Music Recommendation System

Final Project - CS 6220 - Data Mining Techniques

Contributors (Team Tulip)

1. Anusha Kalbande : kalbande.a@northeastern.edu
2. Krina Jitendrabhai Devani: devani.k@northeastern.edu
3. Neha Joshi: joshi.neh@northeastern.edu
4. Vidhi Anand Thacker: thacker.v@northeastern.edu

Contents

1. Abstract

The rapid evolution of technology and music streaming platforms has transformed the music consumption landscape, providing users with unprecedented access to a vast array of songs. However, the abundance of choices presents a significant challenge for users to discover music aligned with their tastes. Concurrently, music streaming platforms like Spotify, YT Music, etc. grapple with efficiently managing extensive song catalogs and assisting customers in navigating through the multitude of options. To address these challenges, our project aims to design, implement, and analyze a robust song recommendation system using the 1 Million Song Dataset.

Our project focuses on producing tailored suggestions by learning from users' listening history by leveraging vast repositories of song and user interaction information. The selected dataset is a significant resource for discovering correlations between users and songs, allowing us to construct a variety of recommendation algorithms.

We employ a variety of recommendation strategies, such as Content-Based Models, Singular Value Decomposition (SVD), Collaborative Filtering, and Popularity-Based Models. By taking into consideration several elements including general popularity, user preferences, hidden patterns in interactions, and content aspects, these algorithms work together to deliver users personalized song recommendations.

Throughout the project, we compare each recommendation strategy, considering factors such as accuracy, diversity, and computational efficiency. Additionally, we explore the impact of dataset size on algorithm performance, providing insights into scalability. The findings contribute to a deeper understanding of the effectiveness of different recommendation approaches in the context of music discovery.

2. Introduction

2.1 Problem Statement

In this project, we aim to develop an effective song recommendation system by integrating various machine learning algorithms, including Popularity-Based, Singular Value Decomposition (SVD), Content-Based Models, and Collaborative Filtering. The challenge lies in providing users with personalized song recommendations based on their listening history and preferences, leveraging a subset of the 1 Million Song Dataset (Taste Profile Subset).

The central problem involves exploring the strengths and limitations of each recommendation strategy and addressing critical questions such as:
How can we optimize collaborative-based recommendation performance for a large-scale dataset?
Which collaborative-based algorithm is most suitable for production-sized challenges?
Which similarity measure will be the best to measure the similarity between songs?
What is the design of an effective content-based recommendation method for this specific dataset?

The project also aims to do a comparative study of all the recommendations as mentioned earlier algorithms using metrics such as accuracy, f1 score, and RMSE value and learn more about the challenges in implementing each algorithm. By addressing these challenges, we seek to enhance the user experience in music discovery and contribute valuable insights to the field of recommendation systems.

2.2 Purpose and Importance

Creating a better song recommendation system is important to make it easier for users to find music they like. When people have lots of songs to choose from, personalized suggestions are crucial for helping them discover new content efficiently.
This project focuses on improving recommendations for a large number of songs and finding the right algorithms to make the system work well in real-world situations. It also aims to accurately figure out which songs are similar and create effective ways to recommend music based on content. These efforts aim to give users a more personalized experience, making progress in recommendation systems and establishing standards for accuracy, variety, and scalability in music suggestions.

2.3 Background and Related Work

Listening to music has changed significantly because of digital platforms and streaming services like Spotify, YT Music, and Apple Music. With so many songs available in different genres, it's hard for users to find what they like. That's where good song recommendation systems come in handy. Our project is all about creating a smart system that understands what users like and makes playlists personalized just for them. We're using different machine-learning techniques to make the experience of discovering music better.

Many researchers have looked into making recommendation systems better, and their findings help guide our project. For example, Resnick and Varian [1] introduced collaborative filtering, a method that modern recommendation systems are built on, which uses a user-item matrix to recommend new songs to the users.

Koren et al. [2] suggested using matrix factorization techniques, like Singular Value Decomposition (SVD), to improve the accuracy of recommendations.

In simpler terms, McFee et al. [3] showed that we can use features like genre and artist to recommend music, and this is something we're incorporating into our models. Also, Deshpande and Karypis [4] found that simple and

efficient methods, like popularity-based recommendations, are equally important. We're using their ideas as a starting point for our project.

When dealing with a lot of data, Bell and Koren [5] suggested scalable collaborative filtering algorithms. This means making recommendations works well even when we have a ton of songs to consider. Their ideas are helping us make our collaborative-based recommendations work efficiently on the 1 Million Song Dataset.

So, our project builds on these ideas but also explores new ways to make song recommendations even better. We want to create a system that not only suggests songs accurately but also thinks about variety, how well it works with a lot of songs, and whether users are happy with the suggestions.

3. Methodology - Proposed Solution

3.1 Dataset

http://millionsongdataset.com/tasteprofile/
https://www.kaggle.com/competitions/msdchallenge/data

For this project, we decided to use the Taste Profile Subset of the 1 Million Song Dataset.

The Taste Profile Subset is a condensed version of the extensive 1 Million Song Dataset, tailored for the exploration and development of music recommendation systems. This subset retains a representative sample of user interactions and song features, making it a valuable resource for researchers and developers interested in building, testing, and optimizing recommendation algorithms. By focusing on a subset, we can streamline computational efforts, reduce processing time, and efficiently experiment with diverse recommendation strategies.

Following is a snippet of the Taste Profile Subset dataset:

userID	songID	play_count
b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOAKIMP12A8C130995	1
b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOAPDEY12A81C210A9	1
b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOBBMDR12A8C13253B	2
b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOBFNSP12AF72A0E22	1
b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOBFOVM12A58A7D494	1

3.2 EDA (Exploratory Data Analysis)

We performed the following analysis on the Taste Profile Subset dataset before using it for any recommendation algo:

3.2.1 Shape of the dataset:

- The shape of the taste profile subset is (10000,3) ie. It has 10000 rows and 3 columns.

3.2.2 Check for Missing values:

- The Taste Profile Subset has no missing values in any column.

3.2.3 Checking for unique rows and duplicates:

- The dataset has 10000 unique rows and no duplicates in any of the columns.

3.2.4 Finding the range of the 'play_count' feature:

- The play_counts in this dataset are in the range of 1-140.

3.2.5 Quartile Distribution of the 'play_count' feature:

- count 10000.000000
- mean 2.522300
- std 4.444736
- min 1.000000
- 25% 1.000000
- 50% 1.000000
- 75% 2.000000
- max 140.000000

$$Z = \frac{x - \mu}{\sigma}$$

Z = standard score

x = observed value

μ = mean of the sample

σ = standard deviation of the sample

- From the above stats, we can see that the mean number of times a song has been played in this dataset is 2.52 and the standard deviation of this distribution is 4.44.
- The 'play_count' variable has a skewed distribution from the above stats.

3.3 Feature Engineering

- The taste profile subset has only 3 columns: userID, songID, and play_count.
- There is no need for feature selection in this case as the purpose of using this dataset is just to experiment with different recommender systems.
- However, the play_counts column has a skewed distribution. This needs to be normalized before it can be used for training the model.
- We applied z-score normalization on the play_count column to achieve this.
- The advantage of using normalization is that now the mean of the distribution is 0 and the standard deviation is 1.

3.4 Popularity Based Recommendation

The analysis of the dataset containing 3.5 million records reveals an intriguing aspect known as the Power-Law phenomenon, which is prevalent in real-world systems. In the context of our study, we are particularly interested in understanding the user distribution when it comes to consuming music services. It's evident that a small fraction of users actively engage with a majority of the songs, while a large number of users access these services sparingly.

To efficiently summarize this massive dataset, we've devised a strategy to determine the number of active users we need to consider for constructing a meaningful recommender system. Instead of directly grouping the data using Pandas, which could be computationally intensive given the dataset size, we've adopted an iterative approach over the records. This method allows us to count the occurrences per user.

We've focused on the top 40% of songs based on play count and sought to ascertain the percentage of users contributing to this selection. By ranking users according to the number of songs they've listened to, we've isolated the top 100,000 users for analysis.

Our observations indicate an intriguing pattern: approximately 30% of the songs constitute nearly 80% of the total song plays, while less than 40% of users listen to nearly all the songs. This insight guided us to subset the songs and users' data, focusing on the top 30% of popular songs and the corresponding user base.

Having obtained approximately 1 million records pertaining to the top 30% of popular songs, we've efficiently managed memory usage by discarding unnecessary subset dataframes.

In the realm of recommendation systems, the popularity-based approach stands as a fundamental strategy. This method suggests content solely based on its overall popularity among users, disregarding individual user preferences or listening histories. Essentially, it recommends the most popular songs to users, assuming they haven't yet experienced them.

3.5 Recommendation using SVD (Singular Value Decomposition)

Singular Value Decomposition (SVD) is a matrix factorization technique. In recommendation systems, it's used to decompose the user-item interaction matrix into user factors, item factors, and singular values. SVD allows for dimensionality reduction, capturing latent features that influence user preferences. In the provided code, Surprise's SVD algorithm is trained and tuned using grid search to minimize RMSE. The resulting model predicts ratings for a test set, and its performance is evaluated by comparing actual vs. predicted ratings. SVD is popular for providing accurate and personalized recommendations.

Singular Value Decomposition (SVD) is used for collaborative filtering in the context of recommendation systems. Here's an explanation of how SVD is used in the code:

1. **Load Dataset into Surprise Format:**
    - The dataset is loaded into a Surprise Dataset format using the Reader and Dataset classes from the Surprise library. The data consists of user-song interactions with the play count as the rating.
2. **Split Data into Train and Test Sets:**
    - The dataset is split into training and testing sets using Surprise's train_test_split method. This is essential for evaluating the performance of the recommendation algorithm.
3. **Define Parameter Grid for SVD:**
    - A parameter grid is defined for tuning the hyperparameters of the SVD algorithm. The grid includes values for the number of latent factors (n_factors), the number of epochs (n_epochs), the learning rate (lr_all), and the regularization term (reg_all).

4. **Initialize SVD Algorithm:**
    - An instance of the SVD algorithm is initialized. SVD is a matrix factorization technique commonly used in collaborative filtering recommendation systems. It factors the user-item interaction matrix into three matrices: user factors, item factors, and singular values.
5. **Use GridSearchCV for Hyperparameter Tuning:**
    - GridSearchCV from Surprise is employed to perform a grid search over the defined parameter grid. The goal is to find the combination of hyperparameters that minimizes the Root Mean Squared Error (RMSE) on the validation set.
6. **Print Best RMSE Score and Parameters:**
    - The best RMSE score and corresponding hyperparameters are printed after the grid search.
7. **Create a Heatmap of RMSE for Hyperparameters:**
    - The results of the grid search are visualized using a heatmap. The heatmap displays the average RMSE values for different combinations of n_factors and n_epochs.
8. **Extract Actual and Predicted Ratings:**
    - The code extracts actual ratings from the test set and predicted ratings using the trained SVD model.
9. **Create Scatter Plot of Actual vs. Predicted Ratings:**
    - A scatter plot is created to visually compare the actual ratings with the predicted ratings on the test set. This plot helps in assessing the model's performance in predicting user preferences for songs.

In summary, the code uses SVD, a matrix factorization technique, for collaborative filtering to make personalized song recommendations. The hyperparameters of the SVD algorithm are tuned using grid search, and the performance is evaluated on a test set using RMSE. Visualization tools like heatmaps and scatter plots are used to analyze and communicate the results.

3.6 Recommendation using K Nearest Neighbours

K-Nearest Neighbors (KNN) is a straightforward and effective recommendation algorithm used in collaborative filtering. Operating on the principle that similar users or items exhibit similar preferences, KNN identifies neighbors based on past interactions. By considering a user's or item's k-nearest neighbors, it makes predictions or recommendations. Key strengths lie in its simplicity and ability to capture local patterns in user-item interactions, making it a widely utilized choice in recommendation systems.

The k-Nearest Neighbors (KNN) algorithm, specifically the KNNBasic variant, is used for collaborative filtering in the context of recommendation systems. Here's an explanation of KNN and its usage in the code:

k-Nearest Neighbors (KNN):

1. Algorithm Overview:

- KNN is a simple and intuitive algorithm used for collaborative filtering in recommendation systems.
- It operates based on the idea that users who have similar preferences in the past are likely to have similar preferences in the future.

2. Training the Model:

- The KNNBasic algorithm is initialized, and then it is trained on the provided training set (trainset).
- During training, the algorithm identifies patterns in user-item interactions to establish similarity between users or items.

3. Making Predictions:

- After training, the algorithm makes predictions on the test set (testset) by considering the k-nearest neighbors of each user or item.
- Predictions are generated based on the observed ratings from similar users or items.

4. Evaluating Performance:

- The Root Mean Squared Error (RMSE) is calculated on the test set to assess the accuracy of the predictions.
- Lower RMSE values indicate better performance, reflecting how well the algorithm's predictions align with the actual ratings.

5. Hyperparameter Tuning with GridSearchCV:

- Hyperparameters for KNNBasic, such as the number of neighbors (k), minimum neighbors for the prediction (min_k), and similarity options, are tuned using grid search.
- Grid search explores different combinations of hyperparameter values to find the set that minimizes the RMSE on a validation set.

6. Visualizing Hyperparameter Tuning:

- The results of the grid search are presented in a heatmap, providing an overview of the RMSE scores for different combinations of hyperparameters.
- This visualization aids in identifying the best hyperparameter values for optimal model performance.

7. Scatter Plot of Actual vs. Predicted Ratings:

- Actual and predicted ratings on the test set are extracted and plotted on a scatter plot.
- This plot helps visually assess how well the algorithm's predictions align with the true ratings.

In summary, the KNNBasic algorithm in the code leverages the k-nearest Neighbors approach to make collaborative filtering recommendations. The algorithm is trained on a training set, and tested on a separate set, and its hyperparameters are tuned for optimal performance using grid search. The RMSE metric is used to evaluate the accuracy of predictions, and results are visualized for analysis.

3.7 Content-Based Filtering

Content-based filtering is a recommendation approach that focuses on the characteristics or features of items to make personalized suggestions to users. In the context of the provided code, Content-Based Filtering is applied to recommend songs based on their textual content. The algorithm calculates the cosine similarity between songs using a matrix representation of their textual features. Specifically, it uses the linear_kernel function from the sklearn.metrics.pairwise module to create a cosine similarity matrix. The content_based_recommendation function is then designed to take a test song's index, the cosine similarity matrix, and the desired number of recommendations as inputs. By comparing the textual features of songs, the function ranks and returns a list of songs that are most similar to the chosen test song. This approach enables the system to offer personalized recommendations by considering the inherent characteristics of songs, such as their titles or artist names, making it particularly useful when dealing with explicit item features. The code further demonstrates the usage of Content-Based Filtering by providing a ranked list of recommended songs based on the textual content similarity to a selected test song.

Content-based filtering is implemented for recommending songs based on their textual content. Here's an explanation of Content-Based Filtering and its usage in the code:

Content-Based Filtering:

1. **Algorithm Overview:**

- **Idea:** Content-based filtering recommends items by analyzing their features and matching them with user preferences.
- **Approach:** In this code, a similarity matrix is computed using the cosine similarity measure, comparing the textual content of songs.

2. **Implementation Steps:**

- **Cosine Similarity Matrix:** A matrix of cosine similarity scores is computed based on the textual content of songs, where each row or column corresponds to a different song.
- **Function:** A function, content_based_recommendation, takes a song index, the cosine similarity matrix, and the number of recommendations as input. It returns a list of recommended songs based on their textual similarity to the input song.

3. **Recommendation Process:**

- **Test Song:** A test song is selected for which recommendations are to be generated. The index of this song is specified as test_song_index.
- **Recommendation:** The content-based recommendation function is then applied to find songs that are textually similar to the test song. The top recommendations are presented based on their cosine similarity scores.

4. **Output:**

- **Printed Recommendations:** The code prints the top recommended songs along with their details such as song title, artist name, and rank.

Usage in the Code:

1. **Cosine Similarity Matrix:**
    - The linear_kernel function from sklearn.metrics.pairwise is used to compute the cosine similarity matrix for the textual content of songs.
2. **Content-Based Recommendation Function:**
    - The content_based_recommendation function takes a song index, the cosine similarity matrix, and the number of recommendations as input.
    - It calculates similarity scores for the input song with all other songs, sorts them, and returns the indices of the most similar songs.
3. **Test Song Recommendation:**
    - A test song is selected, and the content-based recommendation function is applied to get a list of recommended songs based on textual similarity.
4. **Printed Recommendations:**
    - The top recommended songs, along with their details, are printed, providing users with a personalized list of songs based on the content similarity to the chosen test song.

In summary, Content-Based Filtering in this code leverages cosine similarity on textual content to recommend songs that are textually similar to a given test song. The recommendations are then presented to the user in a ranked list.

3.8 Collaborative Filtering

- Collaborative filtering is a method employed in recommendation systems to anticipate a user's preferences by analyzing the preferences and behaviors of a cluster of like-minded users. The fundamental concept is that users with analogous preferences historically are likely to share similar preferences in the future.

- There are two primary types of collaborative filtering:

  1. User-Based Collaborative Filtering: User-based collaborative filtering recommends items based on the preferences and actions of users who closely resemble the target user. The system identifies users with comparable tastes and suggests items favored by those akin users. This method assumes that users who concurred in the past are inclined to concur in the future.

  2. Item-Based Collaborative Filtering: Item-based collaborative filtering provides recommendations by pinpointing items akin to those the user has previously demonstrated interest in. The system seeks items with comparable user interaction patterns. This approach assumes that if a user appreciates a specific item, they are likely to enjoy items that share similarities with it. Item-based collaborative filtering is computationally efficient and particularly suitable for scenarios where the number of users significantly outweighs the number of items.

- For this project, we decided to experiment with the User-Based Collaborative Filtering Algorithm. This is because our dataset mainly consists of information about the users and the number of times they played a particular song.

- Following are the steps involved in creating a recommendation system using the user-based collaborative filtering approach:
  - Data Cleaning and preprocessing (same as above algorithms)
  - Feature Engineering and Data Transformation. (described above)
  - Creating a user-item matrix, where rows represent users, columns represent items, and the entries represent user preferences.
  - Calculate similarity between users based on their preferences. This helps identify users with similar preferences. The similarity measures used in this project are pearson correlation coefficient and cosine similarity.
  - Neighborhood Selection - Determine a subset of users (neighborhood) with the highest similarity to the target user. The size of the neighborhood can impact the system's performance and is often chosen through experimentation.
  - Predict the preferences of the target user for items they have not yet interacted with. This involves aggregating the preferences of users in the neighborhood, weighted by their similarity to the target user.
  - Generate a list of top-N recommendations for the target user. Rank items based on predicted preferences, and present the highest-ranked items as recommendations. The number of recommendations (N) can be configured based on user preferences or system requirements.
  - Evaluate the performance of the recommendation system using appropriate metrics such as RMSE. Split the data into training, validation and testing sets to assess how well the model generalizes to new data.
  - Fine-tune the model parameters, including the size of the neighborhood, similarity threshold, or weighting schemes. Experiment with different similarity metrics to optimize the performance of the recommendation system.
  - Evaluate on test data and generate recommendations for a user.

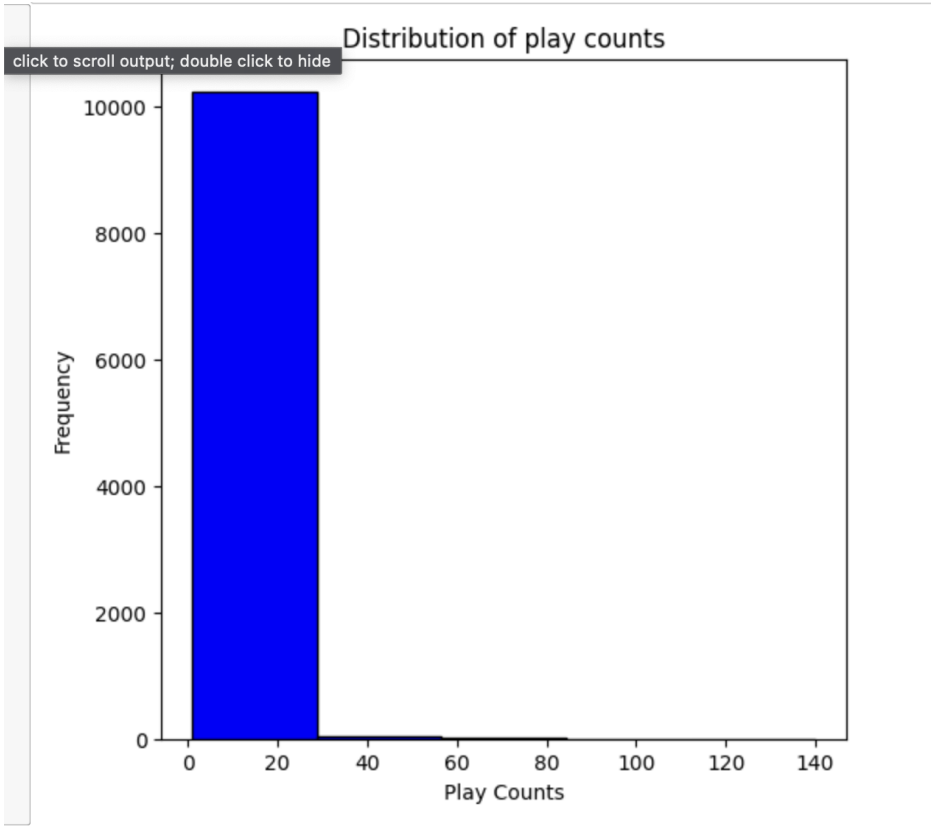
4. Brief explanation of the code.

- **Data Loading and Understanding:**
  - **Loading the Dataset**: The code loads the "train_triplets.txt" dataset, which contains information about user-song interactions (such as user ID, song ID, and play count). Additionally, it loads the "unique_tracks.txt" file, including details about tracks (track ID, song ID, artist name, and song title).
  - **Merging Dataframes**: It merges the 'songs' and 'tracks' data frames to create a unified dataset ('df_merged') that combines user-song interaction details with information about tracks, artists, and song titles.
- **Data Exploration and Visualization:**
  - **Understanding the Dataset**: The code inspects the shape of the merged data frame ('df_merged') to understand its size and structure.
  - **Checking for Duplicates**: It verifies for any duplicate rows within the dataset.
  - **Play Count Analysis**: Determines the range of play counts, offering insights into the distribution of user interactions with songs. Visualizes this distribution through a histogram.
  - **Top Songs and Artists Analysis**: Identifies and visualizes the top 10 most played songs and popular artists using bar plots.
- **Data Normalization:**
  - The StandardScaler function from sklearn.preprocessing was used to apply z-score normalization to play_count.
- **SVD Algorithm Implementation:**
  - **Surprise Library Usage**: Utilizes the Surprise library to create a recommendation system based on Singular Value Decomposition (SVD).
  - **Hyperparameter Tuning**: Employs GridSearchCV to optimize SVD's hyperparameters (n_factors, n_epochs, lr_all, reg_all) by systematically searching through various parameter combinations. Displays the results in a heatmap and generates a scatter plot illustrating actual versus predicted ratings on the test set.
- **KNN Algorithm Implementation:**
  - **KNNBasic Algorithm**: Implements the KNNBasic algorithm for recommendation purposes.
  - **Hyperparameter Tuning with GridSearchCV**: Conducts hyperparameter tuning for KNNBasic (k, min_k, sim_options) using GridSearchCV. Displays the results in an RMSE heatmap and generates a scatter plot illustrating the actual versus predicted ratings on the test set.
- **Report Explanation:**
  - The report aims to understand user-song interactions and provide insights into popular songs, artists, and user preferences.
  - Discusses the process of building recommendation systems using SVD and KNN algorithms from the Surprise library.
  - Utilizes visualizations (histograms, bar plots, RMSE heatmaps, scatter plots) to visually represent data distributions, parameter tuning results, and model performance on the test data, offering an in-depth analysis of user behaviors and algorithm performance.
- **Collaborative Filtering Implementation:**
  - The user-item matrices were implemented using the pandas.pivot functionality.
  - For the similarity measures, both cosine similarity and pearson correlation coefficient I used the respective functions from sklearn.metrics.pairwise.
  - The actual functions to generate recommendations were written using these similarity helper functions.

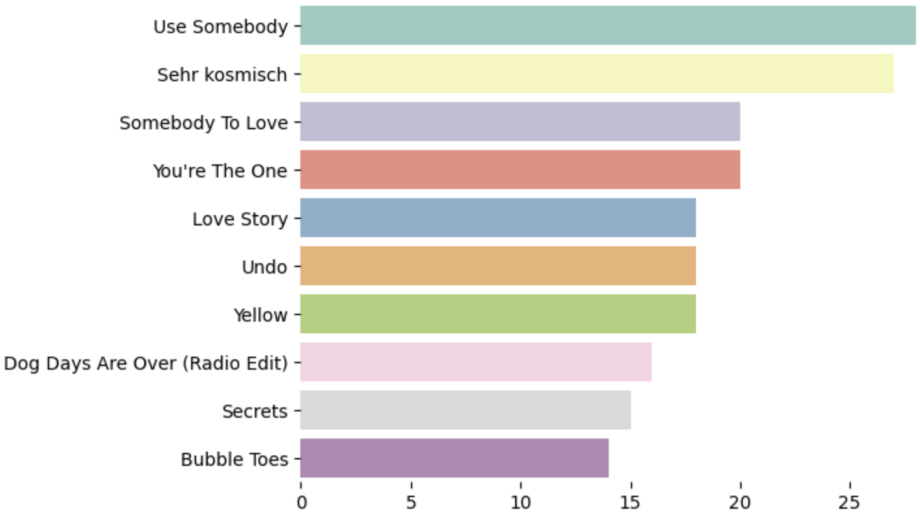
5. Results

- Exploratory Data Analysis

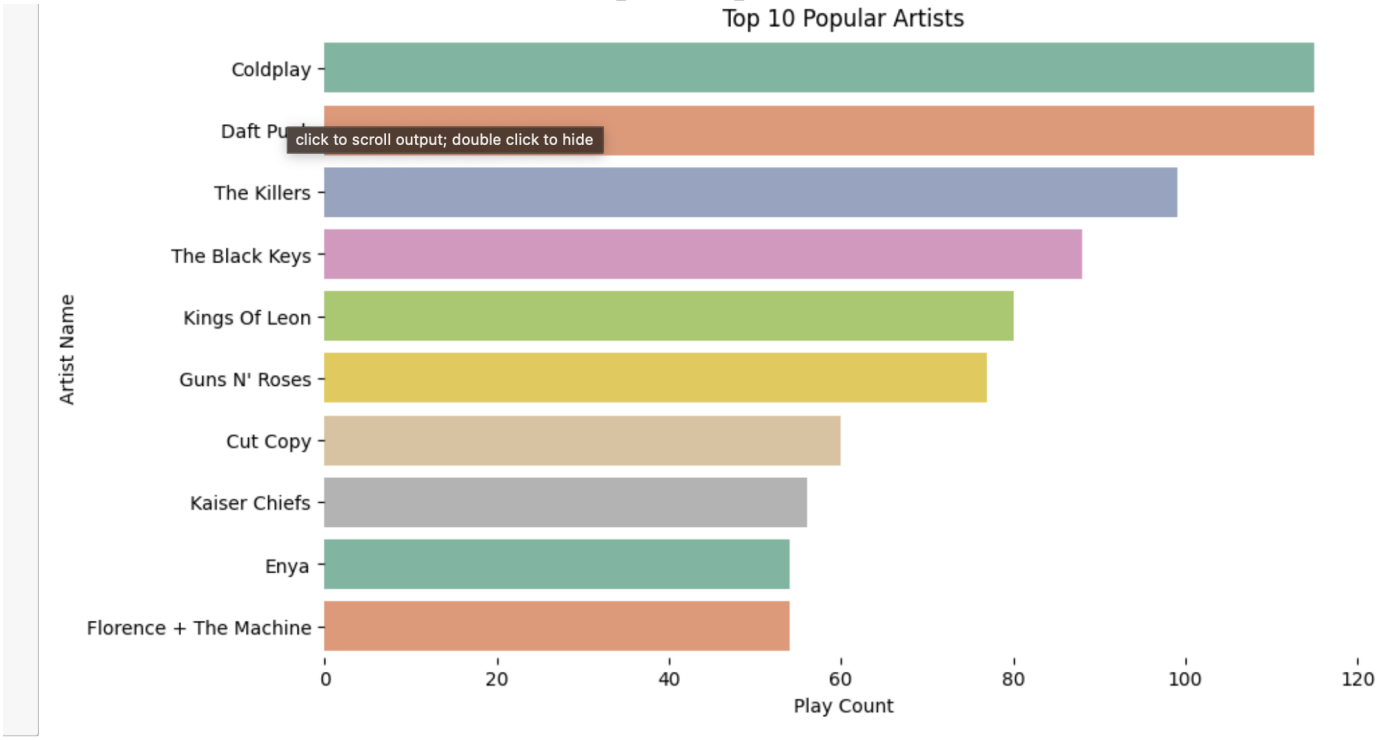
Visualizing the play_count column's distribution



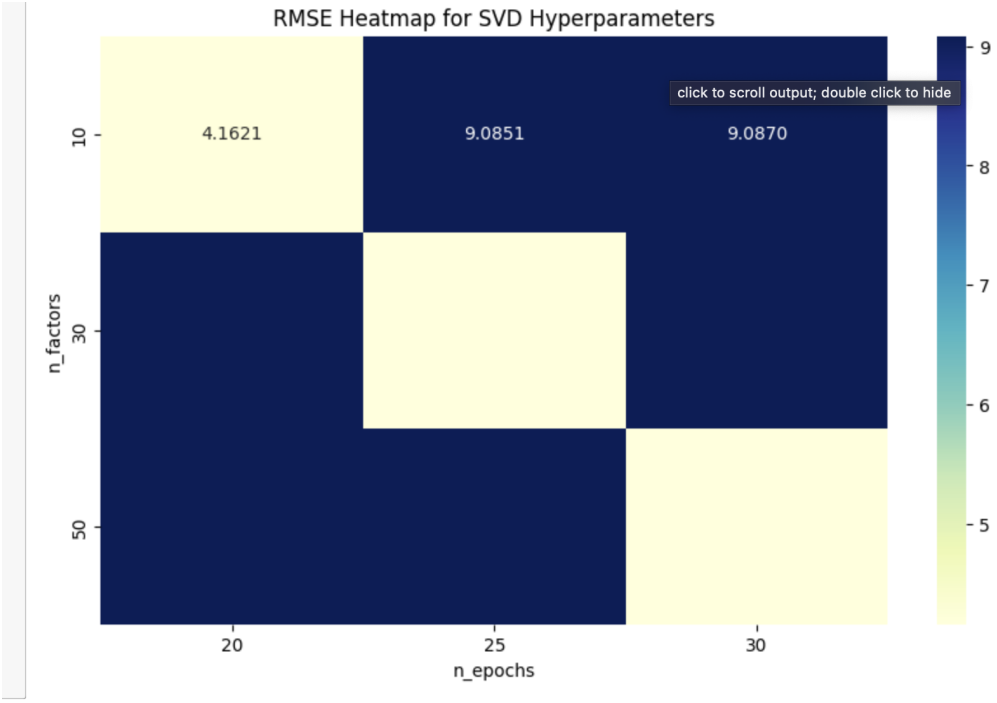
Top 10 songs based on popularity

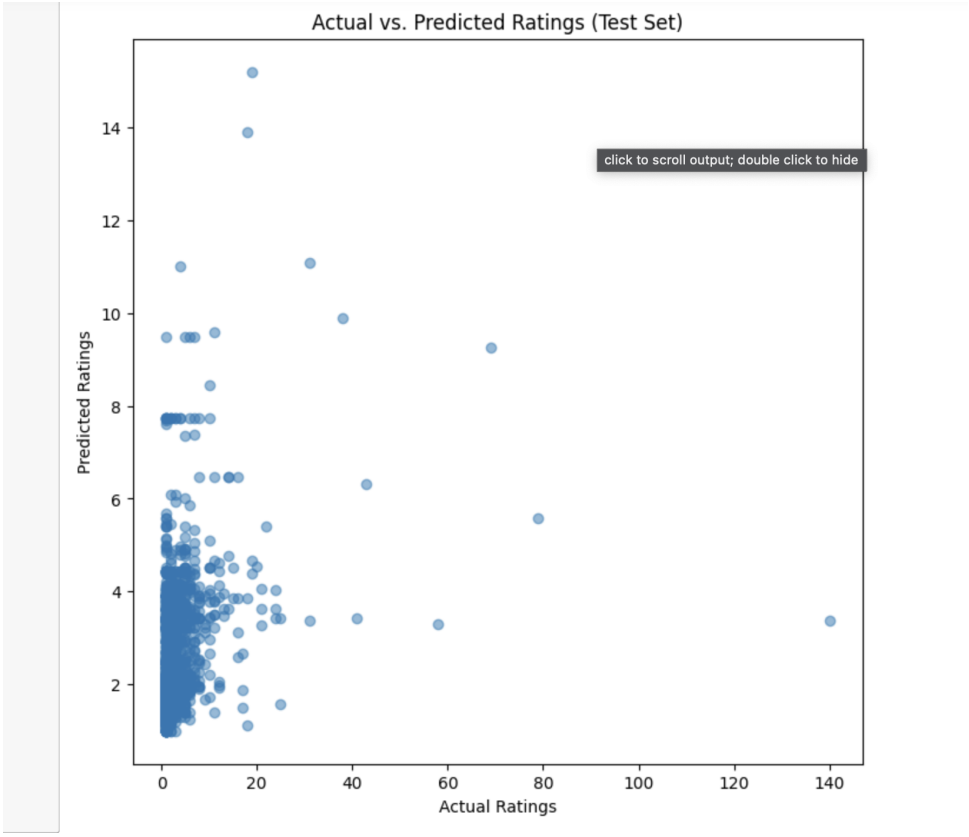


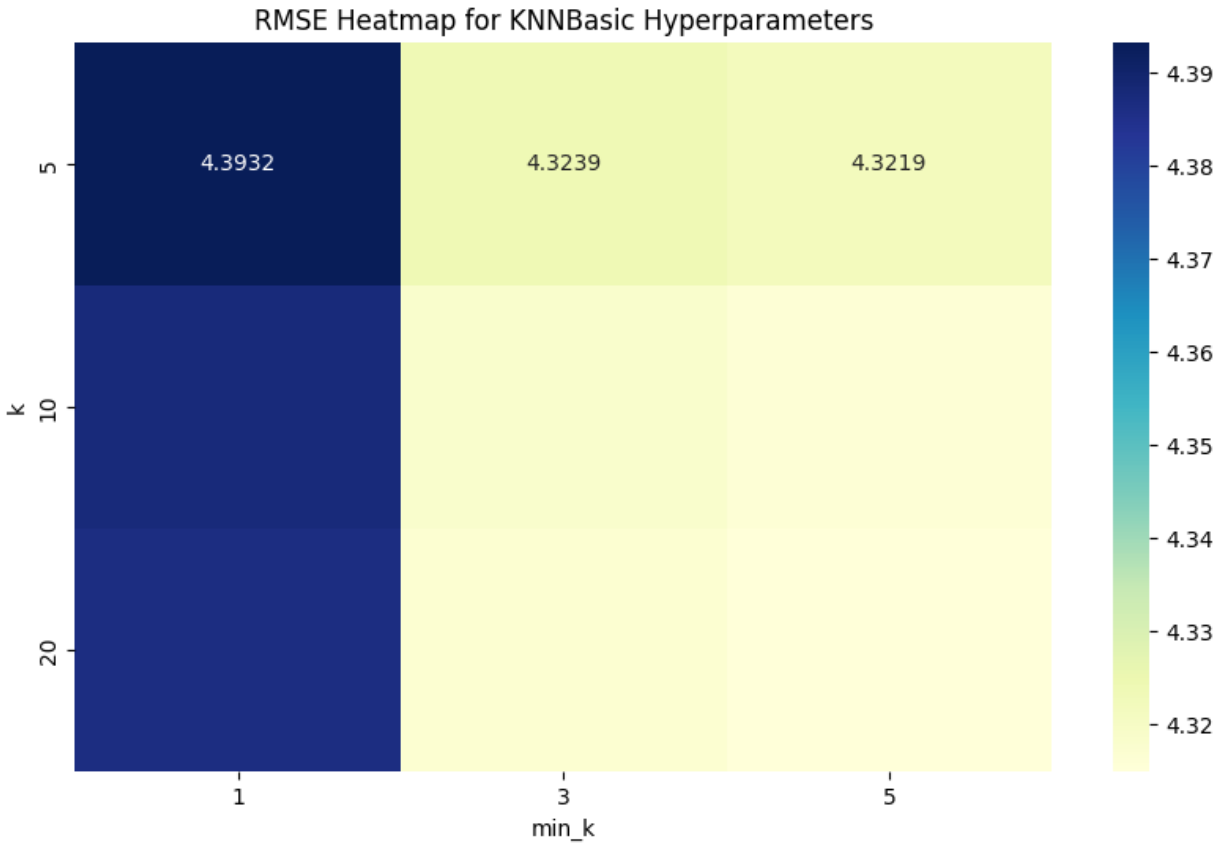
Top 10 Popular Artists

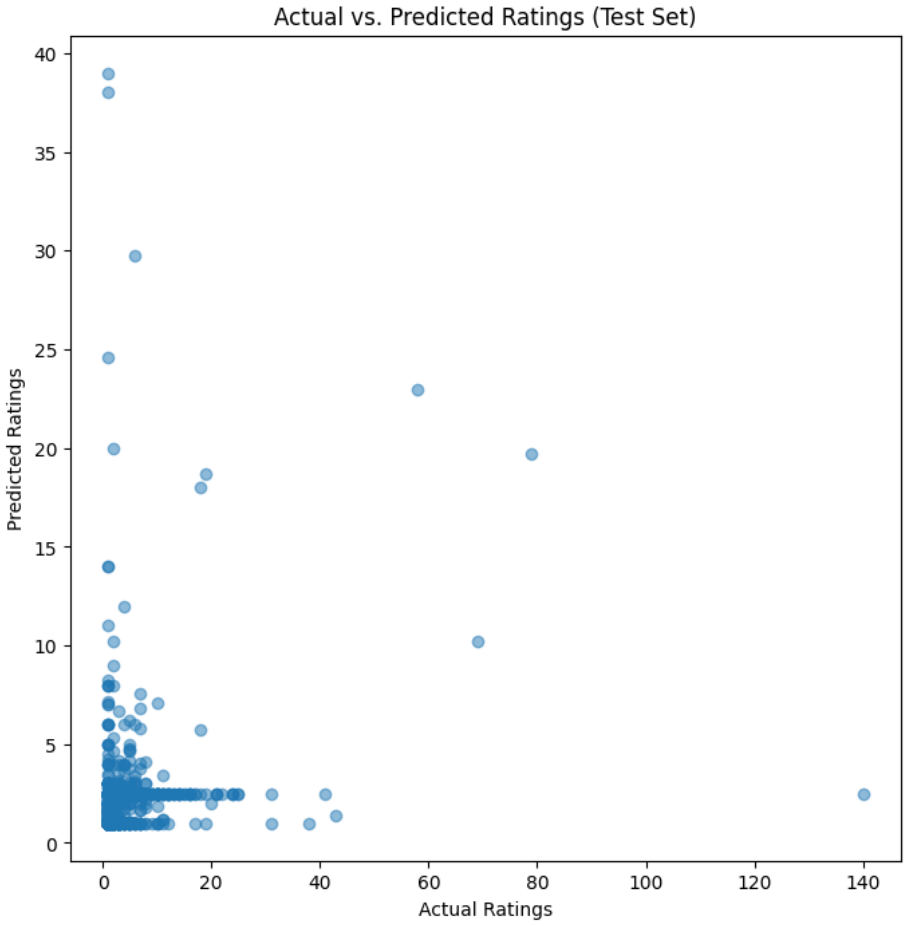


SVD Hyperparameter Heatmap




Actual vs. Predicted Ratings (Test Set)
click to scroll output; double click to hide
Predicted Ratings
Actual Ratings
0
20
40
60
80
100
120
140
2
4
6
8
10
12
14


RMSE Heatmap for KNNBasic Hyperparameters
4.3932
4.3239
4.3219
k
5
10
20
min_k
1
3
5
4.39
4.38
4.37
4.36
4.35
4.34
4.33
4.32



Following are some results for the user-based collaborative filtering approach:

```
RMSE (Cosine) - Train: 46.748448894575404 Validation: 3.1837090947988624
RMSE (Pearson) - Train: 0.28510219690908095 Validation: 0.9701738159034171


Testing different neighborhood values for the validation data
K value: 5, RMSE-Cosine: 3.1837090947988624, RMSE-Pearson: 0.9701738159034171
K value: 10, RMSE-Cosine: 3.1837090947988624, RMSE-Pearson: 0.9701738159034171
K value: 15, RMSE-Cosine: 3.1837090947988624, RMSE-Pearson: 0.9701738159034171
K value: 20, RMSE-Cosine: 3.1837090947988624, RMSE-Pearson: 0.9701738159034171

RMSE (Cosine) on Test: 4.337042410553731
RMSE (Pearson) on Test: 1.0934940037702052
```

Top Recommendations for User b80344d063b5ccb3212f76538f3d9e43d87dca9e (Cosine):

	song	predicted_rating
0	SOAAAGQ12A8C1420C8	33.837002
3158	SOPABZM12A6D4FC668	33.837002
3476	SOQPGDF12AB01858C5	33.837002
5091	SOYTZBN12AB0187A0C	33.837002
3136	SOOXDIJ12A6D4FDE33	33.837002
1693	SOIBCIC12A58A7B55B	33.837002
3009	SOOHWRZ12AC468BA59	33.837002
4539	SOVYNVS12AC3DF64AB	33.837002
4562	SOWBMHX12A8C13851C	33.837002
3545	SOQZYQH12A8AE468E5	33.837002

Top Recommendations for User b80344d063b5ccb3212f76538f3d9e43d87dca9e (Pearson):

	song	predicted_rating
3545	SOQZYQH12A8AE468E5	6.632379
2552	SOMEIDU12AB0182205	5.282400
1953	SOJIJWG12AAF3B46C0	2.807439
3915	SOSXLTC12AF72A7F54	2.194029
3158	SOPABZM12A6D4FC668	1.907453
494	SOCIHMS12A8C142CC7	1.682456
4467	SOVPBWS12A8C141EF1	1.682456
2565	SOMGIYR12AB0187973	1.503864
4839	SOXKGUD12A58A7C687	0.783873
5027	SOYIZSN12A6701E0BB	0.581311

6. Discussion and Conclusion

Collaborative Filtering Approach:

- As we discussed earlier, implementing a user based collaborative filtering recommendation system was more feasible in this case, since we had a good amount of data about which user played which song how many times.
- As we saw in the above results, the RMSE values using the Cosine similarity measure were quite higher than the ones obtained using the Pearson Correlation coefficient on training and validation data.
- In an attempt to reduce these RMSE values, we tried experimenting with different values of k i.e (neighborhood values) using both cosine and pearson similarity measures.
- It was observed that changing the k value had no effect whatsoever on the RMSE scores.
- Finally we observed that even on the test data, the RMSE values obtained using the Pearson similarity measure was less than the ones using the cosine similarity measure.
- Talking about the songs recommended for a particular user using both measures, as we can see in the above screenshot, cosine similarity measure recommends all the songs with the same high predictive rating.
- On the other hand the pearson correlation coefficient, recommends songs with less but varying predictive rating.
- This again is an indicator that pearson similarity measure is performing better than the cosine one.
- In regards to all the experiments conducted above with different similarity measures and neighborhood values, it is clear that Pearson correlation coefficient performs better than the cosine similarity measure when building a recommendation system.

KNN

- The dataset is trained with surprise module. The trained model is then tested on the dataset, and a RMSE value is obtained.
- Then the values are tuned to create a grid based approach to obtain an optimal RMSE value
- Heat Map and Scatter plots are generated for the trained model.

SVD

- Our methodology takes a step further with Singular Value Decomposition, an advanced matrix factorization technique. SVD excels in identifying hidden patterns within user-song interactions, providing a nuanced understanding that goes beyond surface-level preferences.
- Then the values are tuned to create a grid based approach to obtain an optimal RMSE value
- Heat Map and Scatter plots are generated for the trained model.

Popularity-Based Model.

This model is the simplest form of recommendation system. The idea is straightforward - recommend the items that are most popular to all users. For instance, in the context of our music dataset, the songs that have been played the most across all users would be recommended. While this model is easy to implement and understand, it lacks personalization as it doesn't consider individual user's preferences.

Content-Based Model.

This model recommends items by comparing the content of the items to a user profile. The content of each item is represented as a set of descriptors, such as the words in a document, the genre of a song, or the director of a movie. In the context of our music dataset, if a user frequently listens to a particular genre or artist, the system would recommend songs of that genre or by that artist. This model offers a more personalized recommendation, but it may lack diversity as it's based solely on the user's past behavior.

Both models have their strengths and weaknesses, and the choice between them depends on the specific requirements of the project at hand. In many real-world systems, these models are combined to leverage the strengths of both.

7. Future Work

The performance of a recommendation system depends on various aspects such as the data being used, feature engineering, the recommendation algorithm used, and finally how the system works when deployed in the test environment.
Following are some of the areas, which could be enhanced as part of this project:

1. Usage of Hybrid Models:
Content-based and Collaborative Filtering algorithms when used together are known to provide more accurate and personalized recommendations. For this project, we could explore the integration of such techniques build a hybrid model, and then compare its performance against the pre-existing ones.

2. Addressing Cold Start Problem:
The 'Cold Start Problem' in recommendation systems refers to the difficulty of making accurate suggestions for new users or items with limited or no historical interaction data. In such cases, it is difficult to make predictions on what this user might/might not like. Since traditional collaborative filtering methods rely heavily on historical preferences, we need to come up with solutions for this problem. Using external data sources or hybrid models along with content-based recommendations can help in such cases.
E.g: When you sign in to Netflix for the first time, they recommend all the trending content to you until you give the algorithm some preferences.

3. Enhanced Feature Engineering:
Real-world audio datasets are usually huge in size and contain a lot of features like (Mel Frequency Cepstral Coefficients) MFCCs or even advanced audio features like timbre, pitch, loudness, etc. While these features are very crucial in determining the characteristics of the song, they may/may not be useful when recommending a song to a particular user. Additionally, the algorithm will require a lot of computational space as well while dealing with so many features. In such cases using feature engineering techniques such as PCA or t-SNE might prove to be useful. Finding the correlation between the features before training the model on them is also a good start.

4. Making use of Implicit Feedback:
We can try to check if our recommendation system can make use of implicit feedback, such as user listening history or skip patterns. Many recommendation systems primarily use explicit feedback, but incorporating implicit feedback can provide additional insights into user preferences.

5. Evaluation Metrics:
Until now we have evaluated this system using metrics like accuracy, precision, recall, f1-score and RMSE(Root Mean Square Error). These metrics are a good start to judge how well our recommendation algorithm works. But going further we can evaluate our system using more advanced metrics like Novelty, Serendipity and Mean Reciprocal Rank(MRR).

6. Scalability and Efficiency:
We can optimize our system to scale and handle larger datasets efficiently. This could involve using parallel computing or even cloud-based solutions.

8. References.

1. Resnick, P., & Varian, H. R. (1997). Recommender systems. Communications of the ACM, 40(3), 56-58.
2. Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. Computer, 42(8), 30-37.
3. McFee, B., Bertin-Mahieux, T., Ellis, D. P., & Lanckriet, G. R. (2012). The million song dataset challenge. In Proceedings of the 21st ACM International Conference on Multimedia (pp. 909-912).
4. Deshpande, M., & Karypis, G. (2004). Item-based top-N recommendation algorithms. ACM Transactions on Information Systems (TOIS), 22(1), 143-177.
5. Bell, R. M., & Koren, Y. (2007). Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In Proceedings of the 7th IEEE International Conference on Data Mining (ICDM 2007) (pp. 43-52).
6. Xiaoyi Chen, Zhiran Chen, Kaicheng Ding, Weixin Liu, Xuening Wang, Ruitao Yi, Music to My Ear: Recommender System with Million Song Dataset, Carnegie Mellon University.

9. Annex A - Details of contributions from each team member.

- Team Tulip has the following 4 team members. Each one of us worked on a separate algorithm for building a recommendation system out of the 1 million song dataset.
    1) Krina Jitendrabhai Devani: Content-Based Filtering algorithm.
    2) Neha Joshi: Popularity-based recommendation algorithm.
    3) Vidhi Anand Thacker: Singular Value Decomposition(SVD) based recommendations.
    4) Anusha Kalbande: Collaborative Filtering (user-based) based recommendation algorithm.
- All the required EDA and other data preprocessing required for each of these algorithms was performed by the team member responsible for that algorithm.
- After all the experiments were conducted, all the team members worked together on compiling the report, and the presentation and recording the video of the presentation.