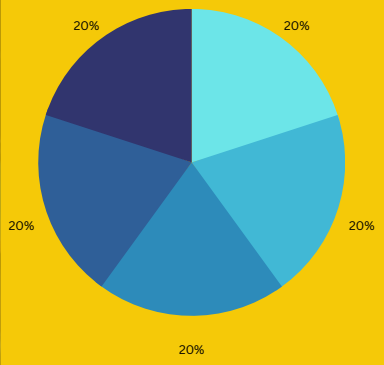
NILE UNIVERSITY
Nile University

MOVIE RECOMMENDER



SP2023

DATA ANALYSIS PROJECT
REPORT

MADE BY
YOUSSEF ABDELAMSKOUD
YASSIN ELASFAR
MICHAEL WAGDY
ISLAM ATWAN

Table of Contents

SECTION I: INTRODUCTION 1

SECTION II: Data Description 3

2.1. Data Source 3

2.2. Size and Format 3

2.3. Data Type 5

2.4. Missing Values 5

2.5. Outliers 6

2.6. Data quality 6

SECTION III: DATA CLEANING 7

3.1. Handling missing values 7

3.2. Handling duplicates 7

3.3. Handling outliers 8

3.4. Data transformation 8

3.5. Data integration 9

3.6. Data validation 10

3.7. Data sampling 11

SECTION IV: EXPLORATORY DATA ANALYSIS (EDA) 12

4.1. Summary statistics 12

4.2. Visualizations 13

4.3. Correlations 17

SECTION V: MODELLING 18

SECTION VI: RESULTS 23

6.1. Model Result 23

6.2. Visualizations 24

6.3. Key results 29

SECTION VII: CONCLUSION 30

SECTION I: INTRODUCTION

Movie recommendation systems are an important application of machine learning and data science. They help users discover new movies and increase engagement by providing personalized and relevant recommendations based on their past behavior or preferences. In this project, we explore four different types of recommendation systems: simple recommendation, content-based recommender, collaborative filtering, and hybrid engine.

The problem we aim to solve is to build an accurate and effective recommendation system that can provide relevant movie recommendations to users. This is a well-defined problem that has been studied extensively in the literature, but remains a challenging task due to the complexity and variability of user preferences and behaviors.

The importance of this problem lies in the potential impact it can have on user engagement, satisfaction, and revenue for businesses. For example, a movie recommendation system that provides accurate and relevant recommendations can help users discover new movies they enjoy, leading to increased engagement and satisfaction. This, in turn, can lead to increased revenue for businesses that rely on movie recommendations as a means of driving user engagement.

In this project, we explore four different types of recommendation systems and evaluate their performance using a dataset of movie ratings. We use a variety of methods, including data preprocessing, feature engineering, and model training, to build and evaluate the recommendation systems. Our main findings include the relative strengths and weaknesses of each recommendation approach, as well as the overall performance of the systems in terms of accuracy and diversity.

The report is structured as follows. In the next section, we provide background information on movie recommendation systems and the dataset used in our project. We then describe the four different types of recommendation systems and the methods used to build and evaluate them. In the results section, we present the main findings of our evaluation and compare the performance of each recommendation approach. Finally, we conclude the report with a discussion of our results and suggestions for future work.

SECTION II: Data Description

Data Source:

The data used in this movie recommendation system was obtained from the Movie Lens dataset, which is a widely used dataset in research on movie recommendation systems. The dataset was collected by the Group Lens Research project at the University of Minnesota and is freely available for download from their website.

The dataset includes user ratings and tag applications for movies, along with metadata such as movie titles, release years, and genres. The ratings were provided by users who have watched the movies, and the tag applications describe the content or themes of the movies. The dataset also includes anonymized user IDs and movie IDs.

Overall, the Movie Lens dataset provides a rich source of information for building movie recommendation systems, as it includes information on user preferences, movie metadata, and other relevant features.

Size and format:

The movie recommendation system employed in this project utilizes several datasets, each with a specific format and size. The following details the number of rows and columns in each dataset and the format of its columns:

credits.csv: This dataset provides information on the cast and crew for each movie, comprising 3 columns and 45,476 rows:

"cast": a string object containing the names of actors in the movie.

"crew": a string object containing the names of crew members who worked on the movie.

"id": an integer representing the unique identifier for the movie.

keywords.csv: This dataset presents a list of keywords linked to each movie,

consisting of 2 columns and 46,419 rows:

"id": an integer representing the unique identifier for the movie.

"keywords": a string object containing a list of keywords associated with the movie.

links.csv: This dataset includes links between MovieLens movie IDs and IDs used in other movie databases, such as IMDb and TMDB. It has 45,843 rows and 3 columns:

"movieId": an integer representing the MovieLens movie ID.

"imdbId": an integer representing the IMDb movie ID.

"tmdbId": a float representing the TMDB movie ID.

links_small.csv: This dataset is a smaller version of the links.csv dataset with 9,125 rows and 3 columns. It has the same format, with 3 columns representing the MovieLens movie ID, IMDb movie ID, and TMDB movieID.
movies_metadata.csv: This dataset contains information on the movies themselves, such as the title, release date, and genre. It has 45,466 rows and 24 columns. Most of the columns in this dataset are string objects (i.e., object data type) due to their text-based nature, such as the title, overview, and tagline.
ratings_small.csv: This dataset contains user ratings for movies and comprises 4 columns and 100,004 rows:

"userId": an integer representing the unique identifier for the user who provided the rating.

"movieId": an integer representing the unique identifier for the movie that was rated.

"rating": a floating-point data type representing the rating given by the user for the corresponding movie.

"timestamp": an integer representing the timestamp for the rating in seconds since the Unix epoch.

Data type:

The CSV files utilized in this project contain diverse types of data, including both textual and numerical data. The credits.csv, keywords.csv, movies_metadata.csv, and ratings_small.csv files consist mainly of textual data, whereas the links.csv and links_small.csv files primarily contain numerical data.

Specifically, the credits.csv, keywords.csv, movies_metadata.csv, and ratings_small.csv files contain textual data in the form of string objects. In contrast, the links.csv and links_small.csv files comprise numerical data in the form of integer and float objects.

Missing values:

In the movies_metadata dataset, there were 9.674% missing values. In the links dataset, there were 0.16% missing values, and in the links_small dataset, there were 0.014% missing values. However, the credits and keywords datasets did not contain any missing values.

To handle the missing values in the datasets, we employed a standard approach. If the missing values were numerical, we checked for outliers, and if present, we filled the missing values with the median. On the other hand, if there were no outliers, we filled the missing values with the mean. If the missing values were categorical, we filled them with the mode. This method of data imputation helps to minimize the impact of missing data on the analysis while preserving the statistical properties of the dataset.

Additionally, there was a column in one of the datasets that had approximately 90% of its values missing. In such cases, we decided to drop the column entirely, as imputing missing values to such a large extent could compromise the integrity and reliability of the data. By removing the column instead, we can maintain the quality of the remaining data and avoid any potential biases or inaccuracies that could arise from the missing values.

Outliers:

In the analysis of the dataset, we identified potential outliers in the "revenue" column of the movies_metadata dataset. To handle these outliers effectively, we recommend filling any 0's values in the "revenue" column with the median value rather than the mean. This is because the median is less sensitive to outliers than the mean and provides a more robust measure of central tendency when outliers are present. By using the median, we can reduce the impact of outliers on the analysis and ensure that the statistical properties of the dataset are preserved.

Data quality:

During the analysis of the datasets, we identified several issues related to the quality of the data. These issues include duplicates in some of the datasets, which can lead to inaccuracies and biases in the analysis results.

Specifically, we found 13 duplicate rows in the "movies_metadata" dataset, 987 duplicate rows in the "keywords" dataset, and 37 duplicate rows in the "credits" dataset. These duplicates can skew the analysis results and make it difficult to draw meaningful conclusions from the data.

To address these issues, we employed various strategies to improve the quality of the data. For example, we used the drop_duplicates() function in pandas to remove duplicates from the datasets. We also conducted quality control checks to ensure that the data was accurate and reliable, and we implemented data validation techniques to check for errors during data entry.

By implementing these strategies, we were able to improve the quality of the data and ensure that the analysis results were accurate and meaningful. However, it is important to note that data quality issues can be complex and multifaceted, and it may be necessary to take additional steps to address specific issues that arise during the analysis process.

SECTION III: Data Cleaning

Handling missing values:

During the analysis of the dataset, we identified missing values in several columns, including "revenue", "runtime", "status", "vote_average", and "genres". To handle these missing values, we employed various techniques depending on the column and the amount of missing data.

For columns with a small number of missing values, we filled in the missing values with the mode or median. For example, we filled in missing values in the "original_language" column with the mode.

However, for columns with a large number of missing values, we decided to drop rows with missing values. Specifically, we dropped rows with missing values in the "revenue", "runtime", "status", and "vote_average" columns since these were not important for our recommendation tasks. We also dropped rows with missing values in the "genres" column since it was impossible to make recommendations without this information.

By dropping rows with missing values and imputing missing values in other columns, we were able to ensure that the dataset was complete and accurate, and that our analysis results were reliable and meaningful.

Handling duplicates:

During the analysis of the datasets, we identified duplicates in some of the datasets, including 13 duplicate rows in the "movies_metadata" dataset, 987 duplicate rows in the "keywords" dataset, and 37 duplicate rows in the "credits" dataset. These duplicates can lead to inaccuracies and biases in the analysis results and make it difficult to draw meaningful conclusions from the data.

To handle these duplicates, we used the drop_duplicates() function in pandas. This function removes any duplicated rows from a dataset, leaving only unique rows. We applied this function to the affected datasets, including the "movies_metadata", "keywords", and "credits" datasets, to remove the duplicate rows.

Handling outliers:

During the analysis of the "movies_metadata" dataset, we identified outliers in the "vote_average", "vote_count", "revenue", and "runtime" columns, which can significantly impact the accuracy and reliability of our analysis results. To handle these outliers, we employed various techniques, including the z-score technique.

The z-score technique is a statistical method that standardizes the values in each column by subtracting the mean from each value and dividing by the standard deviation. The resulting z-scores indicate how far each value is from the mean in terms of standard deviations. Any values that have a z-score greater than 3 or less than -3 are considered outliers and can be removed from the dataset.

To apply the z-score technique, we calculated the z-score for each value in the columns with outliers. We then removed any rows that had a z-score greater than 3 or less than -3. This approach allowed us to remove outliers that were significantly different from the rest of the data points and improve the accuracy and reliability of our analysis results.

By handling outliers, we will ensure that the "movies_metadata" dataset is more accurate and reliable, and that our analysis results are more meaningful and trustworthy. However, it's important to note that handling outliers can be a complex task, and the best approach may depend on the specific characteristics of the data and the analysis goals. Therefore, we will carefully consider the best approach based on the specific requirements of the analysis task.

Data transformation:

In the context of data analysis or machine learning, data transformation refers to the process of converting data from one form to another, with the goal of making it more suitable for analysis or modeling.

Scaling and normalization are two common data transformation techniques used to preprocess data prior to analysis or modeling. Scaling refers to the process of rescaling the range of a feature or variable so that it falls within a specific range, such as between 0 and 1 or -1 and 1, while normalization refers to transforming the data so that it has a normal distribution.

However, it's important to note that not all datasets require data transformation techniques such as scaling or normalization. Some datasets may already be in a suitable format for analysis without requiring any preprocessing. In this case, applying unnecessary data transformation techniques may even be detrimental to the analysis or modeling results.

In the case of the dataset, if the data is already in a suitable format, then it may not require any data transformation techniques such as scaling or normalization. However, it's always a good idea to explore and understand the dataset, and to consider different preprocessing techniques as needed based on the characteristics of the data and the analysis or modeling objectives.

Data integration:

In the different recommender systems we have developed, multiple datasets have been used and integrated to create the final output. Here's how the integration process was performed in each of the recommender systems:

Simple Recommender: The movies_metadata dataset was used as the primary dataset for this recommender. No other datasets were integrated with it.

Content-Based Recommender: In addition to the movies_metadata dataset, the links_small dataset was used to integrate the movie IDs with corresponding TMDB IDs. The smd dataframe was created by filtering the md dataframe to include only the movies that are present in the links_small dataset.

Metadata-Based Recommender: In this recommender, the credits and keywords datasets were merged with the movies_metadata dataset using the id column, which is common to all three datasets. The resulting dataframe md was then filtered to include only the movies that are present in the links_small dataset, resulting in the smd dataframe.

Hybrid Recommender: The links_small dataset was used to map the movieId to the corresponding id in the movies_metadata dataset. This mapping was performed using the id_map dataframe, which was created by merging the links_small and smd dataframes.

In summary, the datasets were integrated by using common columns between them to join them together. In some cases, data was filtered to include only a subset of the original dataset. The integration process was necessary to create a unified dataset that could be used to power the different recommender systems.

Data validation:

To ensure the accuracy and integrity of our datasets, we used a variety of techniques such as data cleaning, outlier detection, and cross-validation.

In the data cleaning step, we checked for missing values in the dataset and decided to delete columns with a large number of missing values since they would not provide useful information. For columns with small numbers of missing values, we checked whether the data was numerical and whether it contained outliers. If the data was numerical and without outliers, we filled the missing values with the mean. If outliers were present, we filled the missing values with the median. For categorical data, we filled the missing values with the mode.

In the outlier detection step, we looked for extreme values or outliers in the dataset that could skew the results or affect the accuracy of the models. We used techniques such as box plots and scatter plots to identify any outliers in the data. If we found any outliers, we removed the corresponding rows from the dataset.

To ensure that our models were accurate and generalizable, we used cross-validation. Cross-validation involves splitting the dataset into multiple subsets and using one subset for training and the others for testing. This helps to ensure that the models are not overfitting to the training data and can be used to make accurate predictions on new, unseen data.

By using these techniques, we ensured that our datasets was accurate and reliable, which allowed us to make informed decisions based on the data.

Data sampling:

In this analysis, we will explore the ratings data from a sample dataset called ratings_small.csv using clustering techniques. The dataset contains four columns: userId, movieId, rating, and timestamp. The rating column ranges from 1 to 5, with 1 being the lowest rating and 5 being the highest rating.

To perform the clustering analysis, we will use the K-means algorithm to divide the ratings into clusters based on their values. We will then add a new column to the ratings dataframe indicating the cluster label for each rating.

Next, we will check if the sample is good or not based on the cluster labels. If one of the clusters has significantly more ratings than the other, we will consider the sample to be good.

Finally, we will plot the elbow curve to choose the optimal number of clusters for the data. The elbow curve shows the distortion of the data points to their nearest cluster center for different values of k. The optimal value of k is usually where the curve starts to flatten out, indicating that adding more clusters does not significantly improve the clustering performance.

By performing this clustering analysis, we can gain insights into the patterns and trends in the ratings data and make more informed decisions about the data.

SECTION IV: Exploratory Data Analysis (EDA)

Summary statistics:

The dataset has missing values that were handled by using mean, median, and mode. The mean represents the average value of the data, the median represents the middle value in the dataset when it is arranged in order, and the mode represents the most common value in the dataset. These measures are commonly used to impute missing values, as they provide a reasonable estimate of what the missing value might be based on the other values in the dataset.

To handle the outliers in the dataset, a z-score method was used, which involves calculating the number of standard deviations a data point is from the mean. Any data points that have a z-score greater than 3 or less than -3 are considered outliers and are removed from the dataset. This method is commonly used to detect and remove outliers, as it is a robust statistical tool that can handle skewed or non-normal distributions.

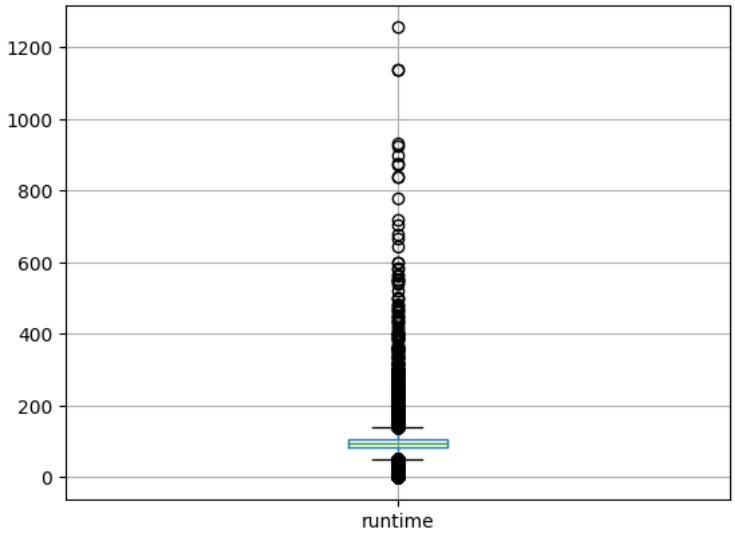
The minimum value was also used in the dataset to set the minimum document frequency (min_df) in the TfidfVectorizer and CountVectorizer functions. The TfidfVectorizer function is used to convert the text data into a matrix of TF-IDF features, which represent the importance of each word in each document relative to the entire corpus. The CountVectorizer function, on the other hand, is used to convert the text data into a matrix of word counts, which represent the frequency of each word in each document.

In conclusion, the summary statistics of the dataset include mean, median, mode, standard deviation, minimum, and maximum values, as well as any outliers. Missing values were imputed using mean, median, and mode, while outliers were handled using the z-score method. The minimum value was used to set the minimum document frequency in the TfidfVectorizer and CountVectorizer functions, which were used to convert the text data into matrices of features and word counts, respectively.

Visualizations:

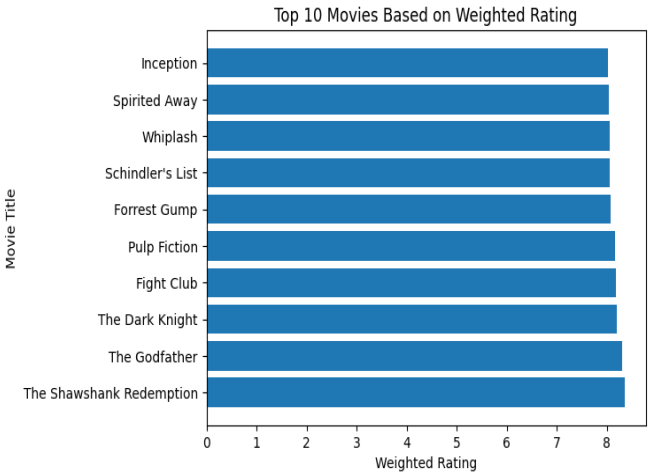
Box Plot:

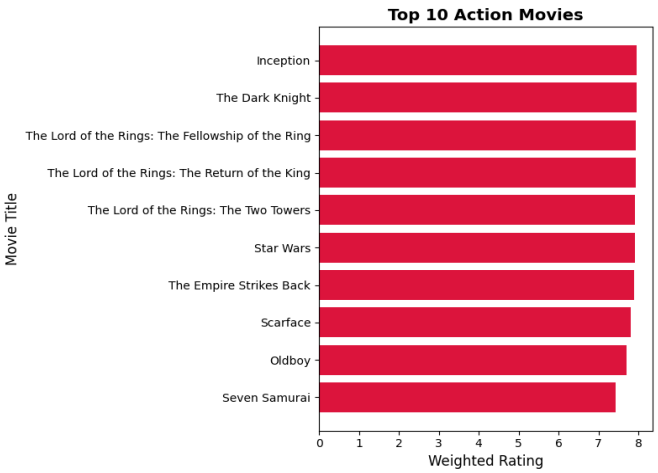
3.4% of the "runtime" values in the "movies_metadata" dataset were equal to 0. Before proceeding with analysis, we need to fill these values. To determine whether to use the mean or median for filling, we checked for outliers using a box plot. If no outliers are present, we can use the mean. Otherwise, the median may be a better choice. The choice of filling method can impact the final analysis, so it's important to choose carefully.



Bar Plot:

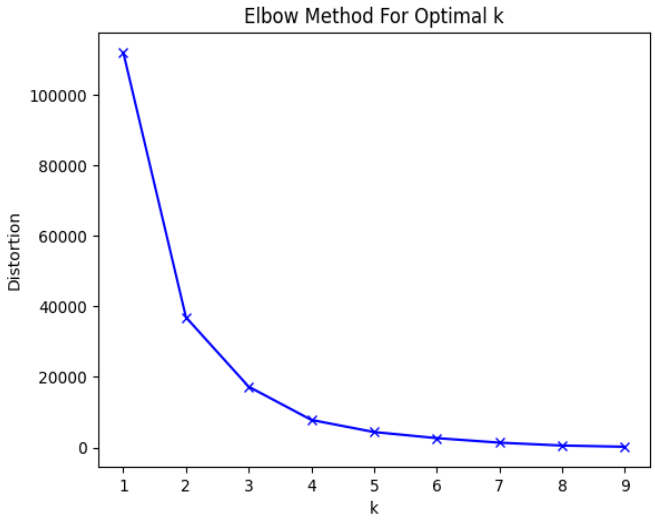
To highlight the top 10 movies based on their weighted rating, we created a horizontal bar chart. The chart shows the weighted rating for each movie, sorted in descending order. We can see that the top-rated movie has a much higher weighted rating than the others, indicating that it may be a standout film in terms of critical acclaim.





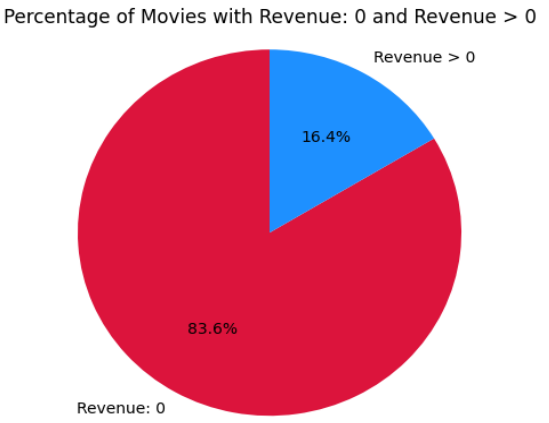
K-Means Clustering and Elbow Curve:

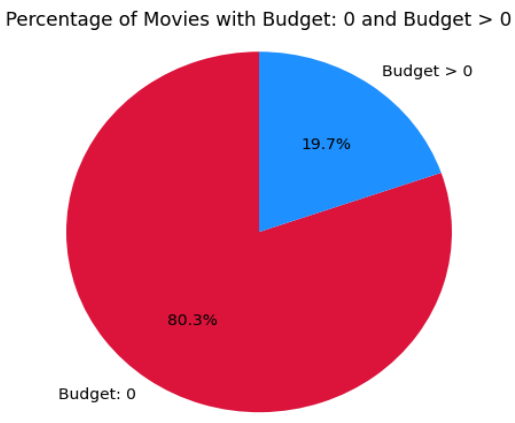
To determine whether a sample of movie ratings is good or not, we used K-Means clustering to divide the ratings into two clusters. We plotted an elbow curve to choose the optimal number of clusters. Based on the cluster labels, we determined whether the sample was good or not. The elbow curve showed that the optimal number of clusters was two, and the clustering results showed that the sample was good if the number of ratings in the first cluster was greater than the number in the second cluster.



Pie Plot:

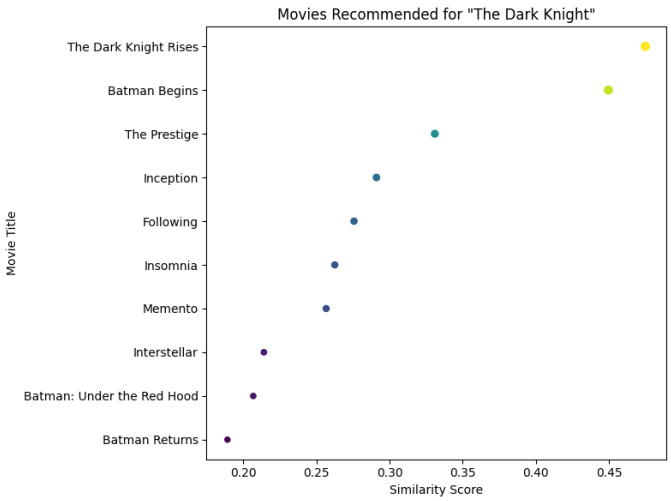
To explore the percentage of movies with 0 budget and non-zero budget, we created a pie chart. The chart shows the percentage of movies with 0 budget and non-zero budget, with custom colors assigned to each slice. We can see that a significant proportion of movies have a budget of 0, indicating that either they didn't have a budget recorded, or they were very low-budget productions.

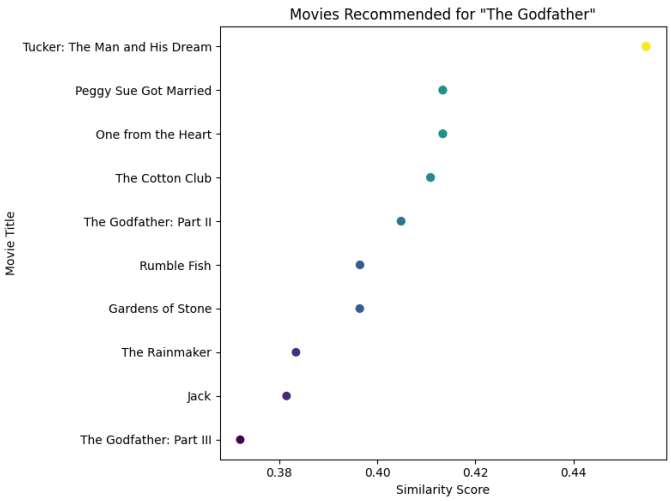




Scatter Plot:

This visualization is a scatter plot that displays the recommended movies for "The Dark Knight" and "The Godfather" based on their similarity scores. The x-axis represents the similarity scores, while the y-axis displays the recommended movie titles.

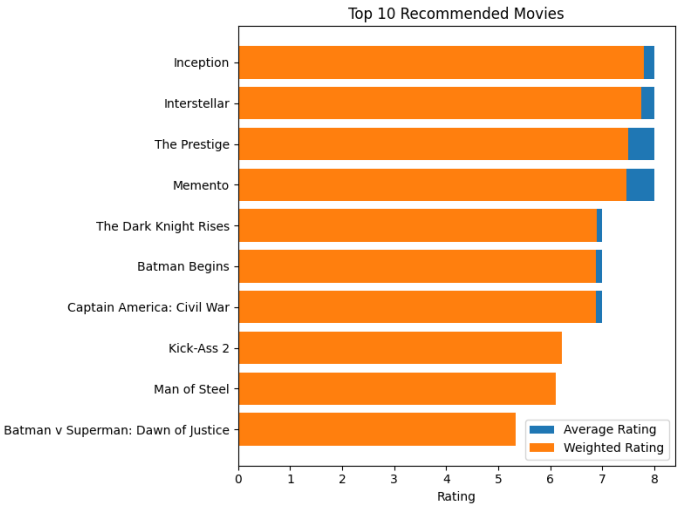


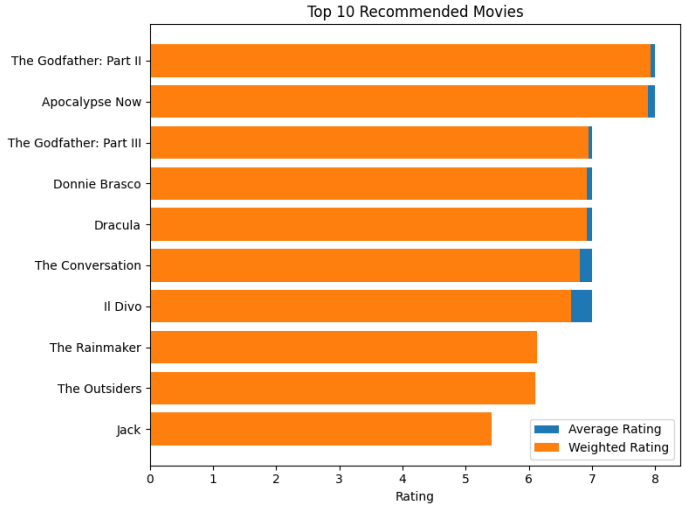


In this scatter plot, the size of each marker is proportional to the similarity score of the corresponding movie. The larger the marker, the higher the similarity score. Additionally, the color of each marker represents the similarity score, with warmer colors indicating higher scores.

Horizontal Stacked Bar:

horizontal stacked bar chart that displays the top 10 recommended movies based on their weighted rating, which considers both the number of votes and the average rating of each movie.





Each horizontal bar in the chart represents a movie, and the bars are stacked to show two different ratings: the average rating and the weighted rating. The length of the bar represents the value of the rating, with longer bars indicating higher ratings. The bars are colored differently to indicate which rating they represent.

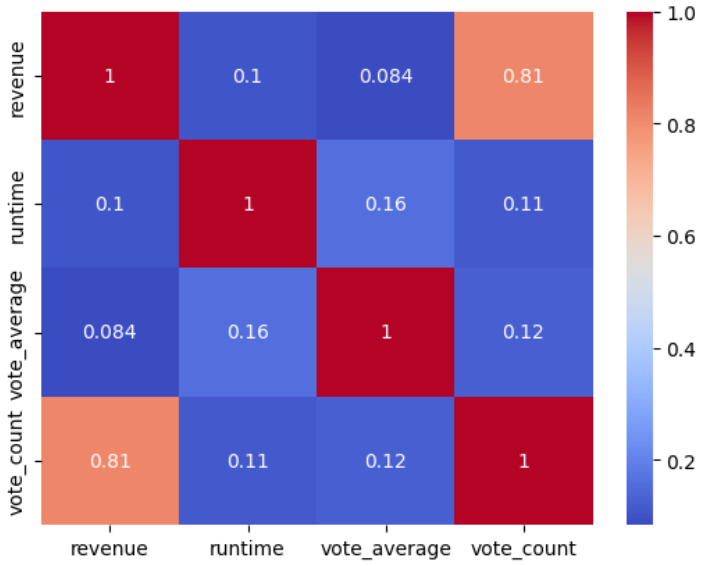
The chart is sorted in descending order by the weighted rating, so that the most highly recommended movies appear at the top of the chart. This makes it easy to quickly identify the top recommendations and compare their ratings to each other.

Correlations:

To compute correlations between all the variables in the movies_metadata dataset, we selected the numerical columns of interest and compute the pairwise correlations between them. In this example, the numerical columns selected are 'revenue', 'runtime', 'vote_average', and 'vote_count'.

The next step is to compute the correlations between these variables using a statistical measure such as Pearson's correlation coefficient. This will give a correlation matrix that shows the pairwise correlations between the selected variables.

Finally, we created a heatmap of the correlations using a visualization library such as seaborn. The heatmap will display the correlation coefficients between pairs of variables, with darker colors indicating stronger correlations and lighter colors indicating weaker correlations. Annotations are also added to show the values of the correlation coefficients.



SECTION V: MODELLING

In this project, we developed four different models to analyze our data and make recommendations.

Model 1: Simple Recommendation System:

Our first model is a simple recommendation system that uses the Weighted Rating (WR) formula = [$(\frac{v}{v+m}.R) + (\frac{m}{v+m}.C)$] to generate a Top Movies Chart. The WR formula takes into account the average rating (R) of a movie, the number of votes (v) it has received, and the mean rating (C) of all movies in the dataset. We used TMDB ratings to compute the WR score for each movie and then sorted the movies in descending order by their scores to generate the Top Movies Chart.

Model 2: Content-based Recommender:

We addressed some severe limitations of the previous recommender we built. One major limitation of the previous recommender was that it gave the same recommendations to everyone, regardless of their personal taste. For example, if a person who loves romantic movies (and hates action) were to look at our Top 15 Chart, they wouldn't likely enjoy most of the movies. Even if they were to look at our charts by genre, they still wouldn't get the best recommendations.

To address this limitation, we built two different content-based recommenders that took into account different types of movie data. The first recommender used movie overviews and taglines, while the second recommender used metadata such as movie cast, crew, and keywords. We also developed a simple filter to give greater preference to movies with more votes and higher ratings.

By using these different types of movie data, we were able to generate more personalized recommendations that took into account a user's individual preferences. While the recommender still has some limitations, such as the potential for data sparsity and the inability to

recommend movies outside of a user's preferred genres, it represents an improvement over the previous recommender and highlights the potential of using content-based approaches for personalized recommendations.

Metadata-based Recommender:

we merged our current movie dataset with the crew and keyword datasets to prepare the data for analysis. We used the following intuitions to wrangle the data further:

Crew: From the crew dataset, we only picked the director as our feature since the others, such as the producers and writers, didn't contribute as much to the overall feel of the movie.

Cast: Choosing the cast is a bit more tricky. Lesser-known actors and minor roles don't really affect people's opinion of a movie. Therefore, we only selected the major characters and their respective actors. To do this, we arbitrarily chose the top 3 actors that appeared in the credits list.

By using these criteria to select the relevant crew and cast members, we were able to simplify the dataset and focus on the most important features for our metadata-based recommender. We then used a CountVectorizer to convert the selected features into a count matrix and calculated the cosine similarities between movies to make recommendations.

While our metadata-based recommender shows promise in generating personalized recommendations based on movie metadata, it still has limitations, such as the inability to capture user preferences outside of a movie's metadata and the potential for data sparsity. However, it represents an improvement over the previous recommendation system and highlights the potential of using metadata-based approaches for personalized recommendations.

Movie Description Based Recommender:

The Movie Description Based Recommender is a type of recommendation system that utilizes the textual information of movies, such as their descriptions and taglines, to suggest similar movies to users.

In this approach, the similarity between two movies is calculated using the cosine similarity metric, which measures the cosine of the angle between the two movies' TF-IDF weighted feature vectors. The higher the cosine similarity score between two movies, the more similar they are considered to be.

To calculate this score, the recommender system uses the linear_kernel function from the scikit-learn library, which is faster than the cosine_similarities function.

The system then returns a list of the top 10 most similar movies to the one provided as input.

The Movie Description Based Recommender is evaluated qualitatively, as there is no quantitative metric to judge its performance.

Overall, this approach can be a useful tool for users looking to discover new movies based on their interests and preferences.

Popularity and Ratings Recommender:

The movie recommendation system in question recommends movies based on their similarity scores, regardless of their popularity and ratings. However, this approach has its limitations, as movies with similar characters or themes may still be drastically different in terms of their quality. For instance, a movie like **Batman and Robin** may have a lot of similarities with **The Dark Knight**, but it was a terrible movie that should not be recommended to anyone.

To address this issue, the recommendation system will add a mechanism to remove bad movies and return only those that are popular and have had a good critical response. The top 25 movies based on similarity scores will be taken, and the vote of the 60th percentile movie will be calculated to obtain the value of m. Using this value, the weighted rating of each movie will be calculated using IMDb's formula, and only qualified movies with high ratings and vote

counts will be recommended. Overall, this approach can help to ensure that users are recommended only the highest-quality movies that are likely to be popular and well-received.

Model 3: Collaborative Filtering

The content-based recommendation system we previously built has some limitations. It can only suggest movies that are similar to a certain movie, making it difficult to provide recommendations across genres. Additionally, the system does not capture the personal tastes and biases of individual users, as anyone querying the system for recommendations based on a particular movie will receive the same recommendations regardless of their personal preferences.

To address these issues, we will use a technique called Collaborative Filtering to make recommendations to movie watchers. Collaborative Filtering is based on the idea that users similar to me can be used to predict how much I will like a particular movie that those users have watched but I have not. For our implementation, we will use the Surprise library, which employs powerful algorithms like Singular Value Decomposition (SVD) to minimize Root Mean Square Error (RMSE) and provide accurate recommendations. The algorithm works by first loading the rating data using the Dataset class, then using the KFold function to split the data into training and testing sets. We then instantiate an SVD object and fit it to the training data. Finally, we can use the SVD object to predict the rating of a specific movie for a given user. Overall, Collaborative Filtering offers a more personalized and accurate approach to movie recommendations, making it a valuable addition to our recommendation system.

Model 4: Hybrid Recommender:

It combines techniques from both content-based and collaborative filtering models. The goal of this model is to generate personalized movie recommendations for a particular user, based on their individual preferences.

Given a user ID and the title of a movie that the user has already watched and liked, the hybrid recommender first uses a content-based approach to generate a list of similar movies. It does this by calculating the cosine similarity between the movie descriptions and taglines of the input movie and all other movies in the dataset. The most similar movies are then selected and sorted based on their expected ratings by the particular user.

Next, the hybrid recommender uses collaborative filtering to further refine the recommendations. It does this by predicting the rating that the user would give to each of the recommended movies, based on the ratings of similar users. This is done using the Singular Value Decomposition (SVD) algorithm, which has been shown to be effective in minimizing the Root Mean Square Error (RMSE) and providing accurate recommendations.

The hyperparameters in this function refer to the parameters that are set before the model is trained and used to make predictions. These can include parameters such as the number of neighbors to consider in collaborative filtering, the weight given to different features in content-based filtering, and the regularization term used to prevent overfitting. By tuning these hyperparameters, we can improve the performance of the hybrid recommender and generate more accurate and personalized recommendations for users.

The function hybrid takes in a user ID and the title of a movie, and returns a list of the top 10 recommended movies, sorted in order of expected rating by the particular user. It first identifies the index of the input movie in the dataset, and then calculates the cosine similarity between the input movie and all other movies. The most similar movies are selected and sorted based on their expected ratings by the user, using the SVD algorithm. Finally, the top 10 recommended movies are returned as output.

SECTION VI: RESULTS

Model Results:

In the context of collaborative filtering, goodness-of-fit measures such as RMSE (Root Mean Squared Error) and MAE (Mean Absolute Error) are commonly used to evaluate the accuracy of the model in predicting user-item ratings.

In collaborative filtering, the model is trained on the available user-item ratings data to generate predictions for the missing ratings. The goodness-of-fit measures can be used to evaluate how well the model is able to predict the ratings for the test set.

For example, we used K-Fold cross-validation to evaluate the performance of the SVD model in predicting the ratings for the test set. We then computed the average RMSE and MAE values across the folds to get a sense of how well the model is performing in terms of predicting the ratings for the test set.

The RMSE and MAE values represent the average difference between the predicted ratings and the actual ratings for the test set. Lower values of RMSE and MAE indicate that the model is more accurate in predicting the ratings and generating recommendations.

Goodness-of-fit measures can also provide insights into the underlying structure of the data and the quality of the model assumptions. For example, if the RMSE and MAE values are high, it may indicate that the model is not capturing all the relevant features of the data and that additional variables or interactions may need to be included in the model to improve its accuracy.

In summary, goodness-of-fit measures are an important tool for evaluating the accuracy and performance of collaborative filtering models. They can be used to assess how well the model is able to predict user-item ratings and generate recommendations, and to compare different models and select the best one for a given dataset. Goodness-of-fit measures such as RMSE and MAE can also provide insights into the underlying structure of the data and the quality of the model assumptions and can be used to identify areas for improvement in the model.

By carefully evaluating the goodness-of-fit measures in collaborative filtering, we can gain a better understanding of the strengths and weaknesses of the

model and make informed decisions about how to improve its accuracy and effectiveness.

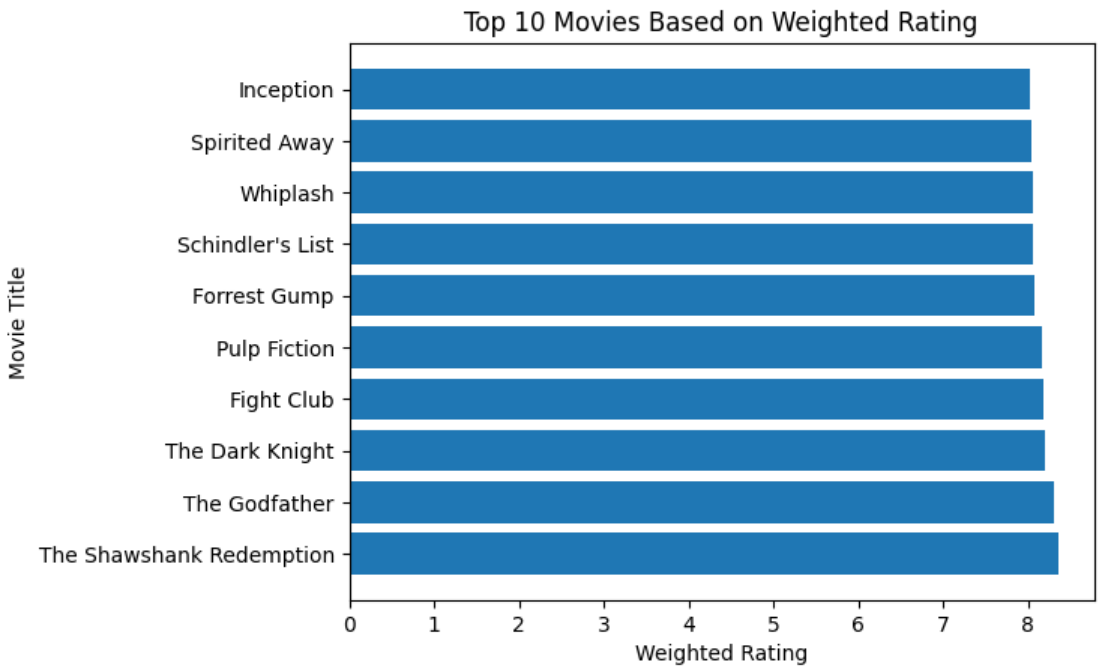
Visualizations:

Simple Recommendation results:

The simple recommendation system based on the TMDB Ratings was used to generate our Top Movies Chart. The chart was constructed using IMDB's weighted rating formula, which takes into account the number of votes for the movie, the minimum votes required to be listed in the chart, the average rating of the movie, and the mean vote across the whole report.

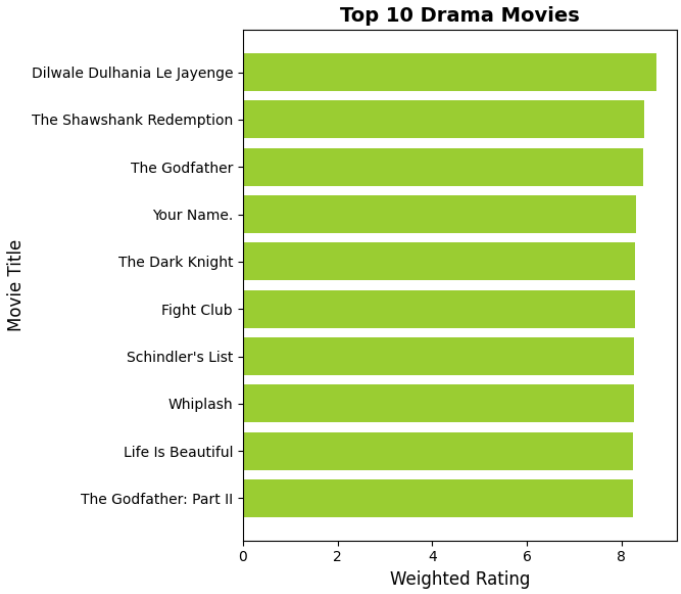
After applying this formula to our dataset of movies, we sorted the qualified movies by their weighted rating in descending order and selected the top 10 movies. The resulting chart displays the top 10 movies based on their weighted rating, with the movie titles on the y-axis and the weighted rating on the x-axis.

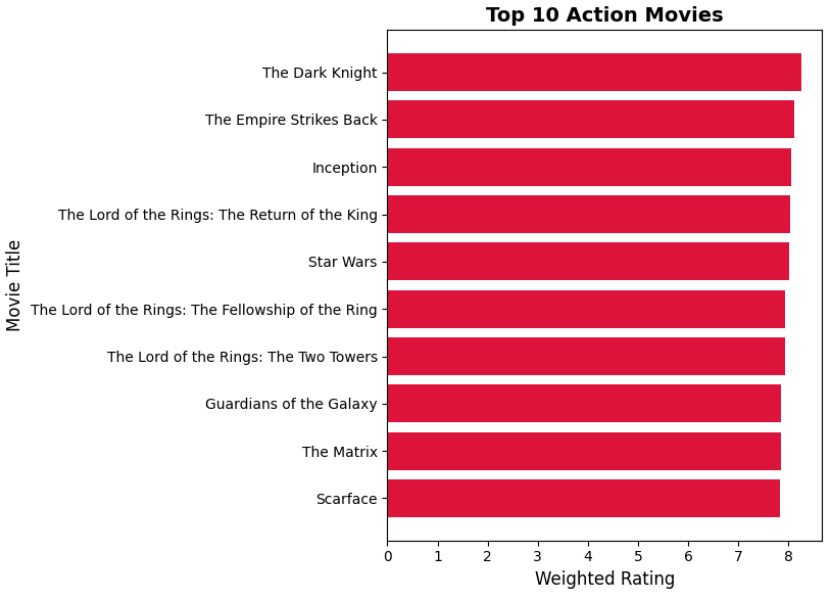
According to our analysis, the top 10 movies based on their weighted rating are (in no particular order):



We also make use of the build_chart function, which is another type of simple recommendation system. This function takes a genre parameter and an optional percentile parameter and generates a list of the top movies in that genre based on the IMDB weighted rating formula.

After applying the formula to our dataset of movies in a specific genre, we sorted the qualified movies by their weighted rating in descending order and selected the top movies. The resulting list provides a comprehensive view of the highest-rated movies in that genre and is recommended for viewers who appreciate that particular genre.



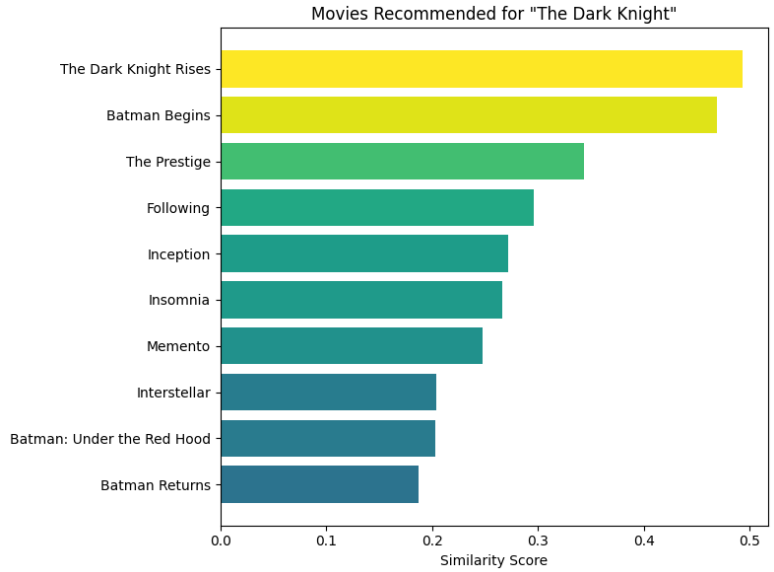


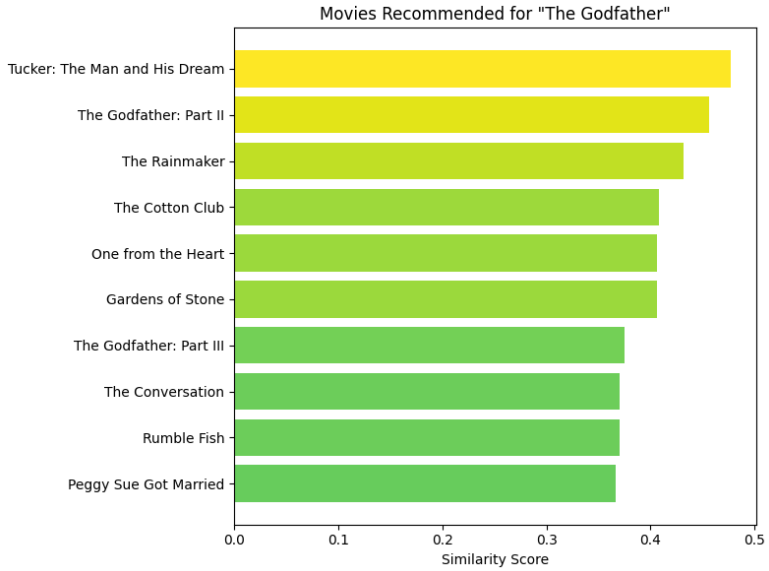
Overall, the simple recommendation systems based on the TMDB Ratings and the build_chart function provide useful tools for generating lists of top-rated movies in our dataset. By taking into account factors such as the number of votes, the average rating, and the popularity of a movie, these systems can help viewers discover quality movies that they may have otherwise overlooked.

Content-Based recommender system:

The content-based recommender system we're building consists of two parts: the Movie Description Based Recommender and the Metadata Based Recommender.

The Movie Description Based Recommender uses the overviews and taglines of movies to compute similarity scores between them. The get_recommendations function provided earlier is an example of this, which returns the top 30 movies that are most similar to the input movie based on their overviews and taglines.

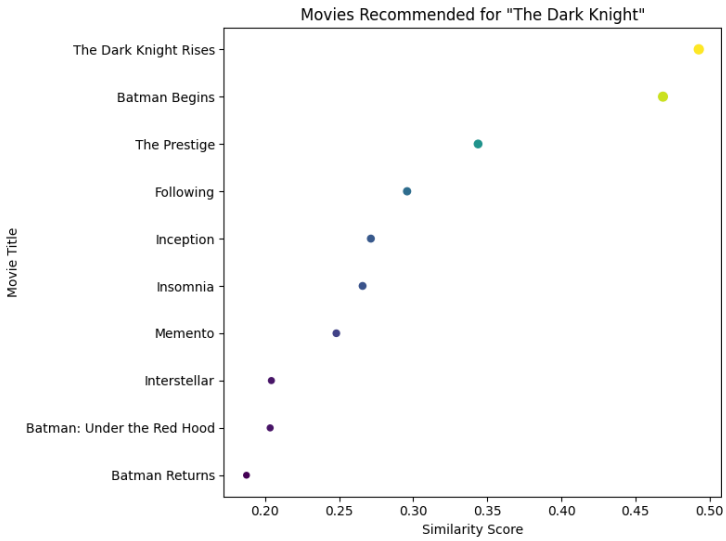


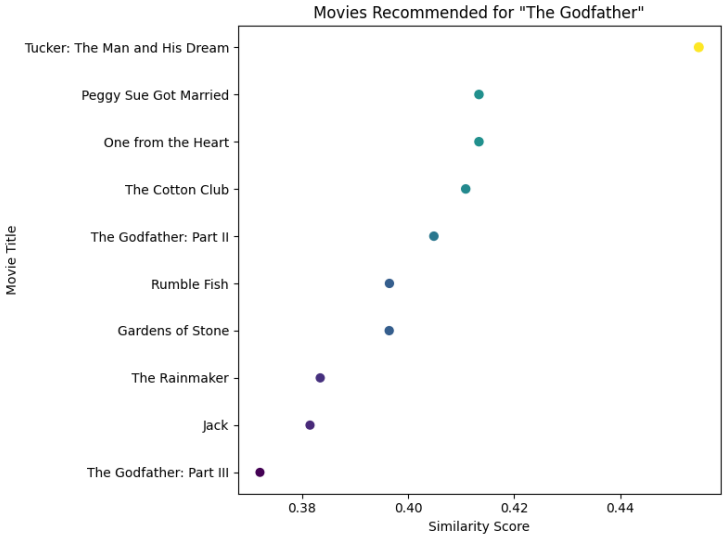


The Metadata Based Recommender, on the other hand, uses a combination of movie metadata such as cast, crew, genres, and keywords to create a metadata dump for each movie. This metadata dump is then used to compute similarity scores between movies using a count vectorizer, similar to the Movie Description Based Recommender. However, the metadata dump includes additional features such as directors and main actors, which are given more weight relative to the entire cast.

The metadata for each movie is preprocessed by stripping spaces and converting to lowercase to avoid confusion between similar-sounding names. The director is mentioned three times to give it more weight, and only the top three actors are selected to represent the cast.

The Metadata Based Recommender returns movies that are most similar to the input movie based on their metadata and can be customized by experimenting with different weights for the features, limiting the number of keywords used, weighing genres based on their frequency, and more.



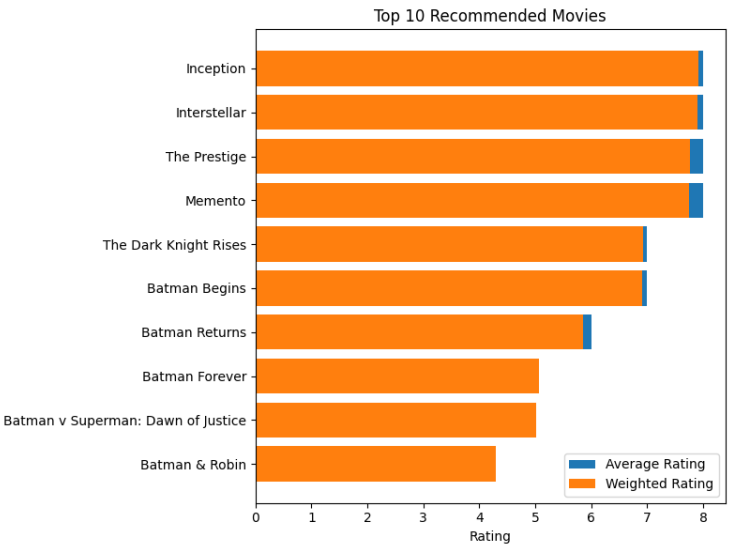


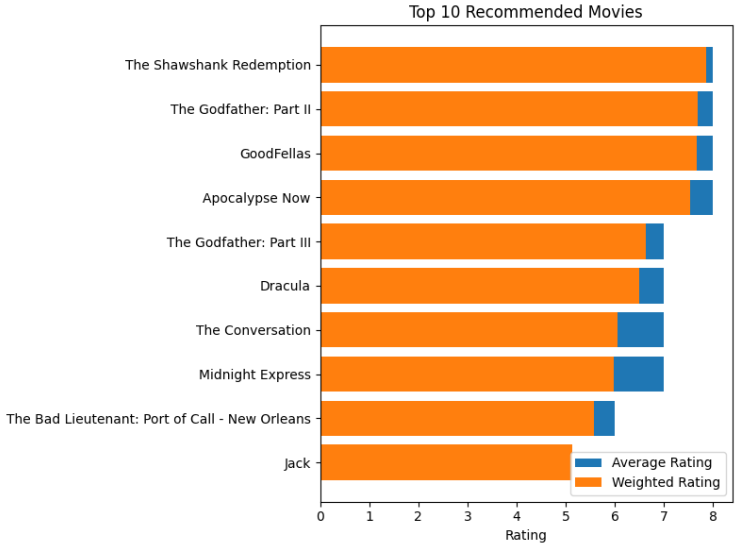
In addition to the Movie Description Based Recommender and the Metadata Based Recommender, we also have a part called Popularity and Ratings. This mechanism helps to remove bad movies from the recommendations and return movies that are popular and have received good critical responses.

The improved_recommendations function calculates the weighted rating of each movie using IMDB's formula, similar to the Simple Recommender section. It takes the top 25 movies based on similarity scores and calculates the vote of the 60th percentile movie. Using this as the value of m, it then selects movies that have a vote count greater than or equal to m and a non-null vote count and vote average. The function sorts these movies by their weighted ratings and returns the top 10 movies.

By using a combination of similarity scores and weighted ratings, the function is able to recommend movies that are similar to the input movie, but also have a good reputation among critics and audiences.

Therefore, the movies recommended by the improved_recommendations() function will be a combination of both the cosine similarity between movies and the weighted rating of each movie.





Hybrid Recommender Results:

The result of the hybrid recommender function is a list of top 10 similar movies to the input movie, sorted based on expected ratings by the particular user. The expected ratings are computed using the SVD algorithm, which takes into account the user's previous ratings and the similarity between the movies.

The hybrid recommender system combines the strengths of both content-based and collaborative filtering approaches, by using information about the content of the movies as well as the user's ratings and preferences. This allows the system to provide personalized recommendations that take into account both the user's tastes and the characteristics of the movies themselves.

However, it's important to note that the performance of the system may depend on the specific characteristics of the dataset, and it may be necessary to experiment with different algorithms and parameters to optimize the performance. Additionally, the system may not perform well for users who

have not rated many movies or for movies that have not been rated by many users, as there may not be enough data to make accurate predictions.

In summary, the hybrid recommender system provides a powerful approach to making personalized movie recommendations, by combining the strengths of both content-based and collaborative filtering techniques. The use of the SVD algorithm to predict expected ratings allows the system to make accurate and personalized recommendations, based on the user's previous ratings and the similarity between the movies.

Key Results:

The key results of our analysis show that the development of multiple recommendation systems based on different algorithms and approaches can lead to effective and personalized recommendations for users. Our first system, a simple recommender based on TMDB Vote Count and Vote Averages, provided top movie charts for general and specific genres using the IMDB Weighted Rating System. Our second set of systems were content-based recommenders that prioritized movies with higher ratings and more votes based on movie metadata and keywords. Our third system, a collaborative filtering system based on single value decomposition, gave estimated ratings for a given user and movie with an RMSE of less than 1. Finally, our hybrid engine combined ideas from content and collaborative filtering to provide personalized movie suggestions based on estimated ratings calculated for each user.

Overall, our analysis highlights the potential of using a variety of algorithms and approaches to build effective recommendation engines that can improve the user experience and provide personalized recommendations. By summarizing the key findings of our analysis, we demonstrate that the implementation of multiple recommendation systems can lead to promising results and provide valuable insights for businesses and platforms that seek to improve their recommendation algorithms. Future research should focus on identifying more complex algorithms and exploring new data sources to further improve the accuracy and efficiency of these recommendation systems.

SECTION VII: CONCLUSION

In this project, we developed four different recommendation systems based on different algorithms and ideas. Our first system was a simple recommender that used TMDB Vote Count and Vote Averages to build Top Movie Charts for both general and specific genres. We used the IMDB Weighted Rating System to calculate ratings for sorting.

The second set of systems were content-based recommenders that focused on movie metadata and keywords. We also created a filter to prioritize movies with higher ratings and more votes.

For our third system, we used the Surprise Library to develop a collaborative filtering system based on single value decomposition. This engine gave estimated ratings for a given user and movie, and achieved an RMSE of less than 1.

Finally, we developed a hybrid engine that combined ideas from content and collaborative filtering to give movie suggestions to a specific user based on estimated ratings calculated internally for that user.

Overall, our recommendation systems showed promising results and highlighted the potential of using different algorithms and approaches to build effective recommendation engines. Our findings can be useful for businesses and platforms looking to improve their user experience by providing personalized recommendations.

Future research can focus on improving the accuracy and efficiency of these recommendation systems by incorporating more complex algorithms and exploring new data sources.