Presentation on Mini Project

SereneSounds

A Sophisticated Music Curation Engine

Submitted by –

Name : Shib Kumar Saraf
Reg. No. : 22352239
MCA 2nd Year(3rd sem)
Pondicherry University
(Karaikal Campus)

Introduction

SereneSounds

Streamlit-based music recommendation web app offering diverse suggestions through lyric and musical similarity, collaborative filtering, ALS algorithm, and artist similarity using graph models.

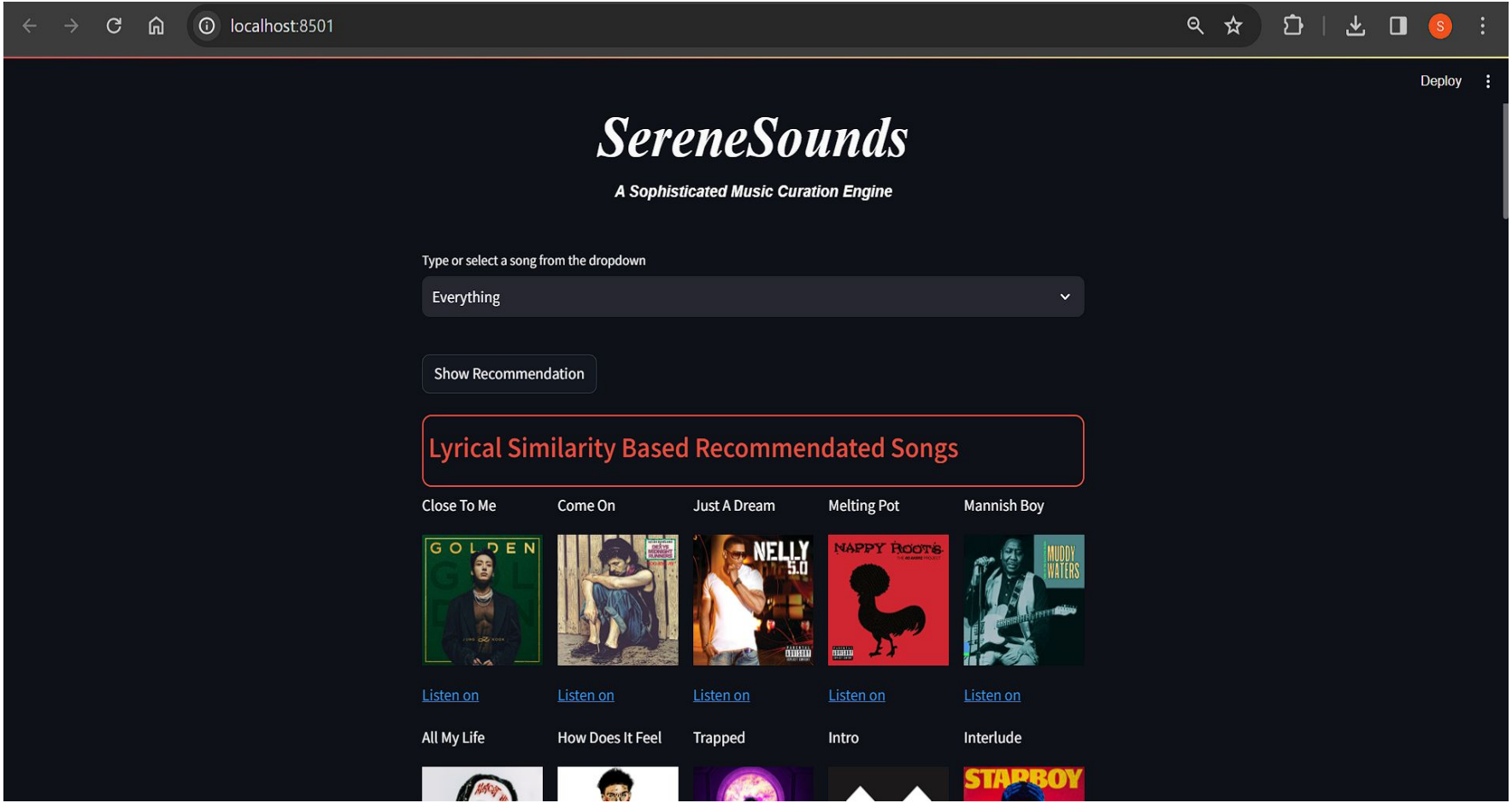
What is Recommendation?

Recommendation is like a smart friend saying, "You might really like this!" based on what you've enjoyed before.

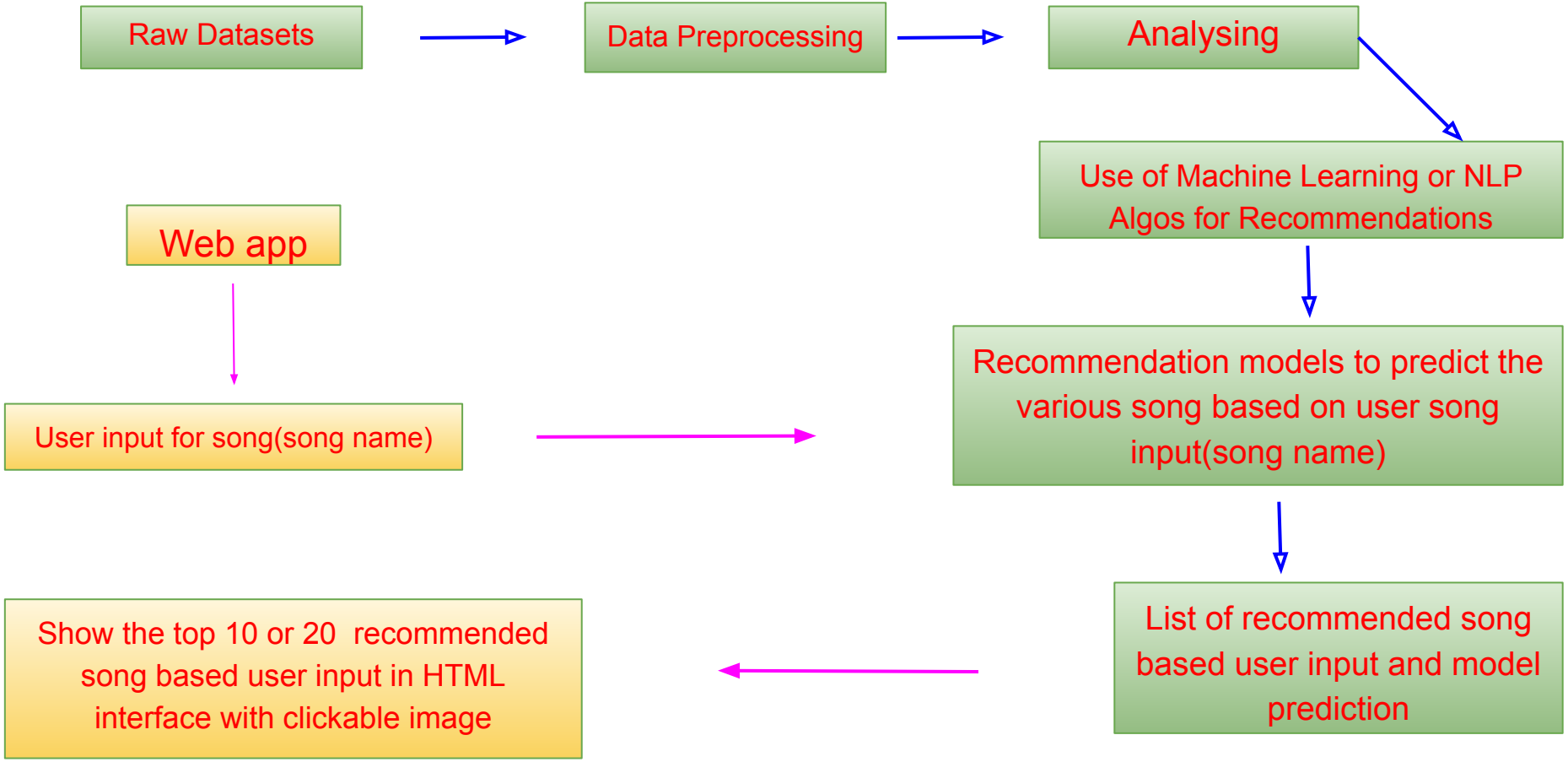
Why It Is Needed?

- *Effortless Music Discovery with Diverse Models.*
- *Tailored Recommendations for Varied Tastes.*
- *Dynamic Adaptation to Evolving Preferences.*
- *Engaging User Experience with Refreshing Approaches.*
- *Inclusive Accuracy through Combined Algorithms.*

Web App


localhost:8501
Deploy
SereneSounds
A Sophisticated Music Curation Engine
Type or select a song from the dropdown
Everything
Show Recommendation
Lyrical Similarity Based Recommendated Songs
Close To Me
Come On
Just A Dream
Melting Pot
Mannish Boy
Listen on
Listen on
Listen on
Listen on
Listen on
All My Life
How Does It Feel
Trapped
Intro
Interlude

Data Flow Diagram



Raw Dataset

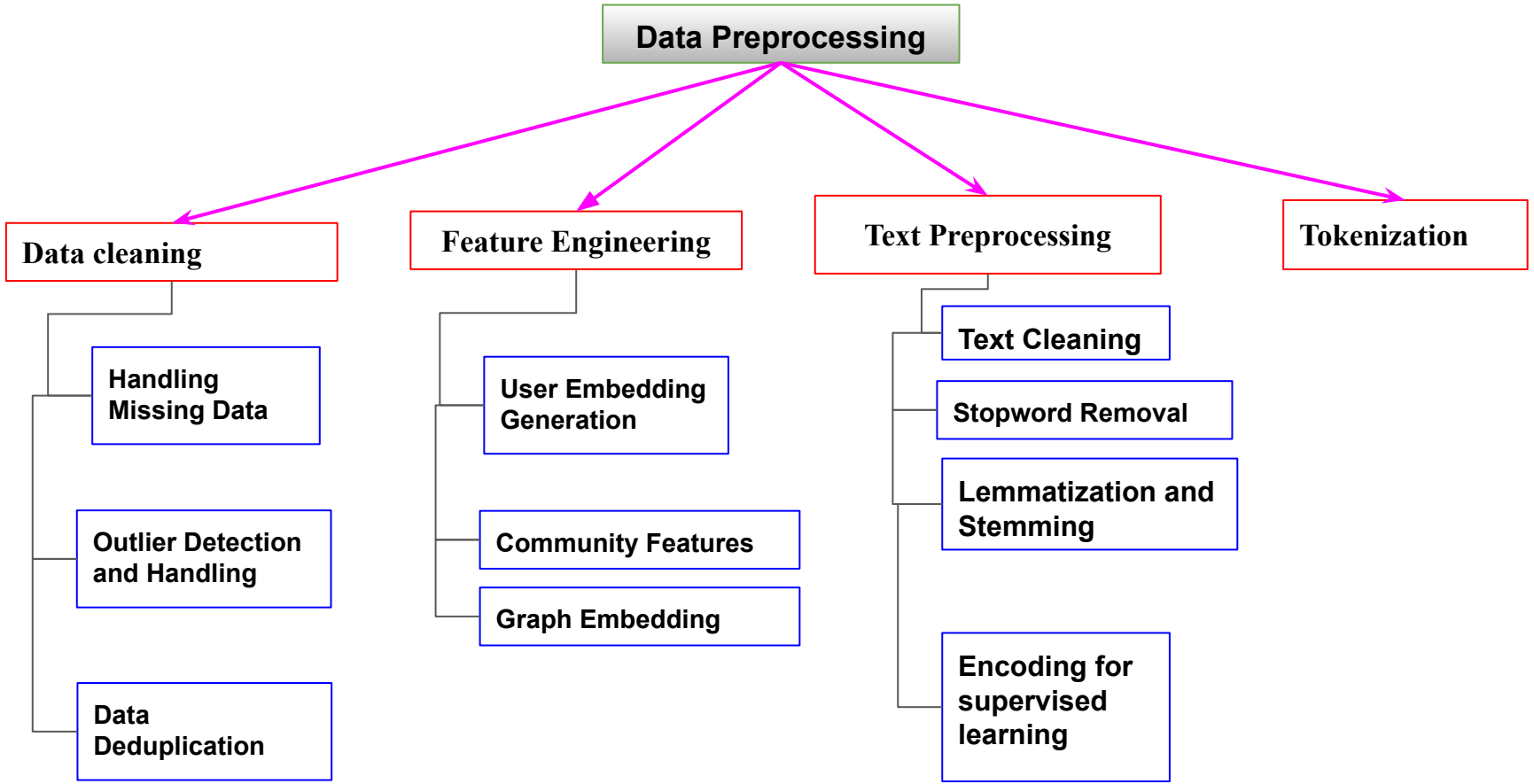
songs.csv

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 187357 entries, 0 to 187356
Data columns (total 21 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   id                187357 non-null  object
 1   name              187357 non-null  object
 2   popularity        187357 non-null  int64
 3   duration_ms       187357 non-null  int64
 4   explicit          187357 non-null  int64
 5   artists           187357 non-null  object
 6   release_date      187357 non-null  object
 7   danceability      187357 non-null  float64
 8   energy            187357 non-null  float64
 9   key               187357 non-null  int64
 10  loudness          187357 non-null  float64
 11  mode              187357 non-null  int64
 12  speechiness       187357 non-null  float64
 13  acousticness      187357 non-null  float64
 14  instrumentalness  187357 non-null  float64
 15  liveness          187357 non-null  float64
 16  valence           187357 non-null  float64
 17  tempo             187357 non-null  float64
 18  time_signature    187357 non-null  int64
 19  link              187357 non-null  object
 20  text              187357 non-null  object
dtypes: float64(9), int64(6), object(6)
memory usage: 30.0+ MB
```

Sample data

	id	name	popularity	duration_ms	explicit	artists	release_date	danceability	energy	key	...	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	time_signature	link	text
0	1v9EovC0CD8tKq88ZEI3Ri	Everything	46	240973	0	['ARASHI']	2009-07-01	0.645	0.892	1	...	1	0.0396	0.413000	0.000000	0.1160	0.922	121.015	4	/y/young+jeezy/everything_20757378.html	Whats up? you got a man?I what the fuck is tha...
1	77QLO6fafr5BGeV2ADuFKz	Ooh La La	44	256021	0	['Bappi Lahiri', 'Shreya Ghoshal']	2011-10-29	0.742	0.972	6	...	1	0.1500	0.102000	0.000033	0.0563	0.766	126.049	4	/b/britney+spears/ooh+la+la_21061266.html	You don't have to look like a movie star \nOo...
2	7IEVmnzJr1V7duYKPWplzp	Bringin' On The Heartbreak	59	273733	0	['Def Leppard']	1981-07-11	0.329	0.676	9	...	0	0.0324	0.036600	0.000000	0.1600	0.529	78.217	4	/d/def+leppard/bringin+on+the+heartbreak_20038...	Gypsy, sittin' lookin' pretty \nThe broken ro...
3	52am0fzNdEwaQpRKoxMmfY	Have Yourself A Merry Little Christmas	3	166907	0	['Tex Beneke And His Orchestra']	1936	0.192	0.152	8	...	1	0.0302	0.873000	0.000017	0.1060	0.165	83.425	3	/d/dem+lovato/have+yourself+a+merry+little+ch...	Oh yeah, mmm \nHave yourself a merry little C...
4	53cfefzTz5SwERUGEuZ0q	This Is The Life	55	185973	0	['Amy Macdonald']	2007	0.516	0.898	1	...	0	0.0452	0.240000	0.000000	0.0647	0.817	190.068	4	/w/weird+al+yankovic/this+is+the+life_20145870...	I eat filet mignon seven times a day \nMy bat...

Data Preprocessing



Data Analysing

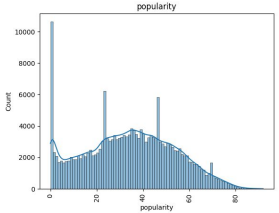
Statistical Analysis

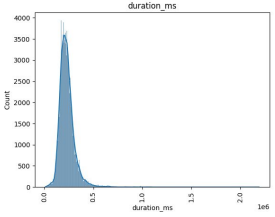
```
song_data.describe()
```

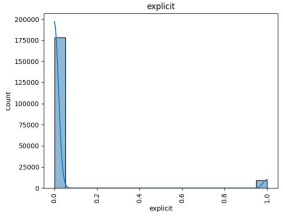
	popularity	duration_ms	explicit	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	time_signature
count	187357.000000	1.873570e+05	187357.000000	187357.000000	187357.000000	187357.000000	187357.000000	187357.00000	187357.000000	187357.000000	187357.000000	187357.000000	187357.000000	187357.000000	187357.000000
mean	34.072679	2.290143e+05	0.048714	0.543940	0.534185	5.169447	-10.018579	0.68449	0.059560	0.400187	0.108215	0.179891	0.488158	118.023097	3.892014
std	19.579817	7.738818e+04	0.215271	0.165694	0.258453	3.517247	4.757214	0.46472	0.065282	0.354469	0.256931	0.142298	0.254025	29.109832	0.414600
min	0.000000	1.140000e+04	0.000000	0.000000	0.000020	0.000000	-43.821000	0.00000	0.000000	0.000000	0.000000	0.009670	0.000000	0.000000	0.000000
25%	19.000000	1.808800e+05	0.000000	0.432000	0.320000	2.000000	-12.923000	0.00000	0.032000	0.047800	0.000000	0.095700	0.280000	95.506000	4.000000
50%	35.000000	2.185870e+05	0.000000	0.555000	0.542000	5.000000	-9.297000	1.00000	0.039100	0.305000	0.000066	0.124000	0.475000	117.546000	4.000000
75%	49.000000	2.626930e+05	0.000000	0.663000	0.751000	8.000000	-6.291000	1.00000	0.056200	0.755000	0.013400	0.219000	0.695000	135.952000	4.000000
max	92.000000	2.193058e+06	1.000000	0.986000	0.999000	11.000000	2.663000	1.00000	0.960000	0.996000	0.992000	0.995000	0.992000	238.895000	5.000000

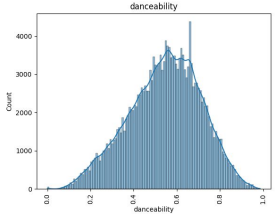
Data Analysing

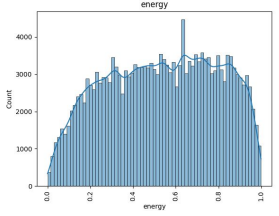
Analysis the data of columns using Charts

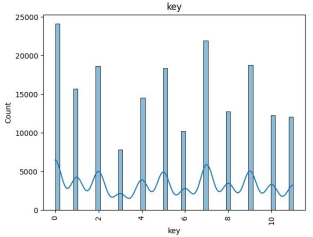


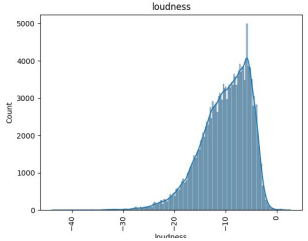


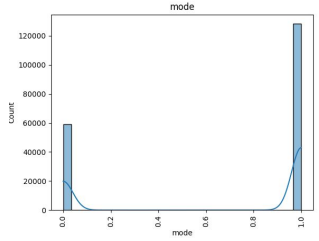


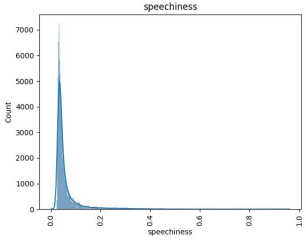


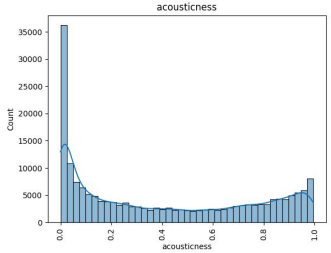


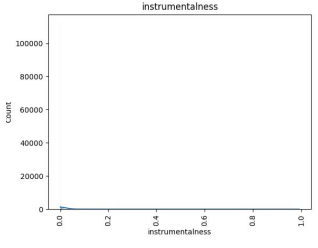


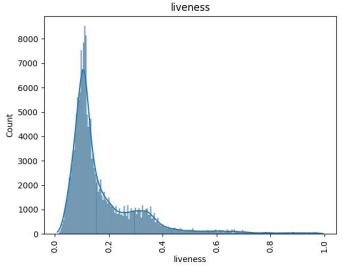


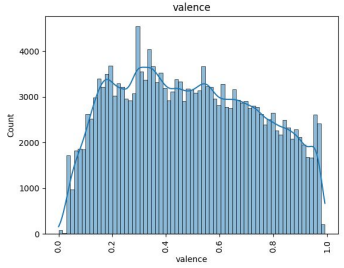


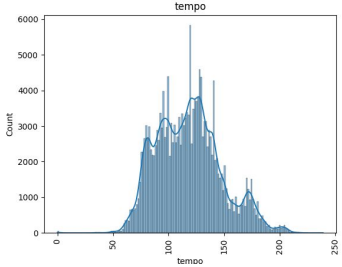


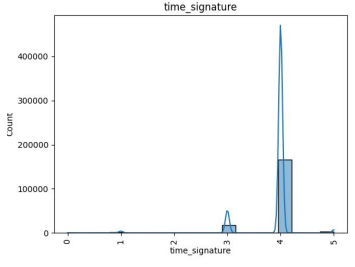






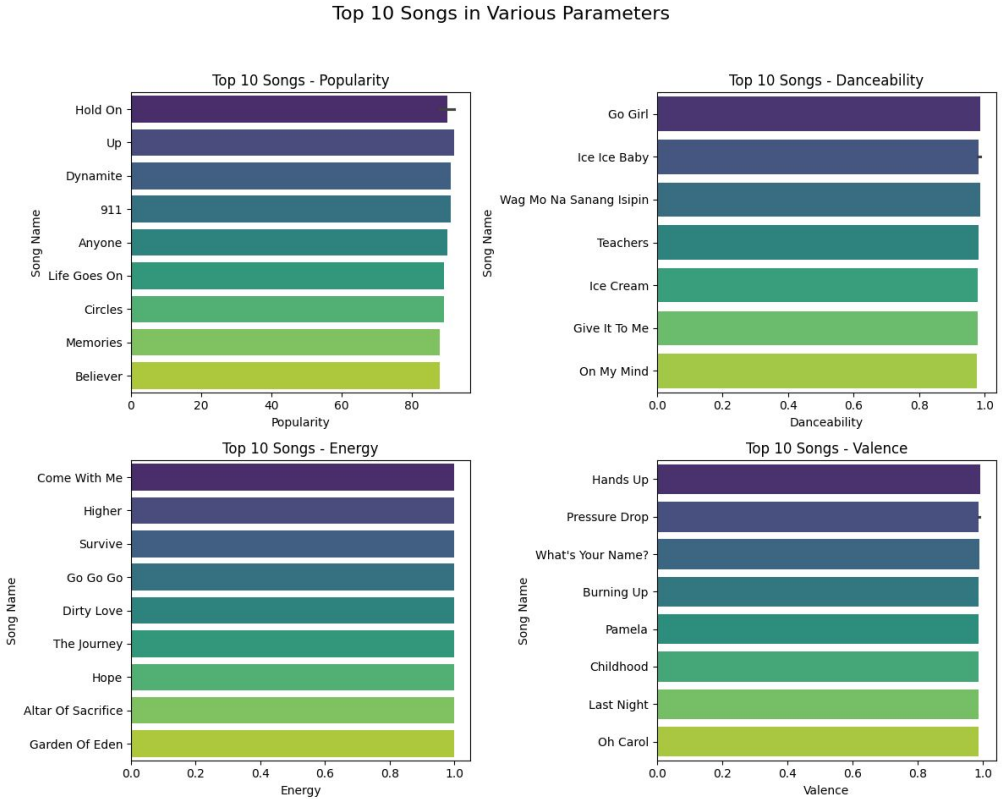






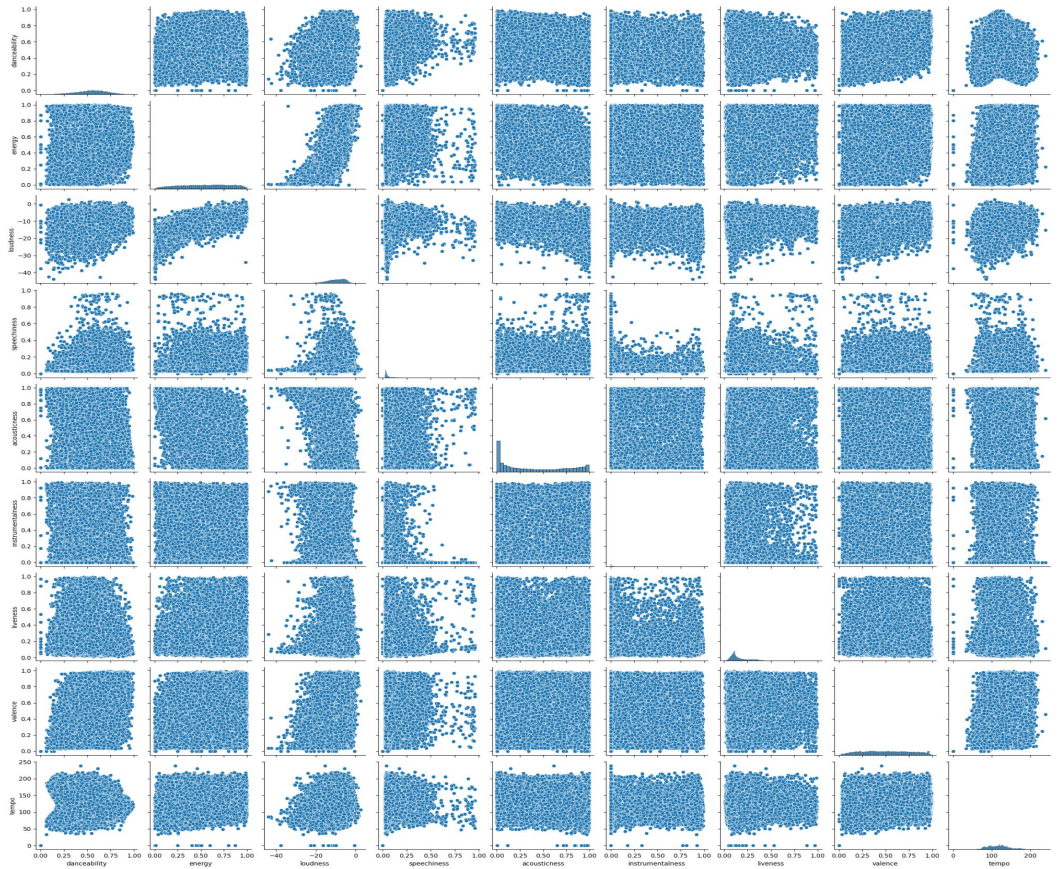
Data Analysing

Top 10 Songs in Various Parameters



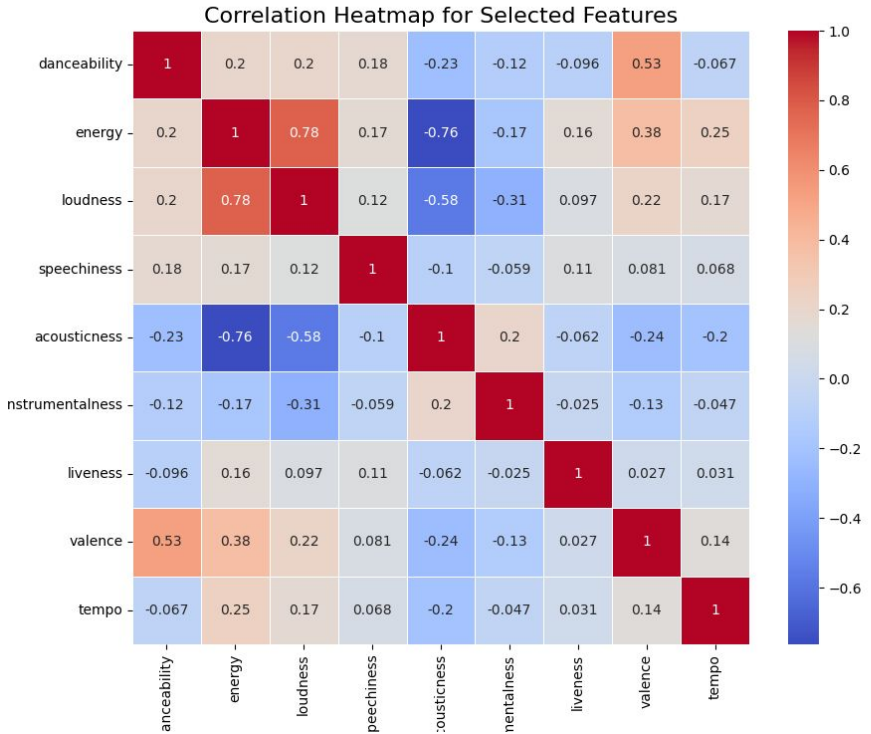
Data Analysing

Pairplots for Various Parameters

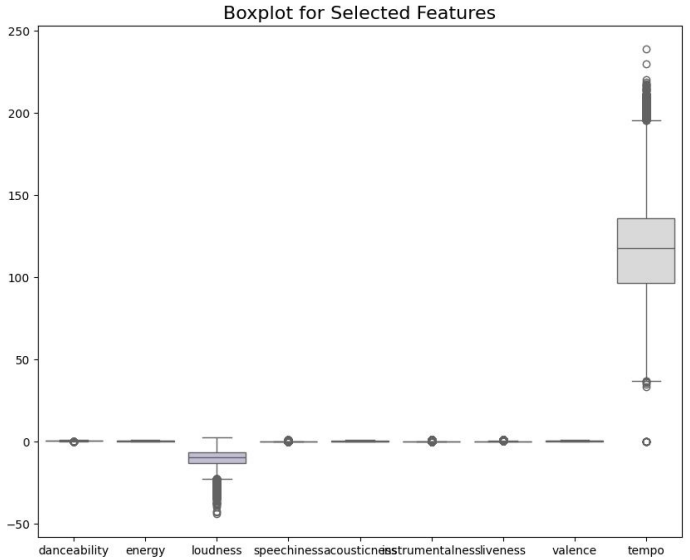


Data Analysing

Correlation Heatmap for Selected Features

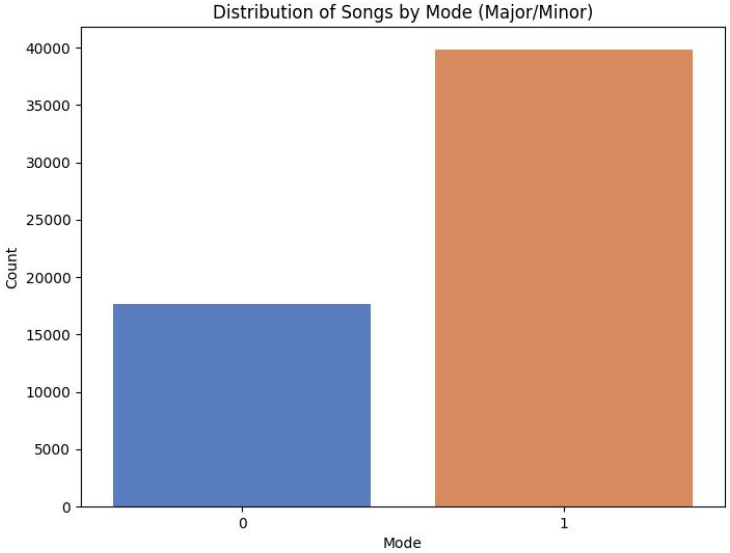


Boxplot for Selected Features

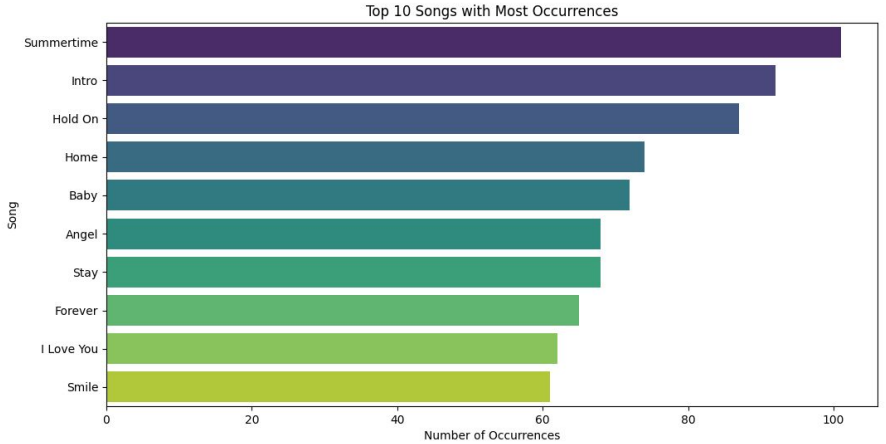


Data Analysing

Distribution of Songs by Mode (Major/Minor)

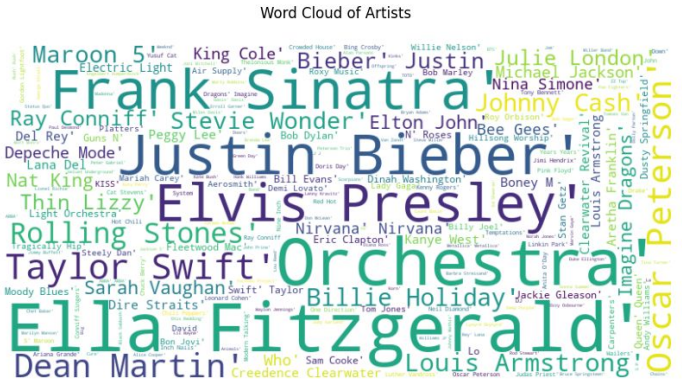


Top N Songs with Most Occurrences

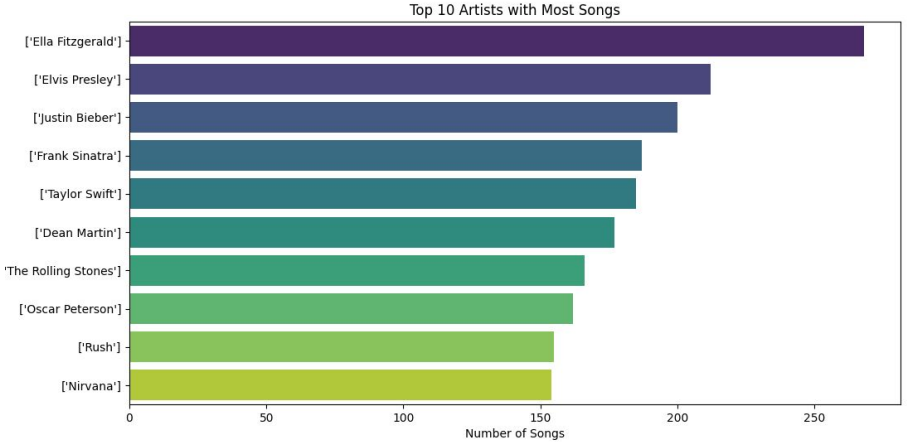


Data Analysing

Word Cloud for Song artists

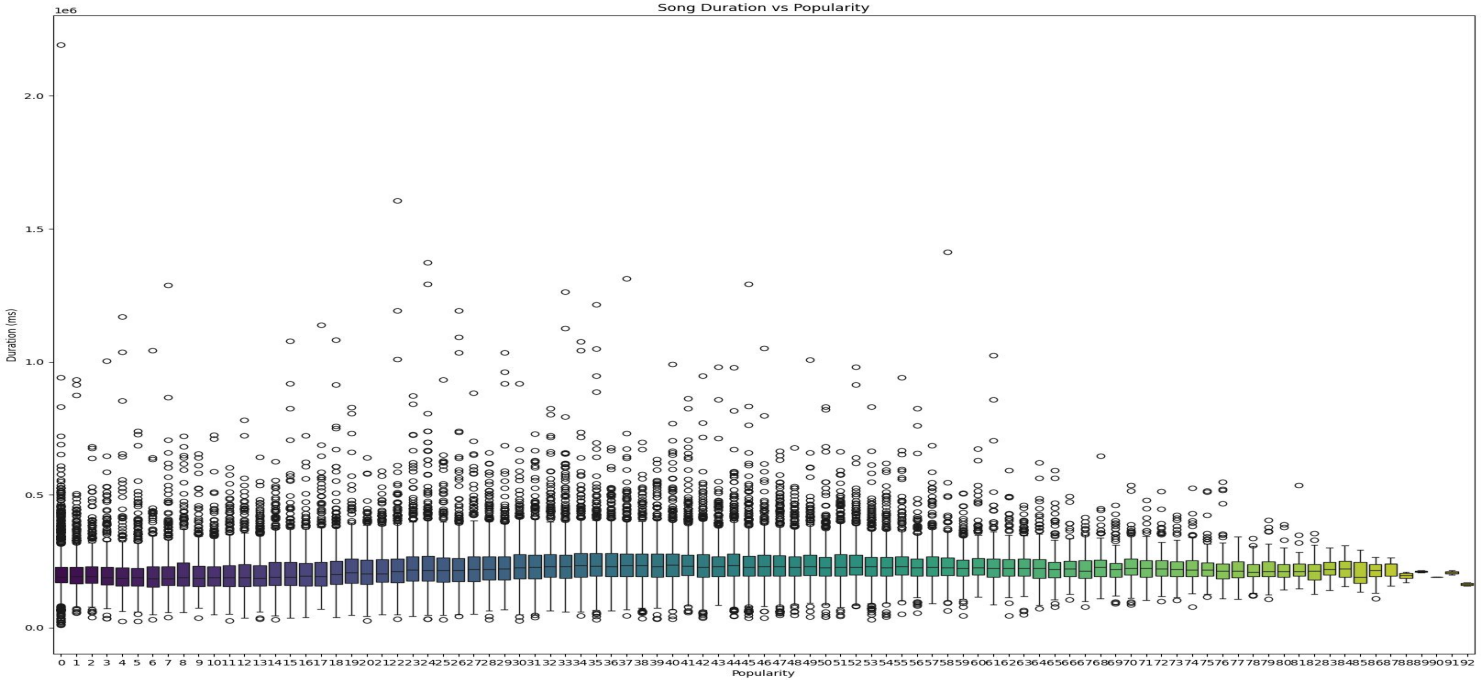


Top N Artists with Most Songs



Data Analysing

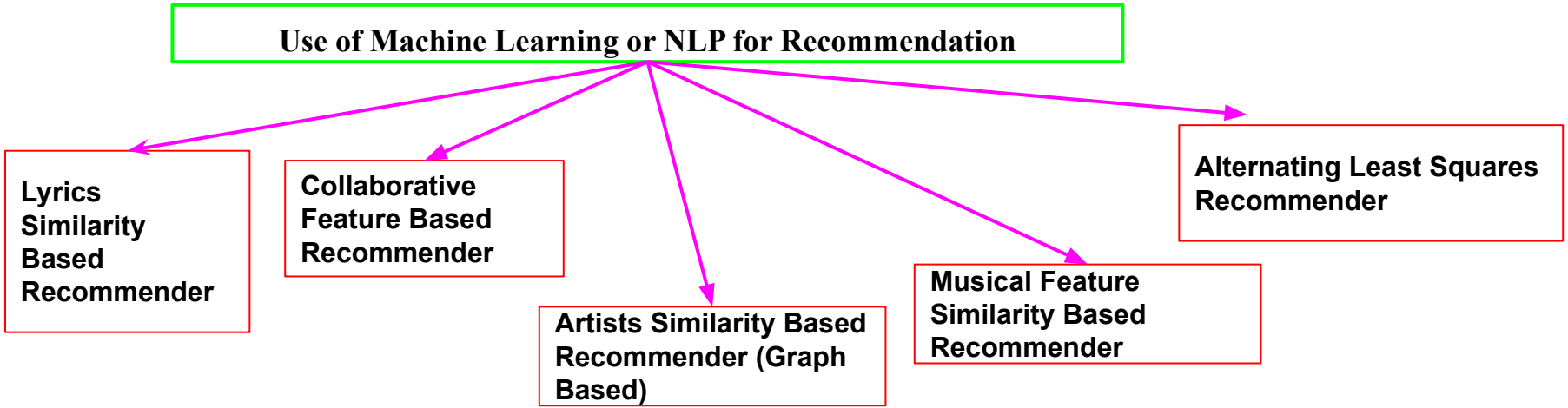
Popularity Distribution



Data Analysing

Word Cloud for Song Lyrics (using a library like wordcloud)

Recommendation models



Lyrics Similarity Based Recommender

Data Preprocessing:

- Text data is cleaned by converting to lowercase, removing non-alphanumeric characters, punctuation, and replacing newline characters.

Feature Engineering:

- A 'tags' column is created by combining song names and preprocessed text.

Text Normalization:

- Stemming, lemmatization, and stop word removal are applied to 'tags' for text normalization.

TF-IDF Vectorization:

- TF-IDF: The product of **Term Frequency (TF)** and **Inverse Document Frequency (IDF)** it quantifies the importance of a term in the context of a specific document and the entire corpus. Words with high TF-IDF scores are often important in representing the content of a document.
- Text is transformed into a TF-IDF matrix, capturing term importance in the corpus.

Recommendation System:

- **Cosine similarity** is calculated from TF-IDF matrix , that is used to recommend songs similar to a given input, demonstrated with an example recommending songs similar to "Friday."

Example usage

```
: get_recommendations("Friday")
```

```
Recommended Songs:
+------------------+------------------+
|       Song       | Similarity Score |
+------------------+------------------+
|  I Stay In Love  |      28.76%      |
|     La La La     |      28.76%      |
|     Frantic      |      28.58%      |
| I Believe In You |      28.55%      |
|   Love Machine   |      28.17%      |
| This Is Our Song |      27.28%      |
|   Rotten Apple   |      25.58%      |
|  Birthday Song   |      25.31%      |
|     So What      |      21.53%      |
+------------------+------------------+
```

Musical Feature Similarity Based Recommender

Feature Concatenation:

- Musical features such as danceability and energy are converted to strings and combined into a new column, 'features_str,' for each song in the dataset.

TF-IDF Vectorization:

- The 'features_str' column undergoes TF-IDF vectorization, transforming the musical features into a matrix that captures their importance across songs.

Similarity Computation:

- Cosine similarity is computed among songs based on their TF-IDF representations of musical features, quantifying the similarity between songs in the high-dimensional feature space.

Recommendation Function:

- The code defines a function, get_recommendations, which utilizes the computed similarity scores to recommend songs that share musical feature patterns. The top 10 recommendations are displayed along with their respective similarity scores.

```
get_recommendations("Friday")
```

```
Recommended Songs:
+-----------------+------------------+
|      Song       | Similarity Score |
+-----------------+------------------+
|  Computer Blue  |      24.21%      |
|      Solo       |      21.61%      |
|      Lies       |      18.89%      |
|  Forever Young  |      17.68%      |
|     Heaven      |      16.94%      |
|     Hold Me     |      16.62%      |
| I Want You Back |      16.47%      |
|     No Love     |      16.44%      |
|   Photograph    |      15.26%      |
+-----------------+------------------+
```

Collaborative Feature Similarity Based Recommender

Sorting:

- The function sorts the song_data DataFrame based on popularity, time signature, duration, and explicitness in descending, ascending, ascending, and descending order, respectively.

User Song Filtering:

- The user's provided song is filtered from the dataset. If the provided song is not in the dataset, a message is returned indicating so.

Criteria for Recommendation:

- The function extracts attributes such as time signature, duration, and explicitness from the user's song.
- It then filters songs with similar time signature, higher popularity, similar duration (within the median duration of the dataset), and opposite explicitness.

Exclusion of Redundant and Same-Name Songs:

- Redundant and same-name songs are excluded from the recommendations.

Final Recommendation:

- If there are suitable recommended songs, the function returns a list of up to 10 unique song names; otherwise, it returns a message indicating that no similar songs with higher popularity were found.

Example usage:

```
user_input_song = "Beautiful"
recommendations = recommend_song(song_data, user_input_song)
print(recommendations)
```

```
['You', 'Smile', 'Fake Love', 'Lost', 'Jungle', 'Ghost', 'King For A Day', 'Power', 'Happy Song', 'Love Me']
```

Artists Similarity Based Recommender (Graph Based)

Graph Construction:

- A NetworkX graph (G) is created to represent artist collaborations.
- For each song, the code iterates through the artists and creates edges between pairs of artists. The weight of each edge represents the number of collaborations between the artists.

Graph Serialization (Saving and Loading):

- The graph (G) is serialized and saved to a pickle file named "graph_data.pkl."

Graph Deserialization:

- The serialized graph is loaded back from the pickle file.

Recommendation Function:

- The function recommend_songs takes an input song name, the graph, and an optional parameter for the number of recommendations.
- It finds the artist(s) associated with the input song.
- Identifies similar artists by exploring the graph and excluding the input song's artist(s).
- Retrieves songs by the similar artists, excluding the input song.
- Sorts the recommendations by popularity and selects the top ones.
- Ensures uniqueness in the list of recommended song names.
- Returns a list of unique recommended song names.

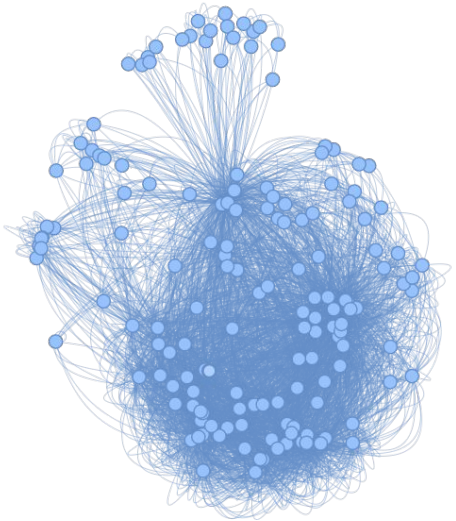
Example usage

```
input_song_name = "Beautiful"
recommendations = recommend_songs(input_song_name,G)
recommendations
```

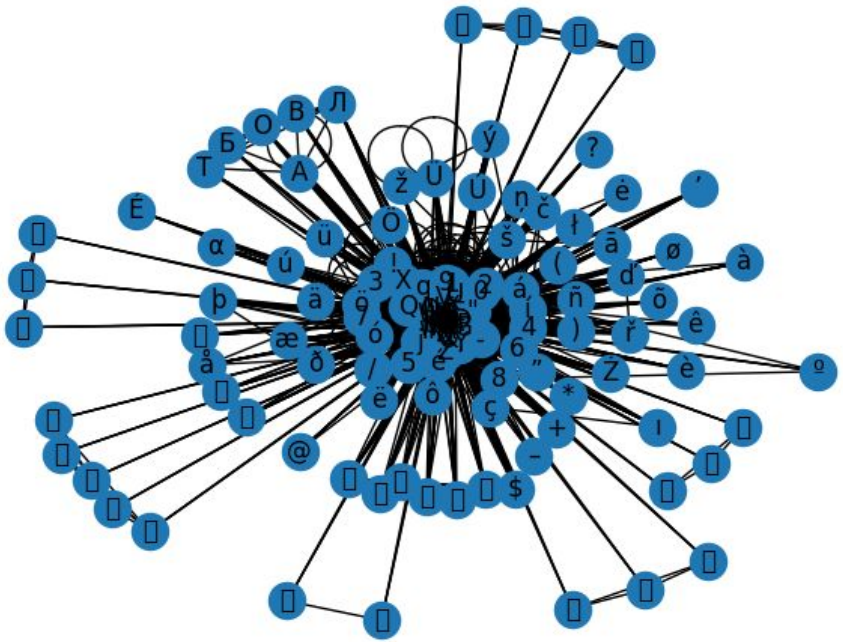
```
['One More Time',
 'Numb',
 'Stay',
 'Bruises',
 'Hold On',
 'Stronger',
 'Gone',
 'Waiting For Love',
 'Trust',
 'Wake Me Up Before You Go-Go',
 'Memories',
 'The Scientist',
 'Photograph',
 'Best Friend',
 'Someone Like You']
```

Artists Similarity Based Recommender (Graph Based)

Visualize Graph Network



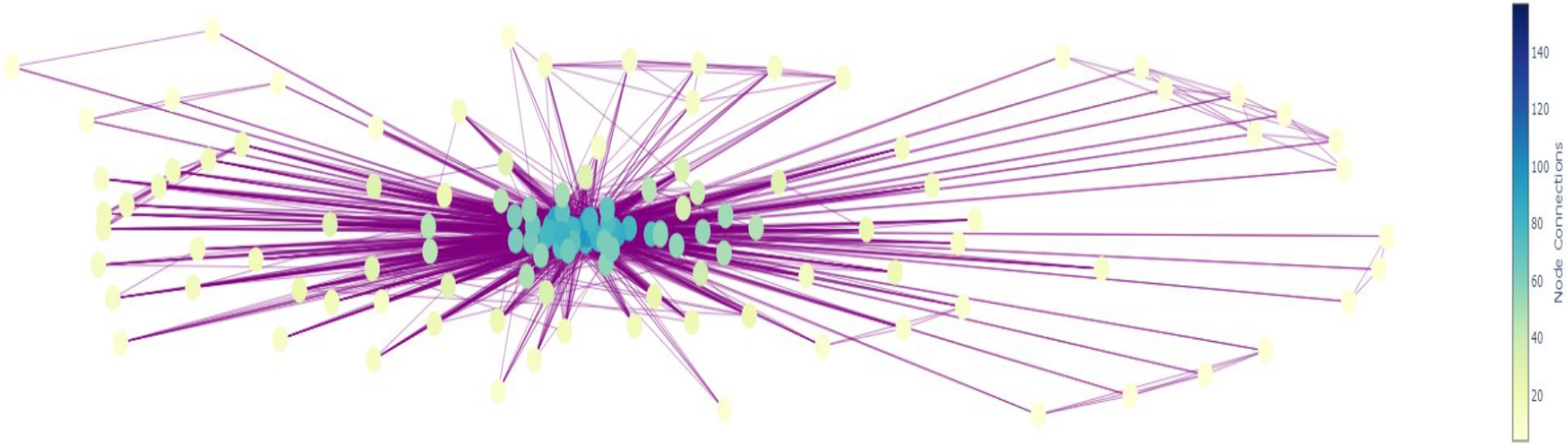
undirected graph where nodes represent artists, and edges between nodes represent collaborations between those artists on songs.



Artists Similarity Based Recommender (Graph Based)

Visualize Graph Network

undirected graph where nodes represent artists, and edges between nodes represent collaborations between those artists on songs.



Alternating Least Squares Recommender

Alternating Least Squares :

It is a collaborative filtering algorithm used for matrix factorization. Collaborative filtering is a technique that makes automatic predictions (filtering) about the preferences of a user by collecting preferences from many users (collaborating).

User-Item Matrix:

- The features for the collaborative filtering model are selected, and a user-item matrix is created using the Compressed Sparse Row (CSR) matrix representation.

ALS Model Initialization and Training:

- An ALS model is created with specified parameters such as the number of factors, regularization, and iterations.
- The model is trained on the user-item matrix using the fit method.

Model Serialization (Saving and Loading):

- The trained ALS model is saved to a file named 'als_model.pkl' using the pickle module.
- The saved model is then loaded back from the pickle file.

Factor Matrices:

- The user and item factors matrices are extracted from the loaded ALS model, and their shapes are printed.

Recommendation Function:

- The function get_recommendations takes a track name, the ALS model, the song dataset (song_data), and an optional parameter for the number of recommendations.
- It calculates the dot product between the user vector of the specified track and the item factors matrix.
- The indices of tracks with the highest scores are obtained.
- Redundant and same-name tracks are filtered out, and a list of recommended track names is returned.

Alternating Least Squares Recommender

Creating a feature matrix

```
user_item_matrix = csr_matrix(song_data[features].values)
user_item_matrix
```

```
<5000x9 sparse matrix of type '<class 'numpy.float64'>'
        with 43543 stored elements in Compressed Sparse Row format>
```

Creating an ALS model and fitting it to data

```
# Create an ALS model
model = AlternatingLeastSquares(factors=50, regularization=0.01, iterations=100)

# Fit the model to the feature matrix
model.fit(user_item_matrix)

# Save the best model to a file using pickle
with open('als_model.pkl', 'wb') as file:
    pickle.dump(model, file)
```

```
100%|██████████| 100/100
```

Load the best model from the file ¶

```
with open('als_model.pkl', 'rb') as file:
    loaded_best_model = pickle.load(file)
```

```
loaded_best_model
```

```
<implicit.cpu.als.AlternatingLeastSquares at 0x24a77801090>
```

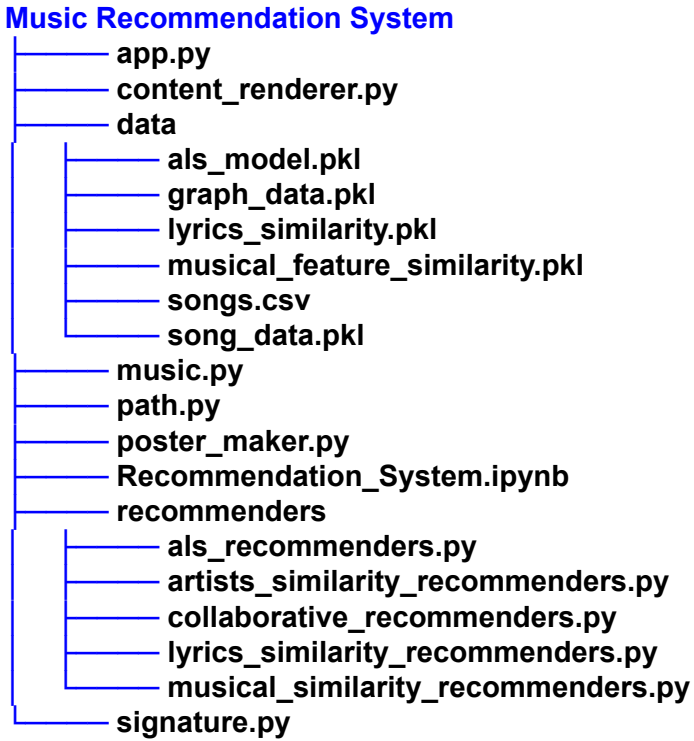
```
user_factors = loaded_best_model.user_factors
item_factors = loaded_best_model.item_factors

# Print the shape of user and item factors matrices
print("User Factors Shape:", user_factors.shape)
print("Item Factors Shape:", item_factors.shape)
```

```
User Factors Shape: (5000, 50)
Item Factors Shape: (9, 50)
```

```
# Example usage:
track_name = "Friday"
recommendations = get_recommendations(track_name, loaded_best_model, song_data)
print("Recommendations for Track:", track_name)
print(recommendations)
```

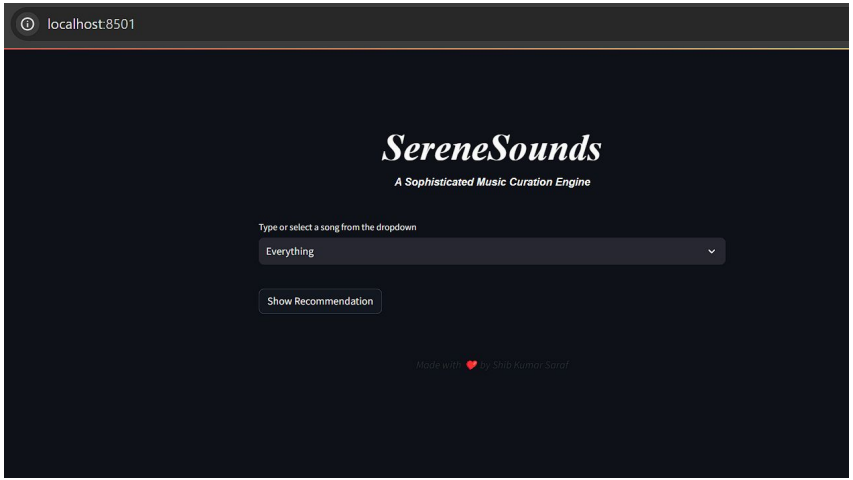
```
Recommendations for Track: Friday
["Bringin' On The Heartbreak", "In Harm's Way", 'This Is The Life', 'Ooh La La', 'Love', 'Solo', 'Stay', 'Everything', 'Have Yourself A Merry Little Christmas']
```

Web App File Structure

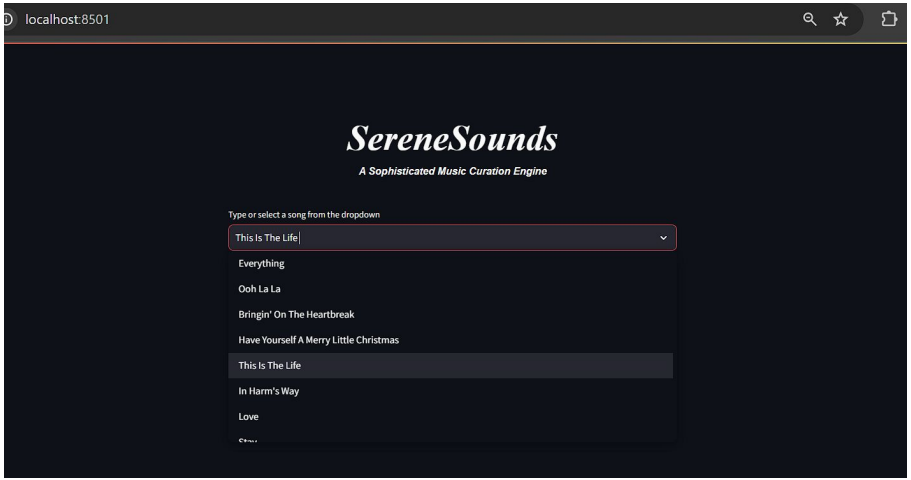


Web App Screenshots

Home Page

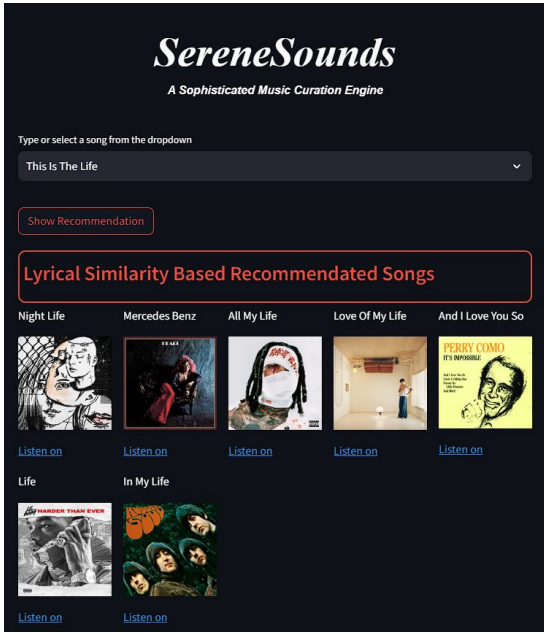


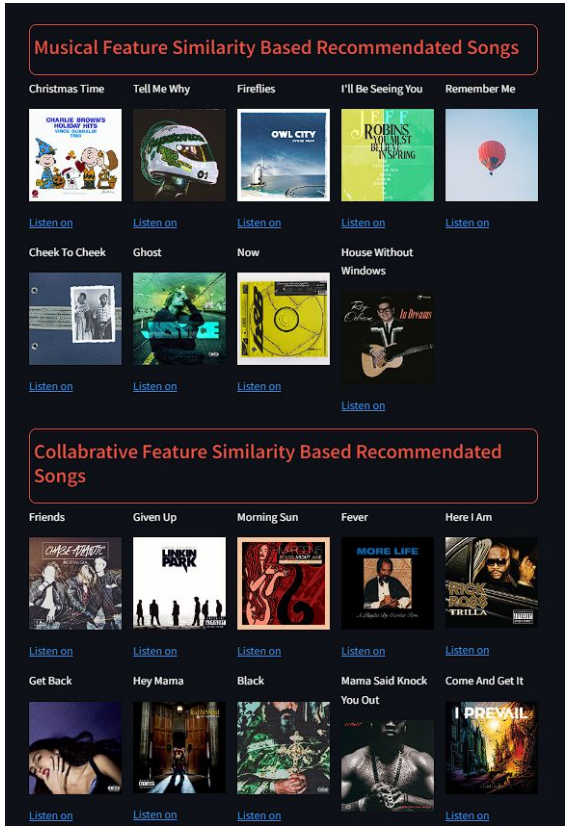
Choose song from Dropdown Menu

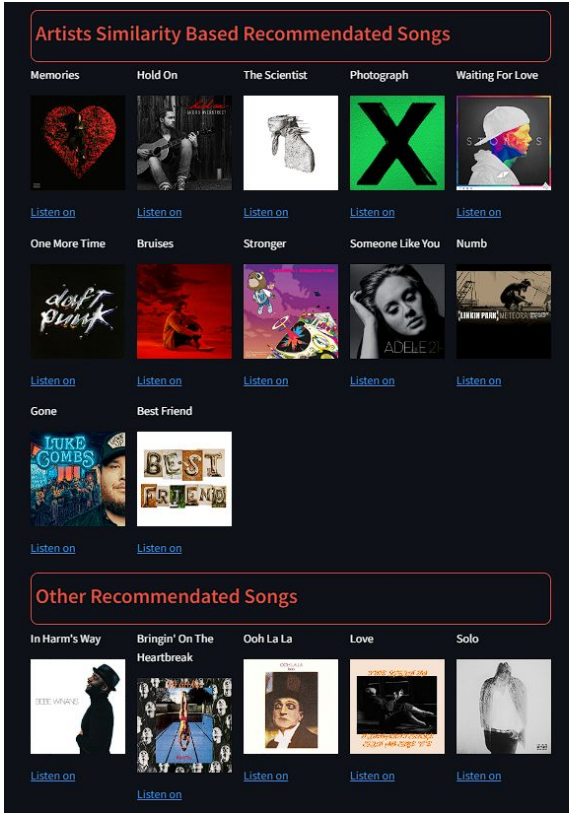


Web App Screenshots

Showing the recommedated song in various category wih music posters and links







THANK YOU …