Project Report: Movie Recommendation using EDA and KNN

Team Members:
Abhishek Kulkarni (A20516035)
Jhanavi Dave (A20515346)

GitHub Repository:
https://github.com/JhanaviDave/CS584_Final_Project_Group_A20515346_A20516035.git

Project Objective:
With the evolution in consumption of digital media, consumers are presented with a huge range of choices for their entertainment. This has proved to be a privilege but has also led to a decision-making paradox, overwhelming the user with a variety of options. The quality and similarity between the choices differ greatly thus, challenging the ability to choose content based on their unique preferences and interests.

Orthodox methods of recommendations for movies are based only on the type of genre or referring to the periodical hits. The data generated from ratings for movies, keywords from movie descriptions, and cast and crew, if utilized appropriately, can notably and accurately perform the recommendations and improve the user's experience while enjoying one of their favorite forms of media - movies.

This project aims to build a movie recommendation system based on the K-Nearest Neighbors (KNN) algorithm, and using a comprehensive database of ~5000 movies, providing recommendations tailored to similar movie preferences, helping users discover new movies they are likely to enjoy, thereby improving content discovery and user satisfaction. The KNN algorithm is trained on the processed dataset, to learn relationships between movies based on their features and user ratings. KNN algorithm is a popular choice for recommendation systems as it relies on the similarity between items (movies in our scenario) to make predictions.

Additionally, this recommendation system can also enable businesses to gain valuable insights into user behavior and preferences, which can inform content offerings and business decisions, if deployed at an enterprise level with a significantly larger dataset (preferably containing almost all the movies)

Data Summary:
The data used in this project is sourced from Kaggle - TMDB 5000 Movie Dataset, which has expansive movies-related data. This data is two-fold:
The first file tmdb_5000_credits.csv contains the cast and crew of 4813 unique movies, identified with movie names and movie IDs.
Second, the tmdb_5000_movies.csv contains detailed information about the movies, from which vote_average, genres, titles, keywords, and other movie-related information.

Some key attributes present in the dataset:
Title: The title of the movie.
Genres: The genres associated with the movie.
Language: The primary language of the movie.
Cast: Information about the actors, directors, etc.
Crew: Information about crew members involved in the movie.
Keywords: Array of keywords from movie description.
Vote_average: average user rating for the movie.

The columns present inconsistencies with missing values and array values(for example, the column keyword contains information in the array format) which will be ironed out in the data processing.

Functions Overview:

The project employs various Python packages and libraries such as Pandas, Scikit-learn, NumPy, Matplotlib, and Seaborn, for processing the dataset and implementation of cosine similarity, RMSE and MAPE functions.

NumPy:

NumPy library is open-source and widely used in the fields of data science and engineering. It is designed to handle large multi-dimensional arrays and matrices, and it also offers a suite of high-level mathematical functions to perform operations on these arrays. Due to its efficient memory usage, it operates faster than traditional Python lists. Being at the heart of the scientific Python and PyData ecosystems, NumPy finds its application in nearly every scientific and engineering domain.

Scikit-learn:

Scikit-learn, also known as sklearn, is a powerful Python library that is extensively used in the field of machine learning. It provides efficient tools for conducting predictive data analysis and is built upon other well-known Python libraries such as NumPy, SciPy, and matplotlib. The library includes modules for various tasks including classification, regression, clustering, and dimensionality reduction. It is designed to be accessible and reusable in different contexts. Scikit-learn is an open-source library and can be used commercially under the BSD license. Additionally, it offers packages for constructing linear models, and tree-based models, among others.

Matplotlib and Seaborn:

Matplotlib is used for creating static, animated, and interactive visualizations. Seaborn, a data visualization library in Python, is built on Matplotlib. It is known for its high-level interface that produces attractive and informative statistical graphics. Seaborn excels in simplifying the creation of complex statistical graphs and making them visually appealing. It integrates well with pandas data structures and its primary function is to aid in the exploration and understanding of data.

Cosine Similarity:

Cosine similarity is a fundamental measure in machine learning that quantifies the similarity between two non-zero vectors. It calculates the cosine of the angle between these two vectors when they are projected into a multi-dimensional space, thereby determining their directional similarity. This metric is extensively used in various tasks such as information retrieval, determining document similarity, recommendation systems, and clustering. One of the advantages of cosine similarity is that it addresses the limitations of the 'count-the-common-words' or Euclidean distance approach. The cosine similarity increases as the angle between the vectors decreases. The value it provides ranges between -1 and 1. Libraries and tools such as Matlab, SciKit-Learn, and TensorFlow commonly use this measure.

Pandas:

Pandas is a library in Python that offers quick, adaptable, and expressive data structures, making it straightforward and intuitive to work with relational or labeled data. It serves as a crucial high-level component for conducting practical data analysis in Python. The two main data structures in pandas, namely Series (1-dimensional) and DataFrame (2-dimensional), cater to the most typical use cases in fields such as finance, statistics, social science, and

various areas of engineering. Built on top of NumPy, pandas is designed to integrate seamlessly within a scientific computing environment. It excels in managing missing data, size mutability, automatic and explicit data alignment, powerful group-by functionality, intuitive merging and joining of data sets, flexible reshaping and pivoting of data sets, and robust IO tools.

RMSE:
Root Mean Squared Error (RMSE) is a vital measure in machine learning, predominantly utilized for assessing regression models. It measures the discrepancy between the predicted and actual values by computing the square root of the average of the squared differences. The emphasis of RMSE is on larger errors rather than smaller ones, providing a more cautious estimate of model accuracy when large errors are particularly undesirable. A model with a lower RMSE is considered to have a better fit for the data. RMSE finds its application in various fields such as sales forecasting, energy consumption prediction, and medical data analysis. Libraries and tools like Matlab, SciKit-Learn, and TensorFlow commonly use this measure.

MAPE:
Mean Absolute Percentage Error (MAPE) is a crucial measure in machine learning, utilized for assessing the precision of a model. It computes the average magnitude of the error generated by a model, indicating the average deviation of predictions. MAPE is essentially the percentage version of the Mean Absolute Error (MAE). It finds its application in various sectors, including finance and economic forecasting. A model with a lower MAPE value is considered to have more accurate predictions. However, MAPE does have its limitations. It is not applicable when the actual values contain instances of zero, as this would result in division by zero in the formula. Furthermore, MAPE can sometimes favor models that tend to under-forecast.

Methodology:

Content-based filtering technique in machine learning leverages item features to make recommendations. It suggests items to users based on the features of individual items and the user's past actions or explicit feedback. For example, if a user has shown interest in a certain type of movie, content-based filtering would suggest similar movies to the user. Some content-based recommendation algorithms match items according to descriptive features (like metadata) attached to items, rather than the actual content of an item. However, in this project content-based methods do match items according to intrinsic item attributes such as genres, keywords or director. This method enables the system to recommend movies that are similar to those the user has previously enjoyed.

Cosine similarity is a widely used technique in movie recommendation systems, especially in content-based filtering. This method measures the similarity between two non-zero vectors, which could represent the attributes of a movie or the preferences of a user in this context.

In this project, each movie is depicted as a vector in a multi-dimensional space, where each dimension corresponds to a different attribute of the movie, such as genre, director, and keywords. The cosine of the angle between these vectors is then calculated. This signifies the degree of similarity between the movies.

Data Processing and EDA:

EDA (Exploratory Data Analysis) was implemented to analyze datasets to summarize the main characteristics. Since it is typically conducted in the early stages of the data analysis process, we could gain insight into data discrepancies and were able to process data, engineer the features of the movie recommendation model, and enhance the performance by removing redundant data.

EDA was also used to analyze missing values by using basic data frame functions to find the total as well as feature level missing values and found none. Unnecessary columns were dropped from the analysis, to reduce the processing load on the model and various other functions were implemented to view details of the file like data averages, counts, standard deviations, etc.

	budget	id	popularity	revenue	runtime	vote_average	vote_count
count	4.803000e+03	4803.000000	4803.000000	4.803000e+03	4801.000000	4803.000000	4803.000000
mean	2.904504e+07	57165.484281	21.492301	8.226064e+07	106.875859	6.092172	690.217989
std	4.072239e+07	88694.614033	31.816650	1.628571e+08	22.611935	1.194612	1234.585891
min	0.000000e+00	5.000000	0.000000	0.000000e+00	0.000000	0.000000	0.000000
25%	7.900000e+05	9014.500000	4.668070	0.000000e+00	94.000000	5.600000	54.000000
50%	1.500000e+07	14629.000000	12.921594	1.917000e+07	103.000000	6.200000	235.000000
75%	4.000000e+07	58610.500000	28.313505	9.291719e+07	118.000000	6.800000	737.000000
max	3.800000e+08	459488.000000	875.581305	2.787965e+09	338.000000	10.000000	13752.000000

	movie_id
count	4803.000000
mean	57165.484281
std	88694.614033
min	5.000000
25%	9014.500000
50%	14629.000000
75%	58610.500000
max	459488.000000

Using nunique() function over both CSV files for movies and end credits, we were able to filter the unique column names as well, which gave us an overview of what the recommendation model could use to suggest movies.

```
# define unique column from movie csv file
movies.nunique()
```

✓ 0.0s

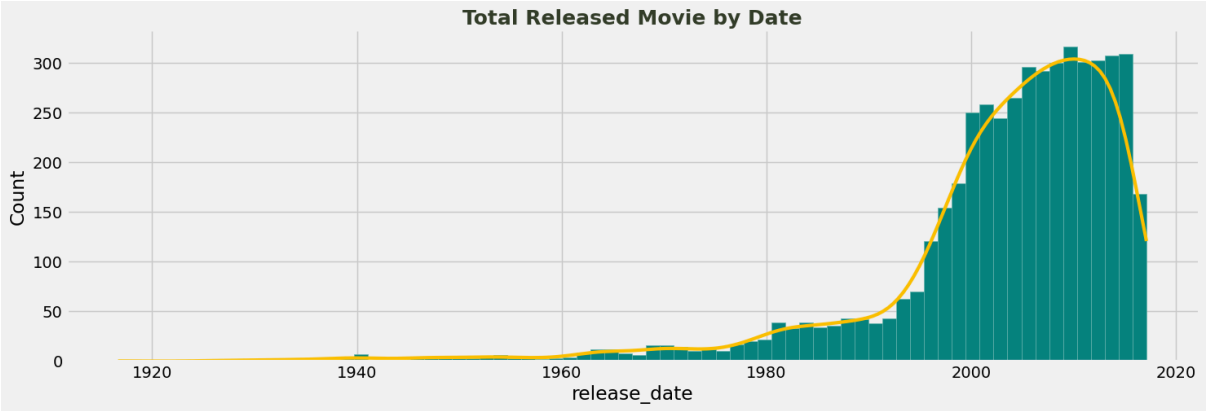
```
budget                  436
genres                 1175
homepage               1691
id                     4803
keywords               4222
original_language        37
original_title         4801
overview               4800
popularity             4802
production_companies   3697
production_countries    469
release_date           3280
revenue                3297
runtime                 156
spoken_languages        544
status                    3
tagline                3944
title                  4800
vote_average             71
vote_count             1609
dtype: int64
```

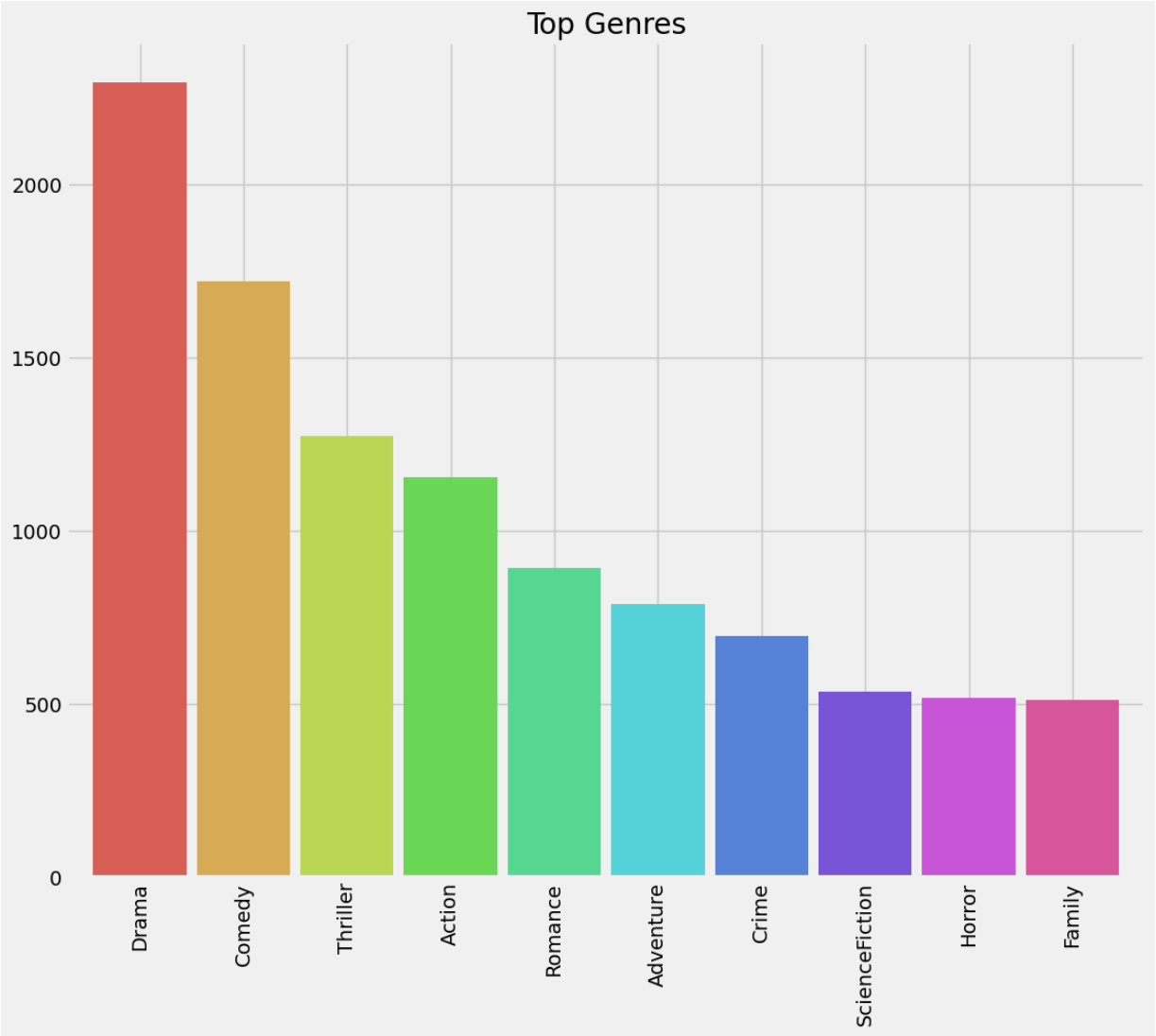
```
# define unique column from end credits csv file
credits.nunique()
```

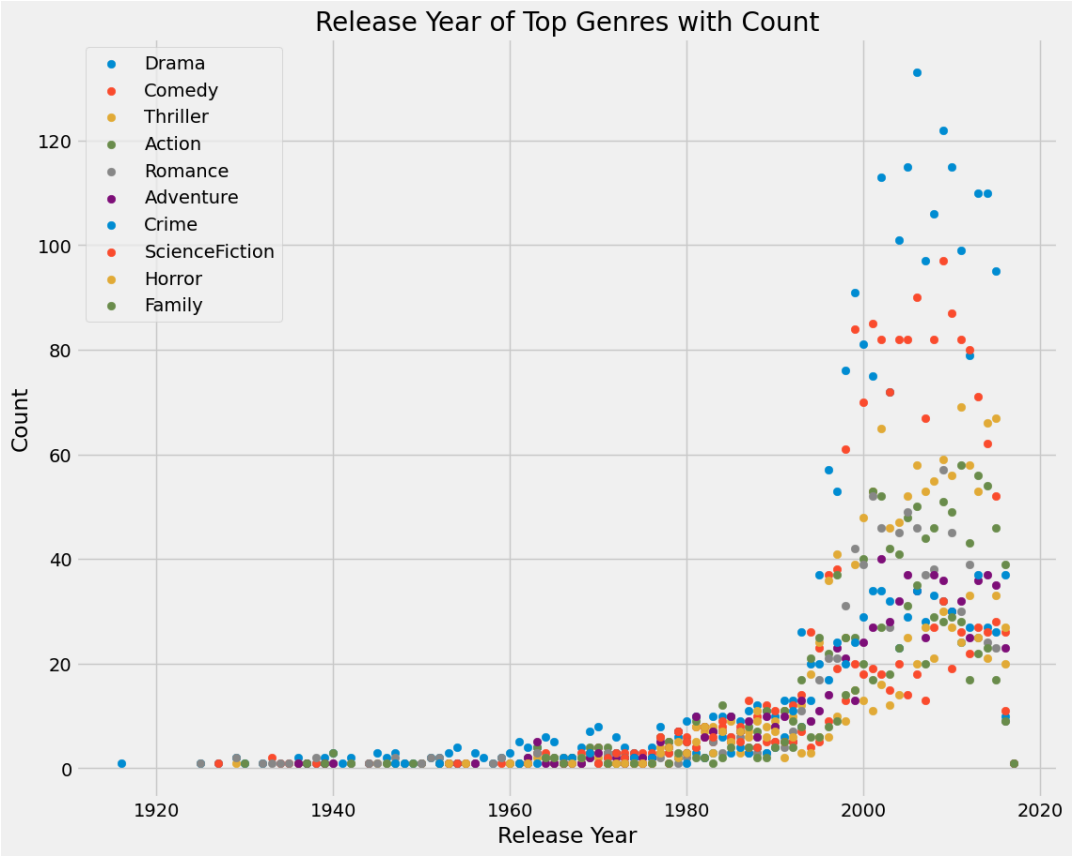
✓ 0.3s

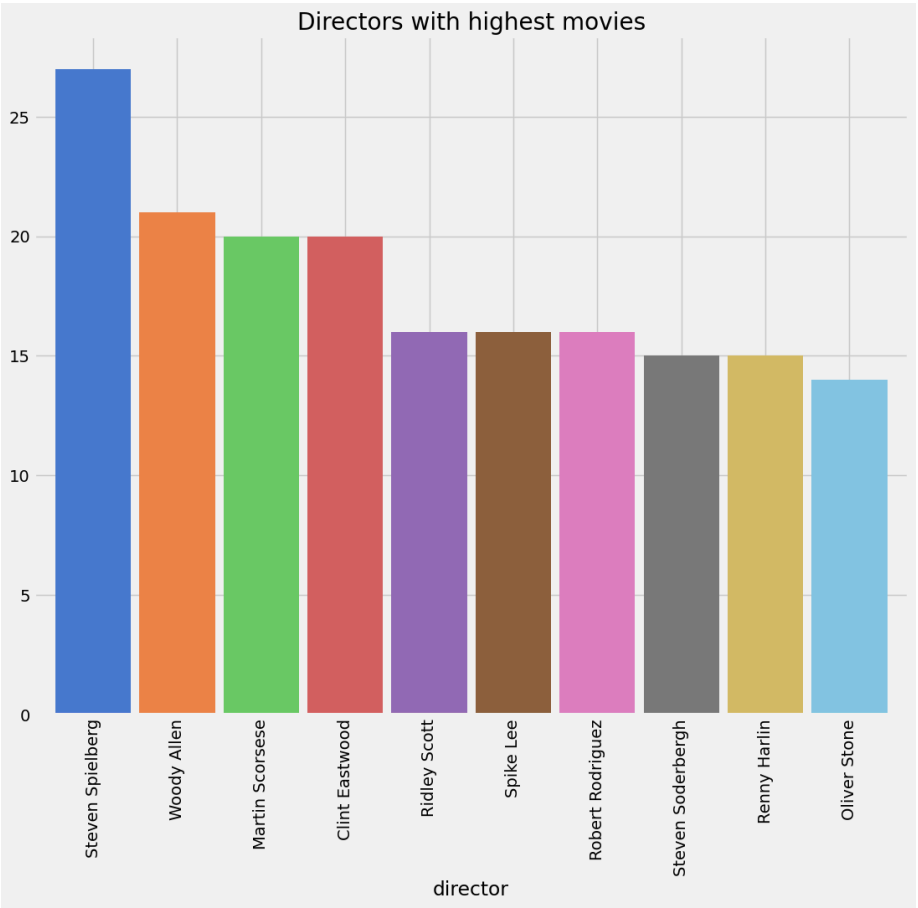
```
movie_id    4803
title       4800
cast        4761
crew        4776
dtype: int64
```

By implementing various data visualizations, we were able to highlight the top performers of the files, and using that data, we were able to recommend movies if the user requested other features for the recommendation. We could filter out the data based on movies released in certain months and years, with top genres over each year, and the directors with the highest count of movies directed in each year.


Total Released Movie by Date
Count
300
250
200
150
100
50
0
1920
1940
1960
1980
2000
2020
release_date


Top Genres
2000
1500
1000
500
0
Drama
Comedy
Thriller
Action
Romance
Adventure
Crime
ScienceFiction
Horror
Family


Release Year of Top Genres with Count
Drama
Comedy
Thriller
Action
Romance
Adventure
Crime
ScienceFiction
Horror
Family
Count
0
20
40
60
80
100
120
1920
1940
1960
1980
2000
2020
Release Year


Directors with highest movies
0
5
10
15
20
25
Steven Spielberg
Woody Allen
Martin Scorsese
Clint Eastwood
Ridley Scott
Spike Lee
Robert Rodriguez
Steven Soderbergh
Renny Harlin
Oliver Stone
director

WordCloud, which is a visual representation of text data, where the size of each word corresponds to its frequency of occurrence was also implemented. This highlighted the most commonly seen keywords, from plots, summaries, hashtags, genres, etc. The size of each word was the amount at which the words were repeated.

Results:

The results were both intriguing and satisfying. The recommendation system was able to recommend movies based on the user's input of watched movies, and the movies suggested were related, as we expected from the model. By executing the model function for many iterations where each iteration was a different movie, manually entered by the user, of various genres, had vivid casts and crews, with different runtimes and similar keywords for description, the similarity between the features were observed.

Using cosine similarity over each feature, the distance was calculated and a movie with the minimal distance in the neighborhood was suggested. We also calculated the Root Mean Square to analyze the distance between the movie watched by the user and the movie recommended by the model, measured over average user ratings - vote_average,and

observed Root Mean Squared Error for top 50 movies was observed to be 1.82, reducing as we increase the number of movies from 5 to 50 to 100. On the other side, Mean Absolute Percentage Error for top 50 movies was observed to be 19.751923255213462%. User feedback was also gathered from various users, and recommendations were approved by 8/10 users, concluding the performance on a positive success rate.

Conclusion:

In conclusion, this project leverages the cosine distance metric within a KNN framework to recommend movies based on their similarity to a given input. The movies with the smallest cosine distances as the nearest neighbors are selected and displayed as the recommended movies. By using cosine distance which is commonly implemented in KNN, the similarity between vectors is calculated. Since the accuracy of the cosine angle and the equidistance of parameters remains almost the same for the cosine distance, this proves as an optimized model.

The model then finally predicts the name of the recommended movies similar to the given input, by matching the title to the movie name in the dataset, then finding the 10-most similar movies using the similarity function, and then recommending the most similar movies with their details.

Future Work:

Including other parameters like user preferences and behaviors, like runtimes, popularity, and explicit feedback using UI/UX from the dataset, would make the recommendations more personalized and accurate. This can be achieved through collaborative filtering techniques. The model will also rely on user preferences and ratings from other users along with other parameters, with a neighbor calculating function that finds the K-Nearest Neighbors of a given movie based on the similarity between the movies in the dataset. Combining content-based filtering with collaborative filtering or other recommendation techniques (hybrid systems) often leads to a robust and accurate recommendation model.

This approach might leverage the strengths of multiple recommendation methods to mitigate their limitations. Recommendations using additional features, such as movie release dates, language, and user-generated tags, could also provide more diverse and informative attributes for recommendation. Conducting thorough evaluation and testing of the recommendation system using techniques such as cross-validation and testing to measure its performance and effectiveness in real-world scenarios

References:

1. Comparative study of recommender system approaches and movie recommendation using collaborative filtering, Taushif Anwar & V. Uma
2. Movie Recommender System Using Collaborative Filtering, Meenu Gupta; Aditya Thakkar et al.
3. Movie Recommender System Using Parameter Tuning of User and Movie Neighbourhood via Co-Clustering, Sonu Airen, Jitendra Agrawal
4. Movie Recommendation System Using Machine Learning, F. Furtado, A. Singh
5. https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata
6. Movie Recommendation and Rating Prediction Using K-Nearest Neighbors