.473914	0.105199	0.319041	0.281327	0.179003	0.254
min	0.000000	0.000000	0.000000	-60.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
25%	0.455000	0.455000	2.000000	-10.082000	0.000000	0.034800	0.019700	0.000000	0.096300	0.261
50%	0.579000	0.656000	5.000000	-7.177000	1.000000	0.047300	0.145000	0.000042	0.128000	0.456
75%	0.695000	0.817000	8.000000	-5.268000	1.000000	0.090300	0.523000	0.025200	0.256000	0.671
max	0.990000	1.000000	11.000000	3.744000	1.000000	0.965000	0.996000	1.000000	1.000000	0.999

Correlation Heatmap



By Ankita Anand

Explanation on Modeling:

- The code starts by applying **TF-IDF** (Term Frequency-Inverse Document Frequency) vectorization on the 'Artist_genres' column, which represents the genres associated with artists. It then processes audio features and other information related to tracks and artists from the Spotify API.

- **Cosine similarity** is calculated between the entire dataset and a user's playlist , which is used to measure the similarity between the user's playlist and other tracks.The DataFrame is sorted by cosine similarity in descending order.Recommendations are generated by selecting tracks that are most similar to the user's playlist, and the top 50 tracks are chosen.

- A DataFrame called 'Fresult' is created to store information about the recommended tracks. This includes the track name and artist name.

Model Deployment

- Users will input their playlist, which should be a list of tracks or songs.
- Data Integration:
  - Check if each track from the user's playlist exists in your dataset. If any are missing, add them automatically. This ensures that you have vectors for all songs in the playlist.
- Playlist Vectorization:
  - Convert the user's playlist into a single vector using the same feature extraction methods and preprocessing used for the rest of the dataset.
- Cosine Similarity:
  - Calculate the cosine similarity between the user's playlist vector and the vectors of all songs in your dataset. This will help identify songs that are most similar to the user's taste.
- Recommendation: Rank the songs based on their cosine similarity scores, and recommend the top N songs to the user.

Output

In [210]:
```
=cosine_similarity(df.drop(['track_uri', 'artist_uri', 'album_uri'], axis = 1),playvec.drop(['track_uri', 'artist_ur
]=cosine_similarity(df.iloc[:,16:-1],playvec.iloc[:,16:])
]=cosine_similarity(df.iloc[:,19:-2],playvec.iloc[:,19:])
ort_values(['sim3','sim2','sim'],ascending = False,kind='stable')
upby('artist_uri').head(5).track_uri.head(50)     #to limit recmmendation by same artist
cks(qq[0:50])
d.DataFrame()
range(50):
t=pd.DataFrame([i])
t['track_name']=aa['tracks'][i]['name']
t['artist_name']=aa['tracks'][i]['artists'][0]['name']
lt=pd.concat([Fresult,result],axis=0)
```

Out[210]:

	0	track_name	artist_name
0	0	Knights of Cydonia	Muse
0	1	Somebody Else	The 1975
0	2	Famous Last Words	My Chemical Romance
0	3	Madness	Muse
0	4	Robbers	The 1975
0	5	Read My Mind	The Killers
0	6	Time is Running Out	Muse
0	7	Trouble	Cage The Elephant
0	8	Call It Fate, Call It Karma	The Strokes
0	9	Cold Cold Cold	Cage The Elephant

Using Pyspark

PySpark played a crucial role in the recommendation system by enabling efficient data processing and computation on large datasets. PySpark's ability to distribute tasks made it well-suited for handling extensive track data. Additionally, PySpark provides a scalable and parallelized environment, ensuring that computations are distributed across multiple nodes, leading to faster processing times.

We utilized PySpark to process and analyze a large dataset for music recommendation. PySpark was employed to perform TF-IDF (Term Frequency-Inverse Document Frequency) feature extraction on text data, specifically artist genres, converting them into numerical feature vectors. Additionally, we calculated cosine similarity between user preferences and the music tracks' features, effectively comparing users' listening histories with song characteristics to generate personalized recommendations. PySpark's distributed computing capabilities allowed us to parallelize data processing, making it suitable for handling large-scale music data efficiently and applying TF-IDF and cosine similarity to enhance the recommendation model's accuracy and scalability.

By Devahuti Talukder

Why we chose our model?

- Effective Feature Extraction: TF-IDF efficiently converts textual data (artist genres) into numerical features, allowing the integration of text-based information into your recommendation model.

- Accurate Item Similarity: Cosine similarity is a robust metric for comparing items. It provides a reliable way to measure the similarity between tracks or genres, ensuring accurate and relevant recommendations.

- Interpretable Recommendations: The resulting cosine similarity scores are intuitive and interpretable, helping users understand why a particular track or genre is recommended, thereby enhancing the user experience.

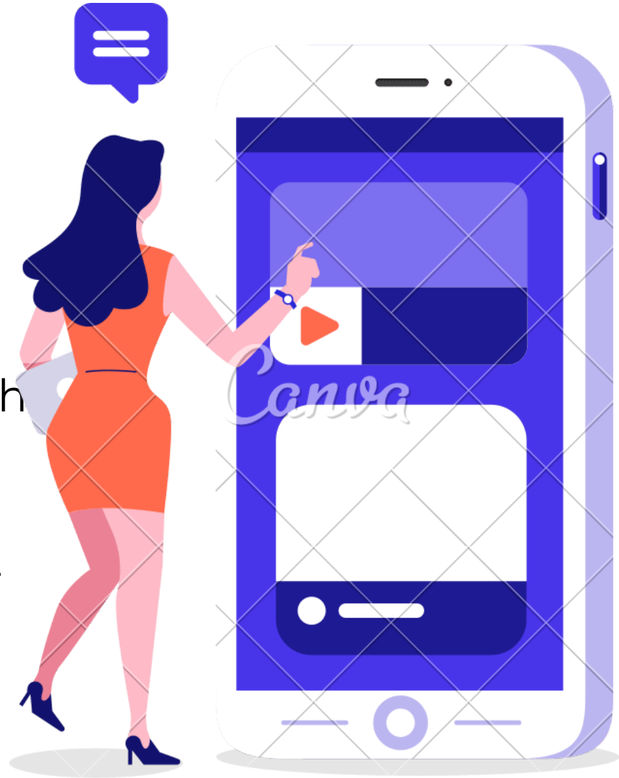
By Devahuti Talukder

Disadvantages

- Cold Start Problem: TF-IDF and cosine similarity-based models may struggle with new tracks or users (cold start problem) because they rely heavily on existing user interaction data. New items or users with limited history may not receive accurate recommendations.

- Lack of Personalization: These techniques primarily focus on item or content similarity and may not capture the full context of user preferences. They lack user-specific information and collaborative filtering aspects that can lead to more personalized recommendations.

- Limited Context: TF-IDF doesn't consider the temporal or sequential aspects of user behavior. It treats all user interactions equally without considering the order or timing of those interactions.



By Devahuti Talukder

Models to Explore

- Matrix Factorization: Matrix Factorization is a powerful recommendation technique that involves breaking down the user-item interaction matrix into two lower-dimensional matrices, one for users and one for items. It works as follows:
- User-Item Interaction Matrix: You start with a matrix where rows represent users, columns represent items, and the values represent user-item interactions (e.g., ratings or implicit feedback).
- Factorization: Matrix factorization aims to factorize this matrix into two matrices: U (user factors) and V (item factors). The product of these matrices approximates the original user-item matrix.
- Personalized Recommendations: To make recommendations for a user, you use the user's embedding from the U matrix and the embeddings of items the user hasn't interacted with from the V matrix. You calculate predicted ratings or scores for these items and recommend the top-rated ones.

By Devahuti Talukder

Future Goals

User-Item Interaction Matrix

- Start with a user-item interaction matrix.
- Each row represents a user, and each column represents an item (e.g., songs).
- The values in the matrix represent user interactions with items, such as the number of times users have played songs.

Matrix Factorization

- Matrix factorization is a technique to break down the user-item interaction matrix into two lower-dimensional matrices: U (user factors) and V (item factors).
- The goal is to approximate the original matrix R as a product of U and V.
- The equation R ≈ U * V illustrates this relationship.

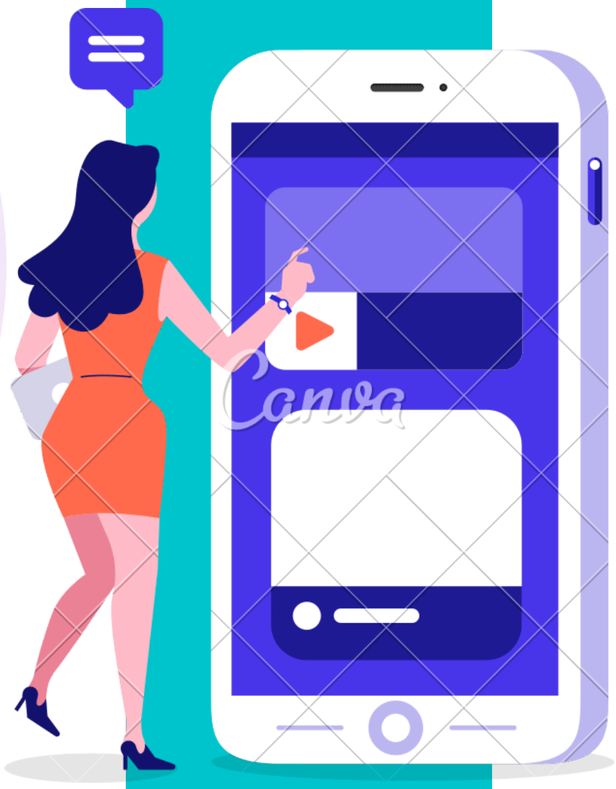
Formula:

$$R_{ij} \approx \mu + user\_bias_i + item\_bias_j + U_i * V_j$$

Matrix factorization models can be fine-tuned using optimization techniques like stochastic gradient descent. These models learn latent factors and biases that capture user preferences, item qualities, and personalized behavior.

By Devahuti Talukder

MongoDB Querying



- Planning to create a **MongoDB-based query system** in HTML using JavaScript that will allow users to filter songs based on various audio features. The goal is to provide a user-friendly interface where users can apply filters to search and retrieve songs from a dataset, making it easy to discover music that matches their preferences.
- Possible combinations of features to **filter songs** in a meaningful way include:
  - Danceable and Energetic Songs: Users can filter for songs with a high danceability value (close to 1.0) and high energy value (close to 1.0) to find upbeat and lively tracks suitable for dancing.
  - Acoustic and Low-Energy Tracks: If someone prefers acoustic and mellow music, they can filter for songs with a high acousticness value (close to 1.0) and low energy value (close to 0.0).
  - Speech vs. Music: Users can filter for music that contains spoken words (speechiness) or music without spoken words, depending on their preferences.
  - Valence A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive
- **Customized Combinations**: Working on allowing users to create custom combinations of filters, such as "Acoustic and Live Performances" or "High Energy Major Key Dance Tracks."

By Ankita Anand

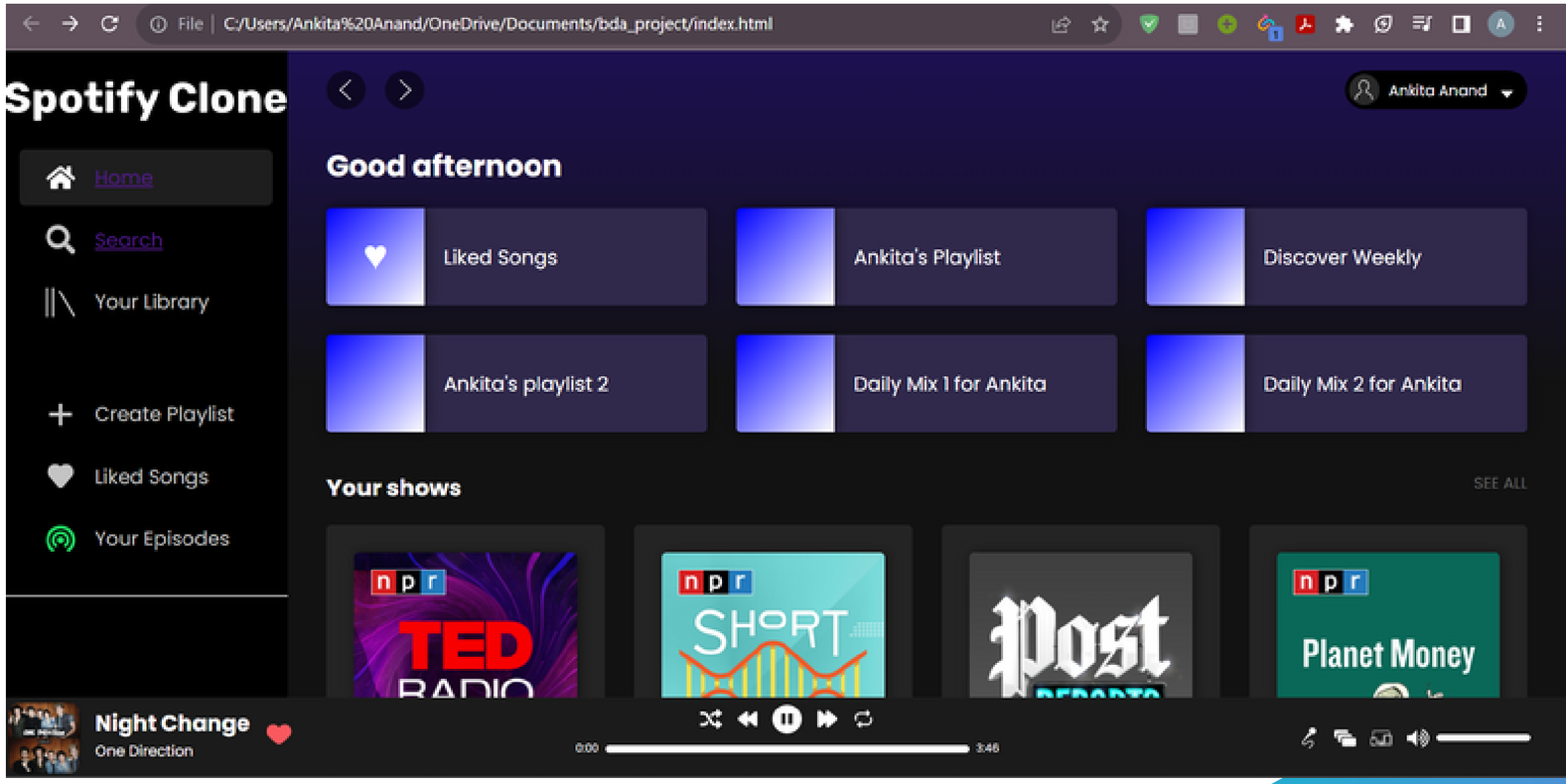
Big Data Searching Algorithms and Techniques

- **MongoDB** Aggregation Framework: MongoDB's aggregation framework is a powerful tool for efficiently filtering, sorting, and manipulating data. It utilizes aggregation pipelines to perform complex queries on your dataset, enabling filtering based on audio features such as danceability, energy, acousticness, speechiness, and valence.
- **PySpark's DataFrame** operations allow you to filter and query your dataset based on specific audio features. You can easily apply conditions to isolate songs that meet criteria like danceability, valence, and more. This means you can efficiently extract songs that match user preferences using PySpark's versatile DataFrame API.
- We can read and write data from MongoDB efficiently, connecting my PySpark processing to my MongoDB-based query system.
- **Fuzzy Search**: Implement fuzzy search algorithms to account for typos or variations in user input. This ensures that users can find songs or artists even if they make slight spelling mistakes in their queries.
- **Temporal Search**: Allow users to search for music based on temporal attributes like release date. Users can filter for songs released during specific time periods or within certain years.

By Ankita Anand

Frontend Developments

User Interface: Creating a web application interface that allows users to select and apply filters for various audio features. This interface can consist of checkboxes, sliders, drop-down menus, or text input fields, depending on the nature of the feature.



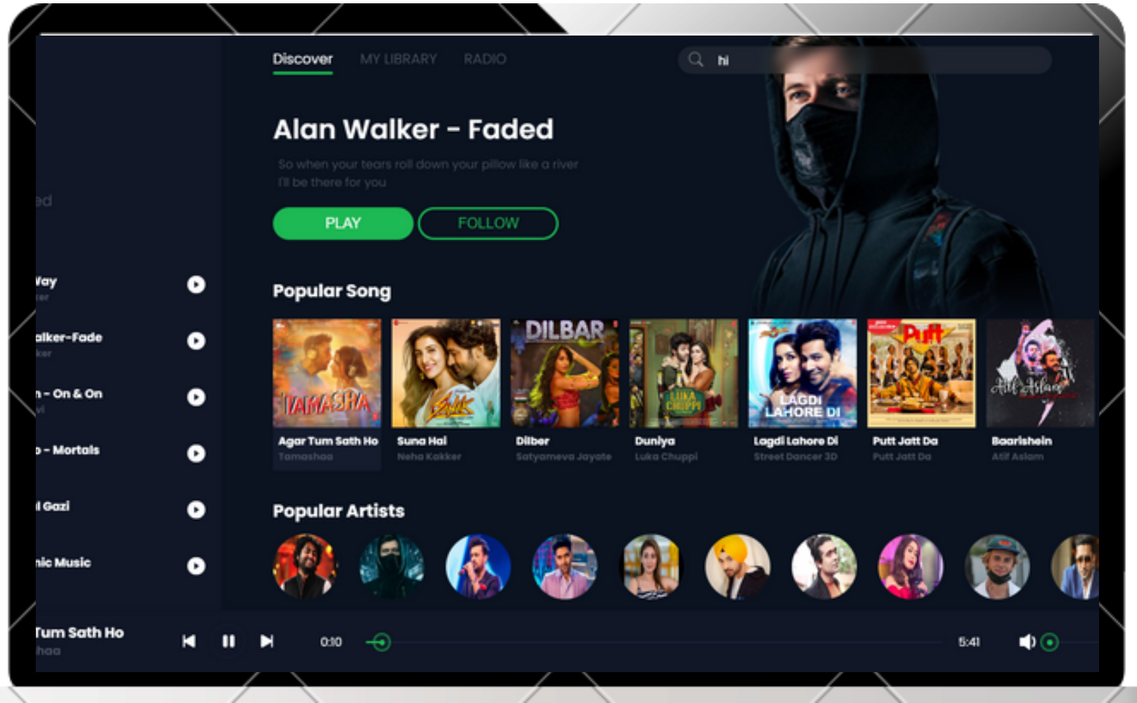
Spotify Clone webpage will be the website home page

By Ankita Anand

Frontend Developments Inspiration

I am working on using this webpage code to integrate my Tracks filtering vision

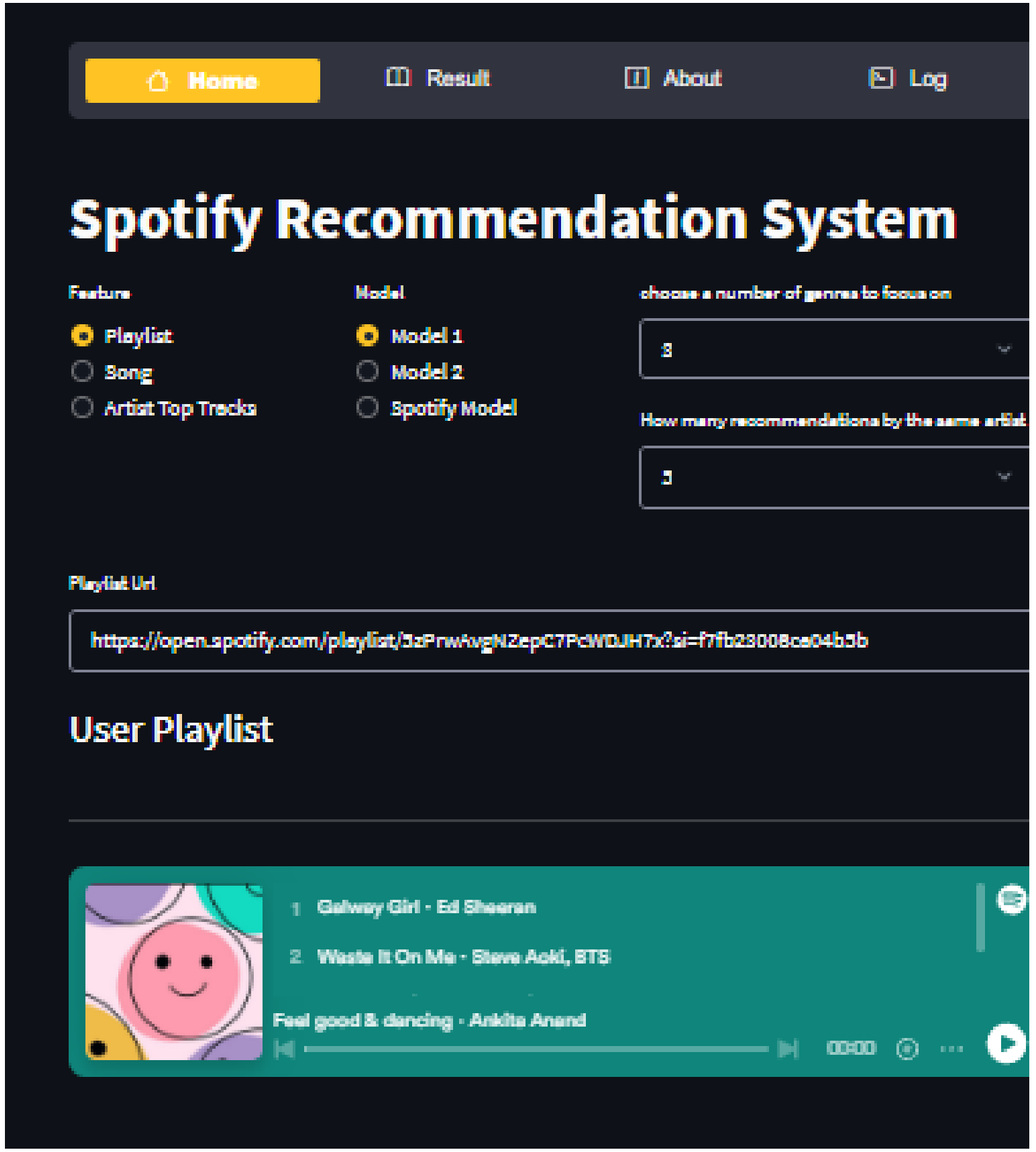
User Interface: Create a web application interface that allows users to select and apply filters for various audio features. This interface can consist of checkboxes, sliders, drop-down menus, or text input fields, depending on the nature of the feature.



By Ankita Anand

Frontend Developments

Streamlit deployment in this project involves creating a web application that allows users to input their playlists and receive song recommendations based on the model's predictions. Users can interact with the deployed web app, input their playlists, and receive song recommendations instantly.



By Ankita Anand

It's Time to Say

Thanks For Watching!

By Ankita Anand