MAR ATHANASIUS COLLEGE OF ENGINEERING
(Affiliated to APJ Abdul Kalam Technological University, TVM)
KOTHAMANGALAM

Department of Computer Applications

Mini Project Report

MOVIE RECOMMENDATION SYSTEM USING MACHINE LEARNING

Done by

Fathima Sana

Reg No:MAC22MCA-2014

Under the guidance of
Prof. Biju Skaria

2022-2024

MAR ATHANASIUS COLLEGE OF ENGINEERING
(Affiliated to APJ Abdul Kalam Technological University, TVM)
KOTHAMANGALAM

CERTIFICATE

KNOWLEDGE IS POWER

Movie Recommendation System Using Machine Learning

Certified that this is the bonafide record of project work done by

Fathima Sana
Reg No: MAC22MCA-2014

during the third semester, in partial fulfilment of requirements for award of the degree

Master of Computer Applications

of

APJ Abdul Kalam Technological University Thiruvananthapuram

Faculty Guide	**Head of the Department**
Prof. Biju Skaria	Prof. Biju Skaria
Project Coordinator	**Internal Examiners**
Prof. Nisha Markose	

ACKNOWLEDGEMENT

With heartfelt gratitude, I extend my deepest thanks to the Almighty for His unwavering grace and blessings that have made this journey possible. May His guidance continue to illuminate my path in the years ahead.

I am immensely thankful to Prof. Biju Skaria, Head of the Department of Computer Applications and my mini project guide, and Prof. Nisha Markose, our dedicated project coordinator, for their invaluable guidance and timely advice, which played a pivotal role in shaping this project. Their guidance, constant supervision, and provision of essential information were instrumental in the successful completion of the mini project.

I extend my profound thanks to all the professors in the department and the entire staff at MACE for their unwavering support and inspiration throughout my academic journey. My sincere appreciation goes to my beloved parents, whose guidance has been a beacon in every step of my path.

I am also grateful to my friends and individuals who generously shared their expertise and assistance, contributing significantly to the fulfillment of this endeavor.

ABSTRACT

The "Movie Recommendation System Using Machine Learning" project is focused on enhancing the user experience within streaming services by offering personalized movie recommendations based on individual tastes. This project employs the collaborative filtering methodology, utilizing the K-nearest neighbor (KNN) algorithm to achieve this goal. Collaborative filtering is divided into two main approaches: finding users with similar tastes and recommending movies they like, and identifying similar movies based on user ratings. By leveraging user engagement data, particularly ratings, the system aims to create accurate and tailored movie suggestions.

The KNN algorithm identifies the K most similar users or items based on historical preferences, allowing the system to recommend movies based on the preferences of like-minded users. This approach capitalizes on the concept that users with similar tastes tend to enjoy similar movies, ultimately leading to more accurate and satisfying recommendations.

The expected outcome of this project is an advanced movie recommender system capable of delivering precise and personalized movie suggestions. By incorporating the K-nearest neighbour algorithm and collaborative filtering, this system has the potential to significantly improve recommendation accuracy and relevance. To develop and evaluate the recommendation system, the Movie lens 100k dataset is used, which provides comprehensive data on user preferences, demographics, and movie details. This dataset serves as a valuable resource for implementing various recommendation algorithms and gaining deeper insights into user behavior.

In summary, this project addresses the challenge of content discovery on streaming platforms by offering highly personalized movie recommendations. Through the application of collaborative filtering techniques, it aims to enhance user satisfaction, engagement, and platform performance in the dynamic field of entertainment.

LIST OF TABLES

2.1 Reference papers summary 5

LIST OF FIGURES

3.1 Snapshot of ratings.csv 7

3.2 Snapshot of movies.csv 8

3.3 Snapshot of users.csv 8

3.4 Number of Movies, Users and Ratings 9

3.5 Data type of Each Attribute 9

3.6 Dropped attribute timestamp 10

3.7 Merged movie title… 10

3.8 Handling missing values and checking duplicates 11

3.9 Number of ratings per user 11

3.10 Number of ratings per movie 12

3.11 Occupation distribution of users 15

3.12 Ratings distribution 16

3.13 Pie chart for movie genre count 16

3.14 Boxplot for ratings 17

3.15 Example figure of kNN 19

3.16 Activity Diagram 21

3.17 Project Pipeline 22

4.1 User feature dataframe 31

4.2 Model knn1 31

4.3 Model knn2 32

4.4 Code snippet for user based recommendation 32

4.5 Recommendation for user 23… 33

4.6 Code snippet for item based recommendation 33

4.7 Recommendation for the movie ‘Speed’ 34

5.1 User movie input 35

5.2 Item based recommendation… 35

5.3 User ID input 35

5.4 User based recommendation 36

6.1 Index page of movie recommender 38
6.2 Input page of movie recommender 38
6.3 User id input 39
6.4 Similar users in recommendation for user 23 39
6.5 Recommendation for user 12 40
6.6 Movie title input 40
6.7 Recommendation for the movie’12 Angry Men’ 41
6.8 Validation for user id 41
6.9 Validation for movie title 42
7.1 Git history of 3 sprints… 43
7.2 Git history of sprint 3 43

CONTENTS

1 Introduction **1**

2 Supporting Literature **2**

2.1 Literature Review 2

2.1.1 Summary Table 5

2.2 Findings and Proposals 6

3 System Analysis **7**

3.1 Analysis of Dataset 7

3.1.1 About the Dataset 7

3.1.2 Explore the dataset 9

3.2 Data Pre-processing 10

3.2.1 Data Cleaning 10

3.2.2 Analysis of Feature Variables 13

3.2.3 Analysis of Class Variables 15

3.3 Data Visualization 15

3.4 Analysis of Algorithm 18

3.4.1 Activity Diagram 21

3.5 Project Plan 22

3.5.1 Project Pipeline 22

3.5.2 Project Implementation Plan 23

3.6 Feasibility Analysis 24

3.6.1 Technical Feasibility 24

3.6.2 Economic Feasibility 25

3.6.3 Operational Feasibility 26

3.7 System Environment 27

3.7.1 Software Environment 27

3.7.2 Hardware Environment 29

4 System Design **30**

4.1 Model Building 30

4.1.1 Model Planning.. 30
4.1.2 Training..34
4.1.3 Testing 34

5 Results and Discussion 35

6 Model Deployment 37

7 Git History 43

8 Conclusion 44

9 Future Work 45

10 Appendix 47

10.1 Minimum Software Requirements 47
10.2 Minimum Hardware Requirements...49

11 References 50

1. INTRODUCTION

In today's digital landscape, the sheer abundance of content available on platforms like Netflix, Hulu, and others has given rise to the crucial role played by recommendation systems. These systems are akin to intelligent curators, utilizing sophisticated algorithms to predict and propose content that aligns with users' preferences. Nowhere is this more relevant than in the realm of movies, where the diversity of genres, languages, and platforms necessitates a more tailored approach to content discovery.

In the face of information overload, recommendation systems act as invaluable navigators, helping users cut through the noise and discover content that resonates with their tastes. Whether it's movies, music, books, or other forms of media, these systems provide a personalized touch to the user experience, simplifying the process of finding relevant and enjoyable content. The entertainment industry, particularly in the context of movies, has witnessed a paradigm shift toward personalized content recommendation.

At the core of personalized content recommendation is the goal of enhancing user satisfaction and engagement. By offering suggestions based on individual viewing history, preferences, and ratings, recommendation systems contribute to a more immersive and enjoyable user experience. In the context of movies, the ability to tailor suggestions based on a user's cinematic preferences simplifies the content discovery process, fostering a more meaningful connection between the user and the content.

In this project, our focus is on the development of a movie recommendation system, employing collaborative filtering with k-nearest neighbors (KNN) and cosine similarity. Through the exploration of these techniques, we aim to contribute to the ongoing evolution of personalized content recommendation, particularly within the cinematic domain, with the ultimate objective of enriching the digital viewing experience for users.

2. SUPPORTING LITERATURE

2.1 Literature Review

Paper 1: Gupta, Meenu, et al. "Movie recommender system using collaborative filtering." *2020 international conference on electronics and sustainable communication systems (ICESC)*. IEEE, 2020

In the rapidly evolving landscape of digital entertainment, the demand for personalized and accurate movie recommendations has become paramount. Gupta, Meenu, et al.'s paper, "Movie Recommender System Using Collaborative Filtering," presented at the 2020 International Conference on Electronics and Sustainable Communication Systems (ICESC), addresses this need by introducing an innovative approach to augment movie recommendations through collaborative filtering. Collaborative filtering, a widely employed technique in recommender systems, capitalizes on user preferences and similarities to make precise predictions. The authors recognize the significance of this method and propose a novel algorithm that seamlessly integrates both item-based and user-based collaborative filtering techniques. This dual approach aims to enhance the accuracy of movie suggestions, catering to the diverse tastes and preferences of users in the digital era. The paper begins by acknowledging the evolving landscape of digital entertainment and the necessity for recommender systems to adapt. It emphasizes the need for solutions that not only provide accurate suggestions but also account for individual preferences in a dynamic and evolving user environment. The proposed algorithm, as detailed by the authors, represents a significant step forward in the realm of collaborative filtering. By incorporating both item-based and user-based approaches, the algorithm seeks to address the limitations of traditional methods that rely solely on one of these techniques. The fusion of these methodologies allows for a more nuanced understanding of user behavior and preferences, leading to more accurate and personalized recommendations. In conclusion, Gupta, Meenu, et al.'s paper stands as a pivotal contribution to the field of recommender systems. By proposing a novel collaborative filtering algorithm that seamlessly combines item-based and user-based approaches, the authors address the contemporary challenges of providing accurate and personalized movie recommendations in the digital age.

Paper 2: Raghavendra, C. K., and K. C. Srikantaiah. "Similarity based collaborative filtering model for Movie Recommendation Systems." *2021 5th international conference on intelligent computing and control systems*

Raghavendra, C. K., and K. C. Srikantaiah's paper, "Similarity based collaborative filtering model for Movie Recommendation Systems," presented at the 2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS), conducts an in-depth exploration of collaborative filtering techniques in the context of movie recommendation systems. The authors delve into various methodologies, emphasizing the significance of similarity-based approaches for achieving heightened accuracy in movie recommendations. The paper opens with a meticulous literature review, positioning collaborative filtering as a cornerstone in the field of recommender systems. Recognizing the diverse range of approaches, Raghavendra and Srikantaiah focus on similarity-based models, underscoring their pivotal role in refining recommendation accuracy. The review not only provides a comprehensive overview of existing algorithms but also sheds light on the evolving landscape of collaborative filtering. One of the paper's key contributions lies in the nuanced analysis of different similarity measures and their impact on recommendation accuracy. The authors delve into the intricacies of similarity-based models, scrutinizing the effectiveness of measures such as cosine similarity, Pearson correlation, and Jaccard index. The authors go beyond a mere exposition of existing models; they critically evaluate the challenges and limitations inherent in current collaborative filtering approaches. Raghavendra and Srikantaiah highlight the need for personalized recommendations that go beyond traditional user-item interactions, emphasizing the incorporation of user preferences, item features, and contextual information to achieve a more holistic understanding of user behavior. In conclusion, "Similarity based collaborative filtering model for Movie Recommendation Systems" provides a comprehensive and insightful exploration of collaborative filtering techniques in the realm of movie recommendations. The authors' emphasis on similarity-based models, detailed analysis of similarity measures, and identification of challenges contribute significantly to the understanding of this dynamic field. This paper not only informs current practices but also inspires future research to propel movie recommendation systems towards higher levels of accuracy and personalization.

Paper 3: Anwar, Taushif, and V. Uma. "Comparative study of recommender system approaches and movie recommendation using collaborative filtering." *International Journal of System Assurance Engineering and Management* 12 (2021): 426-436

Taushif Anwar's paper, "Comparative Study of Recommender System Approaches and Movie Recommendation Using Collaborative Filtering," published in the International Journal of System Assurance Engineering and Management in 2021, undertakes a thorough exploration of diverse methodologies within collaborative filtering for movie recommendations. The author strategically examines the nuances of user-based and item-based approaches, shedding light on their respective strengths and limitations. The literature review within the paper unfolds the landscape of collaborative filtering, a widely adopted technique harnessing user behavior data to curate personalized movie recommendations. Anwar meticulously dissects the foundations of collaborative filtering, distinguishing between user-based and item-based methods. User-based methodologies revolve around the identification of similar users, leveraging their preferences to generate recommendations, while item-based approaches center on discerning similarities among items or movies themselves. This dichotomy provides a nuanced understanding of the intricate strategies deployed in collaborative filtering. The review also brings to the forefront the challenges intrinsic to collaborative filtering, with a spotlight on the cold start problem and sparsity issues. Anwar recognizes the complexities that arise when dealing with sparse datasets and the inherent difficulties in initiating recommendations for new users or items. This acknowledgment adds depth to the literature review, presenting a holistic view of the hurdles practitioners face in implementing collaborative filtering models for movie recommendations. In navigating the evolution of recommendation algorithms, Anwar extends the discussion to encompass matrix factorization techniques and hybrid models. These advancements mark a paradigm shift in the quest for enhanced accuracy and effectiveness in movie recommendations. The paper emphasizes the amalgamation of multiple approaches to create hybrid models, showcasing the dynamism of contemporary recommendation systems. In conclusion, "Comparative Study of Recommender System Approaches and Movie Recommendation Using Collaborative Filtering" serves as a compass in navigating the intricate realm of collaborative filtering for movie recommendations. Anwar's meticulous literature review encapsulates the essence of user-based and item-based methods, addresses challenges, and spotlights advancements in recommendation algorithms.

2.1.1. Summary Table

PAPER TITLE	ALGORITHM	DATASET	METHODOLOGY	PRECISION/ RMSE
Movie Recommender System Using Collaborative Filtering, 2020 International Conference on Electronics and Sustainable Communication Systems (ICESC)	K-Nearest Neighbors (KNN)	Movie Lens (Kaggle)	Collaborative filtering techniques, such as user-based or item-based similarity calculations, to make movie recommendations based on user ratings and interactions.	PRECISION: Content-based: 0.501 Collaborative filtering: 0.782
Similarity Based Collaborative Filtering Model for Movie Recommendati on Systems, 2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)	K-Nearest Neighbors	Movie Lens (Kaggle)	Computing user or item similarities based on movie ratings and using these similarities to make movie recommendations in a Similarity Based Collaborative Filtering Model	PRECISION: Item Based Collaborative Filtering: 0.84 User Based Collaborative Filtering: 0.76
Comparative study of recommender system Approaches and movie recommendatio n using collaborative filtering, International journal of System Assurance Engineering and Management	K- Nearest Neighbours SVD SVD++ Co- Clustering	Movie Lens 100k (Group Lens)	Comparative Study to analyze different recommender system approaches, focusing on collaborative filtering and advancements in recommendation algorithms.	RMSE: KNN: 0.979 SVD: 0.9358 SVD++: 0.9201 Co-Clustering: 0.9675

Table 2.1 Reference papers summary

2.2. Findings and Proposals

After a comprehensive review of three noteworthy papers in the domain of movie recommendation systems, it is evident that each contributes significantly to the understanding and advancement of collaborative filtering techniques. In the quest for an optimal recommendation algorithm, the choice of k-Nearest Neighbors (k-NN) with Cosine Similarity emerges as a prudent decision. Considering these insights, the choice of k-NN with cosine similarity emerges as a robust solution. Cosine similarity is well-suited for collaborative filtering tasks, particularly in scenarios where the emphasis is on the direction of vectors (user's preferences) rather than their magnitudes. The k-NN approach leverages the wisdom of the crowd, identifying similar users or items and making recommendations based on their collective preferences.

The first paper by Gupta, Meenu et al. introduces a novel movie recommender system that combines item-based and user-based collaborative filtering methods. The algorithm proposed employs k-NN with cosine angle similarity, showcasing its efficacy in enhancing recommendation accuracy (Precision: 0.782). The second paper by Raghavendra, C. K., and K. C. Srikantaiah emphasizes similarity-based collaborative filtering models, employing k-NN with Pearson Correlation. Their findings reveal a high precision rate of 0.84 In the third paper, Anwar delves into a comparative study of recommender system approaches, evaluating k-NN, SVD, SVD++, and Co-clustering. The collaborative filtering approach, including k-NN, proves effective, achieving an RMSE of 0.9201. Moreover, the k-NN algorithm aligns with the recurring theme across the papers - personalization. It adapts to individual user behaviors, ensuring that recommendations are tailored to specific tastes and preferences. This not only enhances accuracy but also fosters user satisfaction and engagement, addressing the core objective of movie recommendation systems.

In conclusion, the adoption of k-NN with cosine similarity is supported by empirical evidence from the reviewed papers, showcasing its effectiveness in collaborative filtering for movie recommendations. This approach aligns with the evolving landscape of recommender systems, promising a balance between accuracy, personalization, and user engagement.

.

3. SYSTEM ANALYSIS

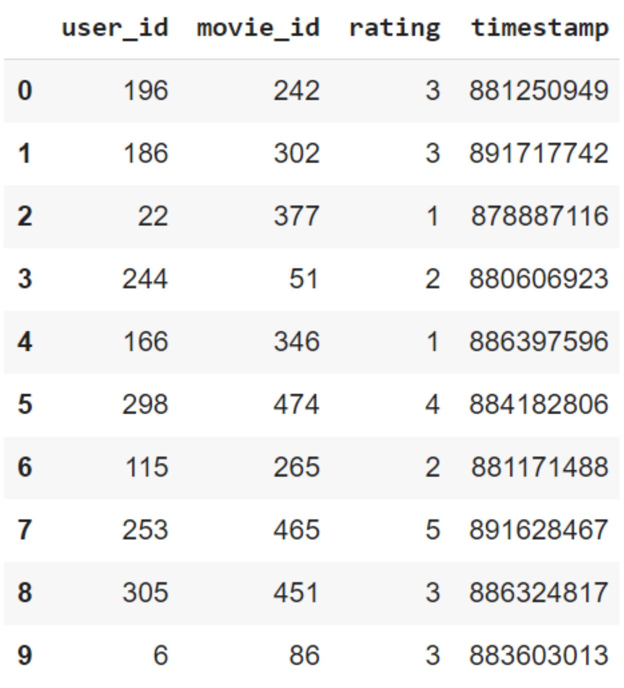
3.1. Analysis of Dataset

3.1.1. About the Dataset

The Movie Lens 100K dataset is a widely used and well-known dataset in the field of recommender systems and collaborative filtering. It was created by the Group Lens Research project and contains a collection of movie ratings and user information.

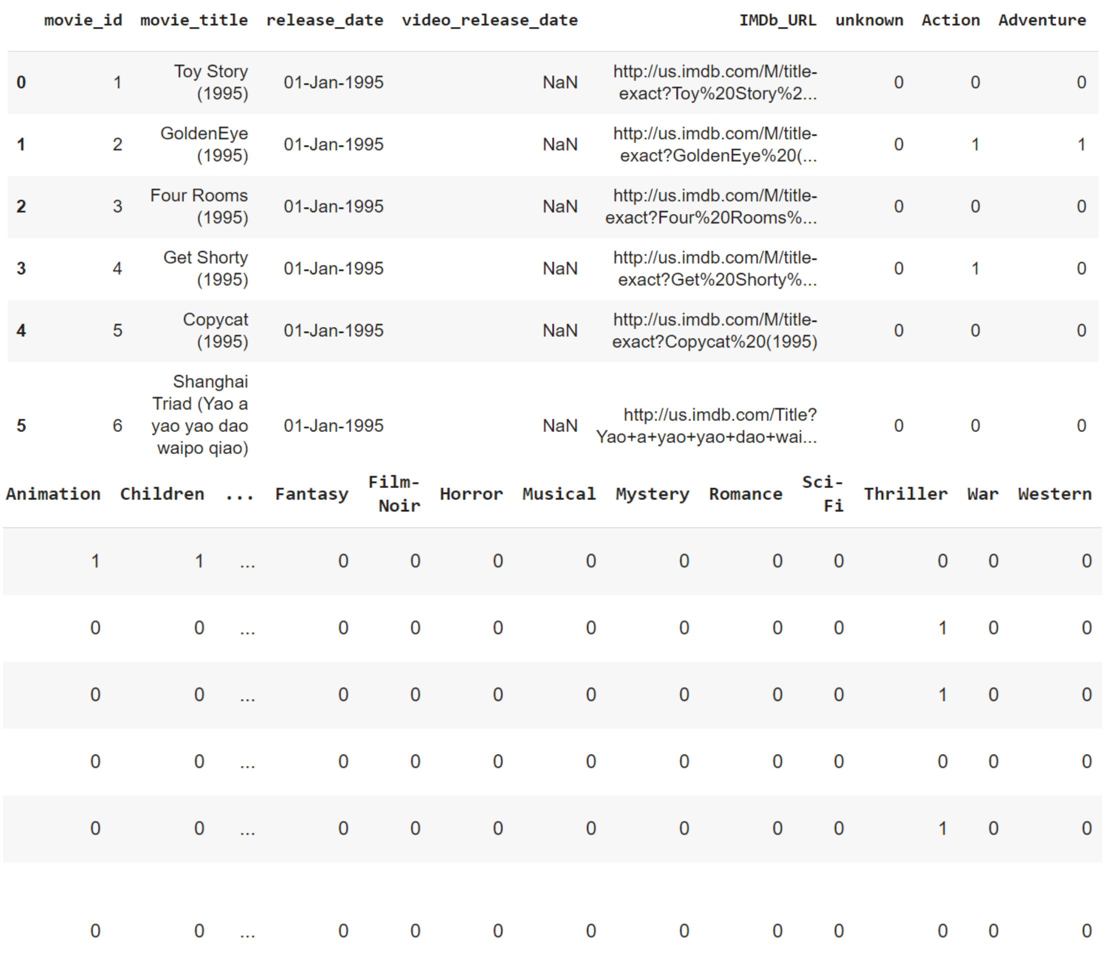
Dataset Link: https://www.kaggle.com/datasets/prajitdatta/movielens-100k-dataset

Size: The dataset contains 100,000 ratings provided by 943 users for 1,682 movies.
Data Split: It is divided into several files, including user data, movie data, and ratings data, to facilitate different types of analyses and modelling.
Sparsity: Due to the large number of movies and users, the dataset is sparse, meaning that not all users have rated all movies, resulting in missing data.



	user_id	movie_id	rating	timestamp
0	196	242	3	881250949
1	186	302	3	891717742
2	22	377	1	878887116
3	244	51	2	880606923
4	166	346	1	886397596
5	298	474	4	884182806
6	115	265	2	881171488
7	253	465	5	891628467
8	305	451	3	886324817
9	6	86	3	883603013

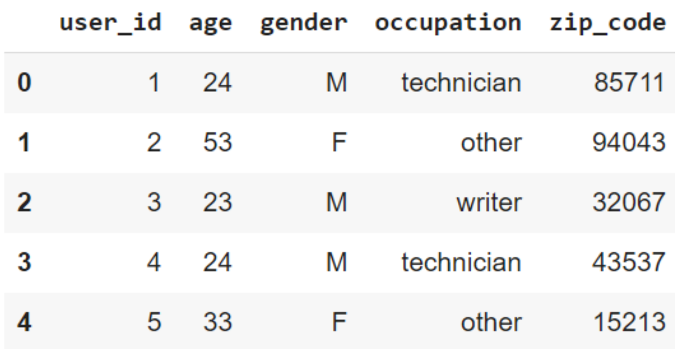
Fig 3.1 Snapshot of ratings.csv



	movie_id	movie_title	release_date	video_release_date	IMDb_URL	unknown	Action	Adventure
0	1	Toy Story (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Toy%20Story%2...	0	0	0
1	2	GoldenEye (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?GoldenEye%20(...	0	1	1
2	3	Four Rooms (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Four%20Rooms%...	0	0	0
3	4	Get Shorty (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Get%20Shorty%...	0	1	0
4	5	Copycat (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Copycat%20(1995)	0	0	0
5	6	Shanghai Triad (Yao a yao yao dao waipo qiao)	01-Jan-1995	NaN	http://us.imdb.com/Title?Yao+a+yao+yao+dao+wai...	0	0	0

Animation	Children	...	Fantasy	Film-Noir	Horror	Musical	Mystery	Romance	Sci-Fi	Thriller	War	Western
1	1	...	0	0	0	0	0	0	0	0	0	0
0	0	...	0	0	0	0	0	0	0	1	0	0
0	0	...	0	0	0	0	0	0	0	1	0	0
0	0	...	0	0	0	0	0	0	0	0	0	0
0	0	...	0	0	0	0	0	0	0	1	0	0
0	0	...	0	0	0	0	0	0	0	0	0	0

Fig 3.2 Snapshot of movies.csv



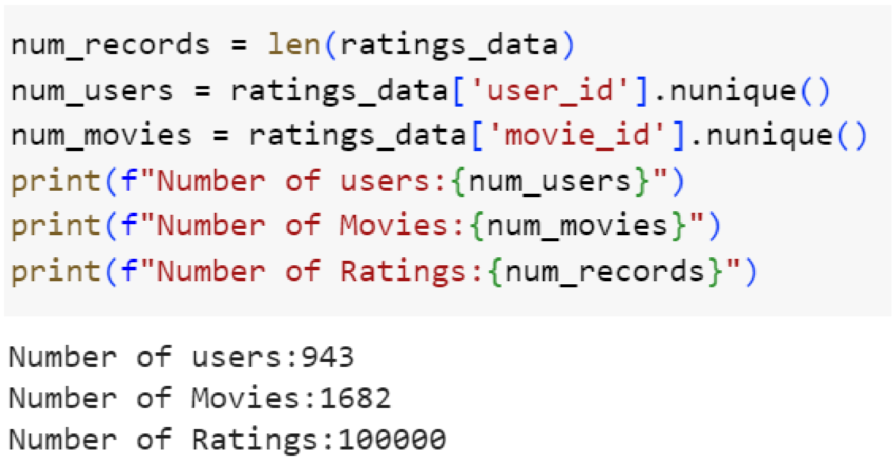
	user_id	age	gender	occupation	zip_code
0	1	24	M	technician	85711
1	2	53	F	other	94043
2	3	23	M	writer	32067
3	4	24	M	technician	43537
4	5	33	F	other	15213

Fig 3.3 Snapshot of users.csv

This is the result produce by listing the dataset files using head() function.

3.1.2. Explore the Dataset

The attributes in this dataset are Academic percentage in different subjects, personality related questions, attitude level in a situation, coding ability etc. And the class label is the suggested job role which is found by analyzing this huge dataset. The expected type of values for each feature is as follows:

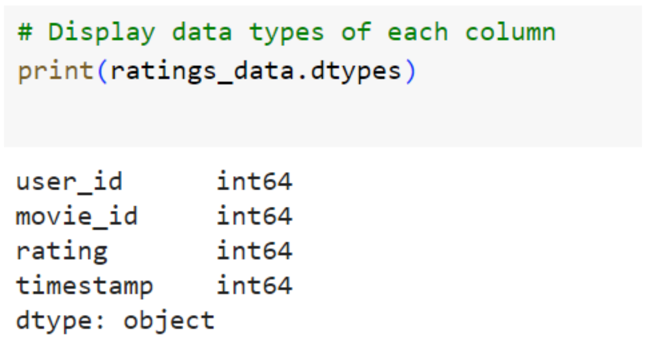


```
num_records = len(ratings_data)
num_users = ratings_data['user_id'].nunique()
num_movies = ratings_data['movie_id'].nunique()
print(f"Number of users:{num_users}")
print(f"Number of Movies:{num_movies}")
print(f"Number of Ratings:{num_records}")
```

```
Number of users:943
Number of Movies:1682
Number of Ratings:100000
```

Fig 3.4 Number of users, movies and ratings

Ratings_data is only considered for the project which includes user_id, movie_id , rating and timestamp.



```
# Display data types of each column
print(ratings_data.dtypes)
```

```
user_id      int64
movie_id     int64
rating       int64
timestamp    int64
dtype: object
```

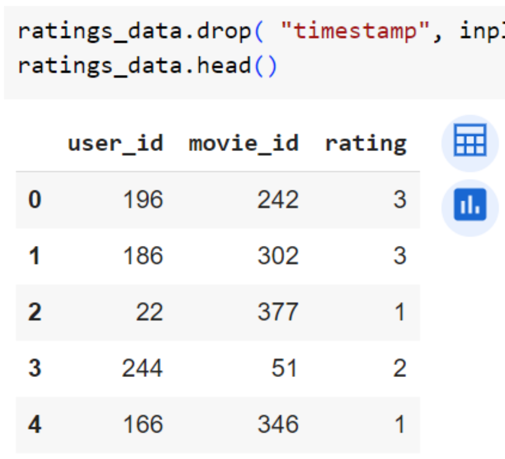
Fig 3.5 Datatypes of each attribute

3.2. Data Preprocessing

3.2.1. Data Cleaning

Data Cleaning is the data pre-processing method we choose. Data cleaning routines attempt to fill in missing values, smooth out noisy data and correct inconsistencies. The dataset taken is already pre-processed, so pre-processing techniques are not need for the dataset.

Firstly, timestamp is dropped since it is not required for recommendation.

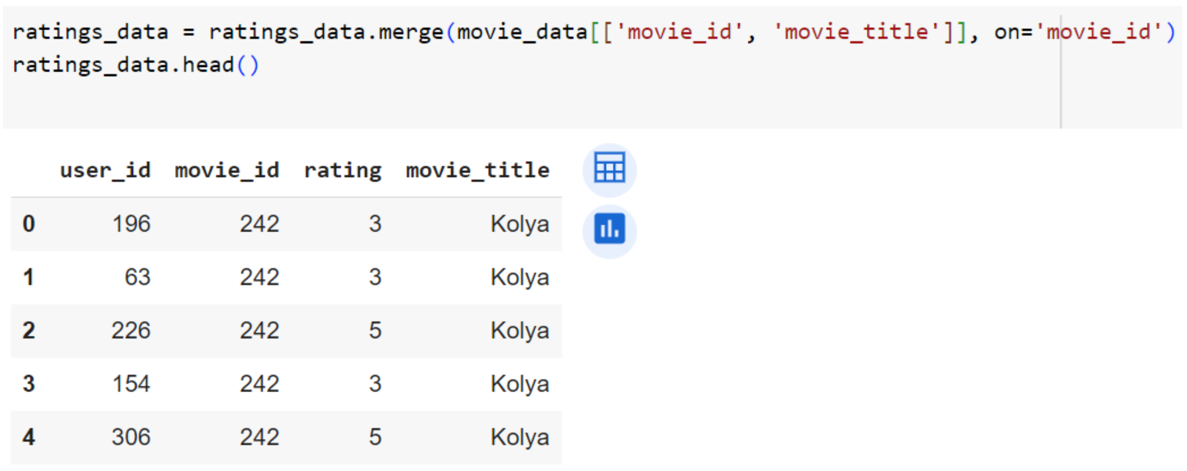


```
ratings_data.drop( "timestamp", inplace = True, axis = 1 )
ratings_data.head()
```

```
   user_id  movie_id  rating
0      196       242       3
1      186       302       3
2       22       377       1
3      244        51       2
4      166       346       1
```

Fig 3.6 Dropped attribute timestamp

movie_title merged to the ratings_data

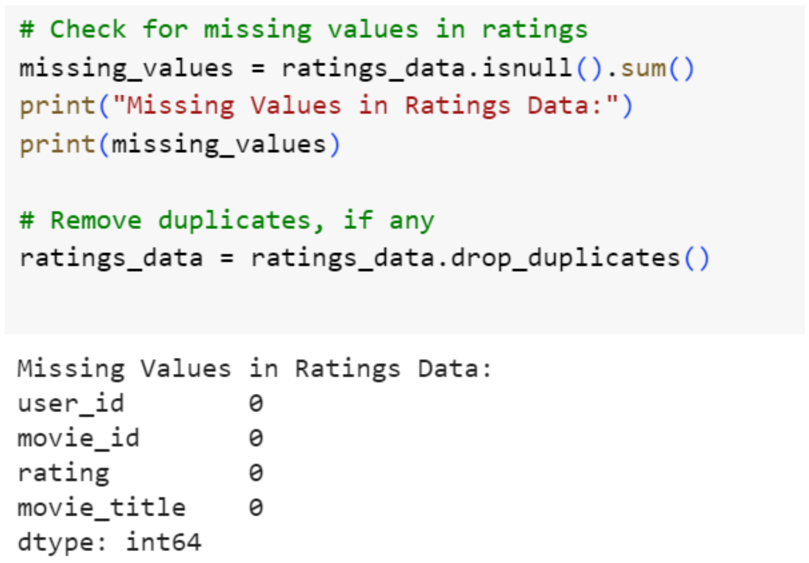


```
ratings_data = ratings_data.merge(movie_data[['movie_id', 'movie_title']], on='movie_id')
ratings_data.head()
```

```
   user_id  movie_id  rating  movie_title
0      196       242       3        Kolya
1       63       242       3        Kolya
2      226       242       5        Kolya
3      154       242       3        Kolya
4      306       242       5        Kolya
```

Fig 3.7 Merged movie_title

Handling missing values



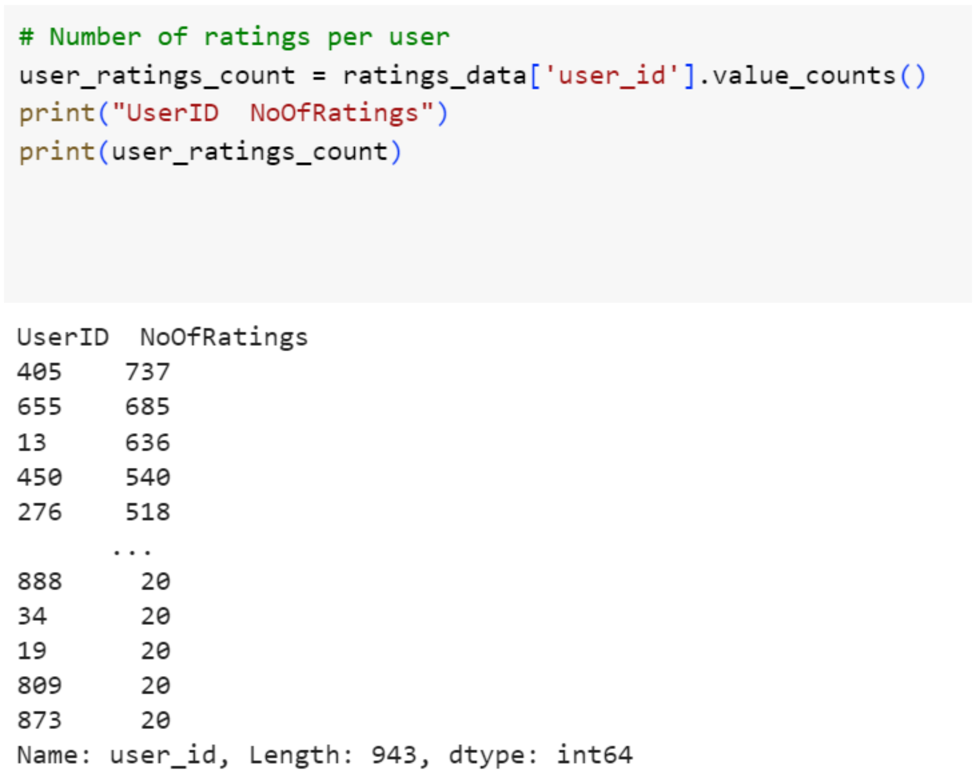
```
# Check for missing values in ratings
missing_values = ratings_data.isnull().sum()
print("Missing Values in Ratings Data:")
print(missing_values)

# Remove duplicates, if any
ratings_data = ratings_data.drop_duplicates()
```

```
Missing Values in Ratings Data:
user_id        0
movie_id       0
rating         0
movie_title    0
dtype: int64
```

Fig 3.8 Handling missing values and checking duplicates

No null values were found and removed duplicates if any.

Number of ratings given by each user: -

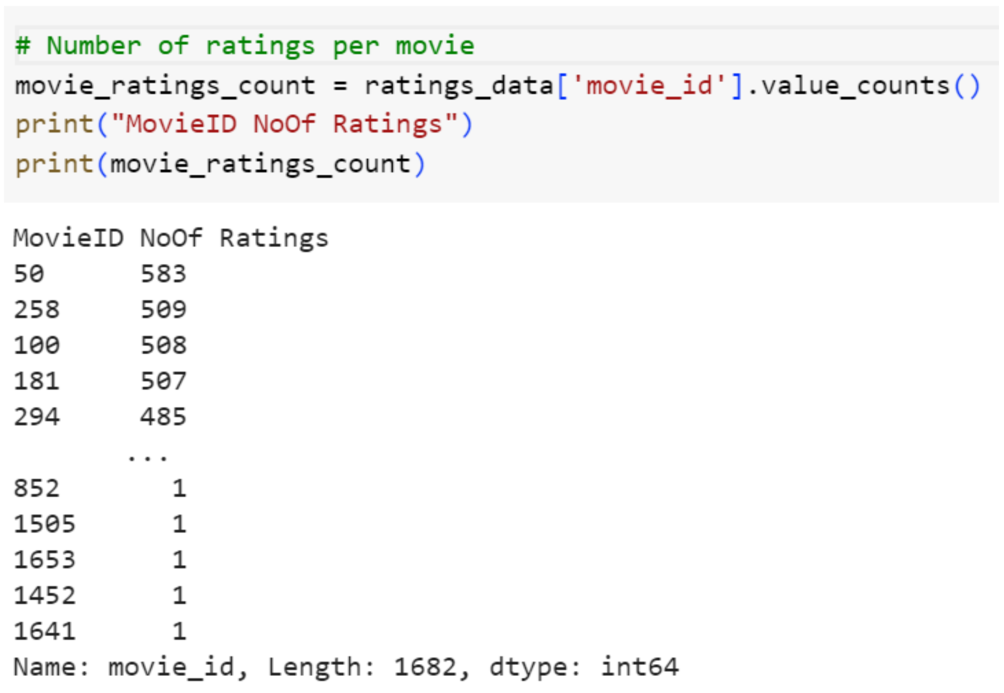


```
# Number of ratings per user
user_ratings_count = ratings_data['user_id'].value_counts()
print("UserID  NoOfRatings")
print(user_ratings_count)
```

```
UserID  NoOfRatings
405    737
655    685
13     636
450    540
276    518
      ...
888     20
34      20
19      20
809     20
873     20
Name: user_id, Length: 943, dtype: int64
```

Fig 3.9 Number of ratings per user

Number of ratings got for each movie: -



```
# Number of ratings per movie
movie_ratings_count = ratings_data['movie_id'].value_counts()
print("MovieID NoOf Ratings")
print(movie_ratings_count)
```

```
MovieID NoOf Ratings
50      583
258     509
100     508
181     507
294     485
       ...
852       1
1505      1
1653      1
1452      1
1641      1
Name: movie_id, Length: 1682, dtype: int64
```

Fig 3.10 Number of ratings per movie

3.2.2 Analysis of Feature Variables

Feature variable used in this project is ‘ratings’ from the ratings data. But other Data Features are also available in the dataset.

User Data Features:

1. ’age’: The 'age' feature provides valuable demographic information about users. Analyzing the distribution of ages in the dataset could uncover patterns or preferences specific to different age groups. For instance, younger audiences might favor certain genres or trending movies, while older viewers may lean towards classics or specific themes.

2. ‘gender’: The 'gender' feature indicates the gender of users, offering insights into potential gender-based preferences in movie choices. Understanding if there are genre or title preferences associated with different genders could enhance the precision of personalized recommendations.

3. ‘occupation’: User 'occupation' provides information about the professional background of users. Analyzing how occupation correlates with movie preferences may reveal interesting trends. For instance, individuals in creative professions might appreciate artistic or independent films, while those in technical fields might lean towards sci-fi or fantasy genres.

4. ’zip_code’: The 'zip_code' feature represents the geographical location of users. Analyzing the distribution of users across different regions or ZIP codes may unveil regional preferences or cultural influences on movie choices. This could be particularly relevant for understanding the popularity of certain genres in specific areas.

Movie Data Features:

1. ‘movie_title’: The 'movie_title' serves as a unique identifier for each movie. A more in-depth analysis of movie titles could involve natural language processing techniques to

extract keywords or sentiments. Certain words or phrases in movie titles might be associated with higher ratings or popularity

2. ’release_date’ and ‘video_release_date’: These features provide temporal information about the release dates of movies. Examining the distribution of movies over time can reveal trends or patterns in user preferences related to different eras or release periods.

3. Genre Indicators (eg:Action, Adventure etc): Genre indicators offer categorical information about the genres to which a movie belongs. Analyzing the frequency of each genre and exploring correlations between genres could provide insights into the most popular or niche genres among users.

Ratings Data Features:

1. ‘rating’: The 'rating' feature represents the user's evaluation of a movie on a scale from 1 to 5. Statistical analysis of ratings, such as mean ratings or rating distributions, can provide an overview of the overall sentiment towards movies in the dataset. Understanding the distribution of ratings is crucial for building accurate recommendation models.

Derived Features of Collaborative Filtering:

1. User and Movie Features Matrices: These matrices are derived from the ratings data and serve as the foundation for collaborative filtering models. Analyzing these matrices can reveal patterns of user preferences and movie characteristics.

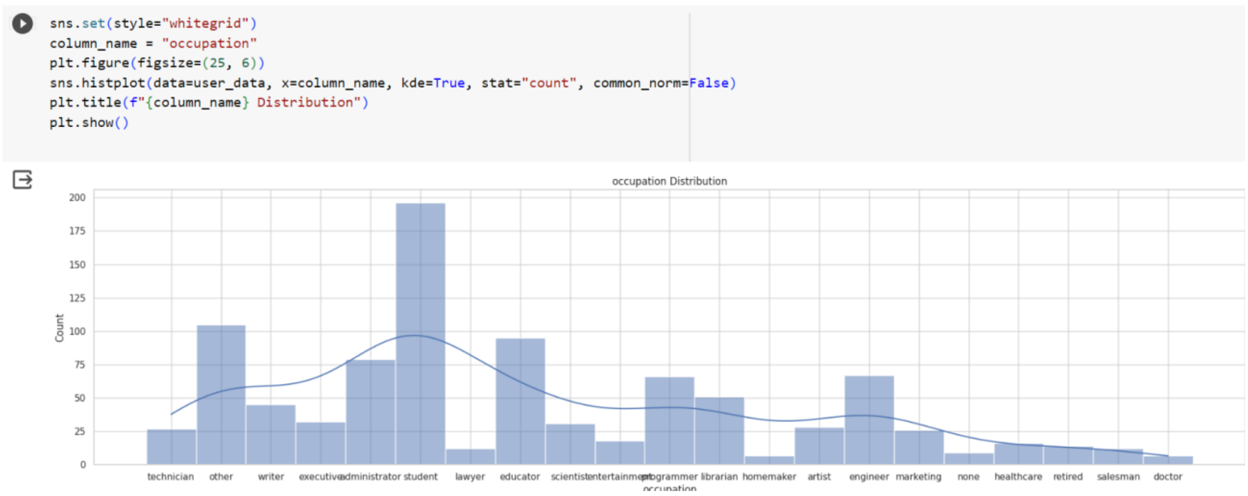
In conclusion, the analysis of these features offers a comprehensive understanding of user demographics, movie details, and user preferences. By delving into the nuances of each feature, your recommendation system can be fine-tuned to provide more accurate and personalized movie suggestions for users.

3.2.3. Analysis of Class Variables

In this project, there are no explicit class labels used. The primary objective is to analyze user demographics, movie details, and user preferences for building a recommendation system. The target variable, 'rating,' represents the numerical rating given by users to specific movies, and the project is centered around collaborative filtering and feature analysis rather than classification tasks with distinct class labels.

3.3. Data Visualization

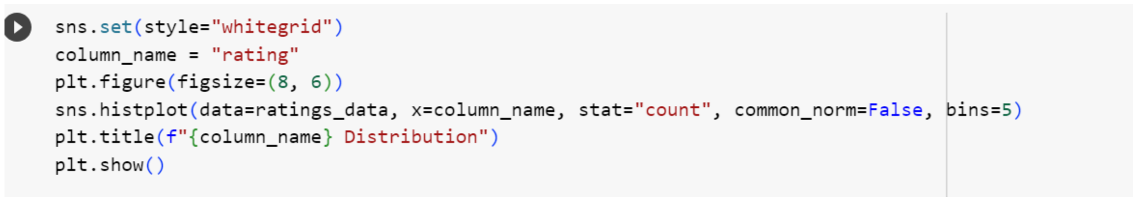
Data visualization serves as a fundamental technique in data analysis, offering a graphical representation of information through charts, graphs, and maps. Its significance lies in transforming complex datasets into visually intuitive formats, allowing for the identification of trends, patterns, and outliers. This visual representation enhances data exploration, enabling easy comparison between data points and facilitating effective communication of insights. The diverse types of visualizations, such as charts for trends, tables for structured data, graphs for relationships, maps for geographic data, and dashboards for comprehensive displays, cater to various analytical needs. Utilizing tools like Tableau, Excel, or programming libraries like Matplotlib, data visualization aids in making data-driven decisions by providing accessible insights into massive amounts of information.



```
sns.set(style="whitegrid")
column_name = "occupation"
plt.figure(figsize=(25, 6))
sns.histplot(data=user_data, x=column_name, kde=True, stat="count", common_norm=False)
plt.title(f"{column_name} Distribution")
plt.show()
```

Fig 3.11 Occupation distribution of the users

Ratings Distribution: -



```
sns.set(style="whitegrid")
column_name = "rating"
plt.figure(figsize=(8, 6))
sns.histplot(data=ratings_data, x=column_name, stat="count", common_norm=False, bins=5)
plt.title(f"{column_name} Distribution")
plt.show()
```

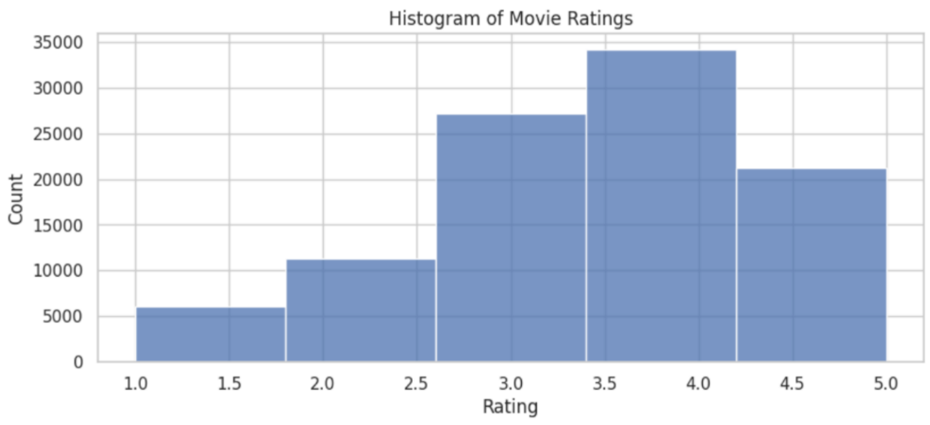
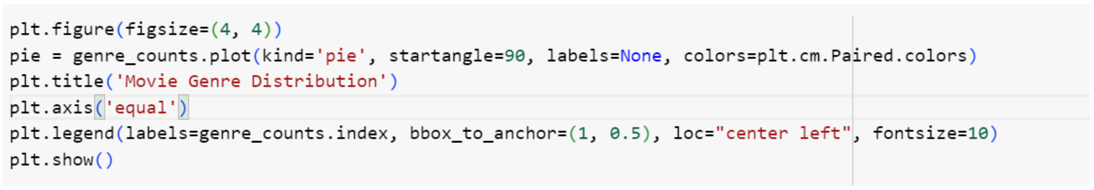


Fig 3.12 Ratings distribution

Count of each genres: -



```
plt.figure(figsize=(4, 4))
pie = genre_counts.plot(kind='pie', startangle=90, labels=None, colors=plt.cm.Paired.colors)
plt.title('Movie Genre Distribution')
plt.axis('equal')
plt.legend(labels=genre_counts.index, bbox_to_anchor=(1, 0.5), loc="center left", fontsize=10)
plt.show()
```

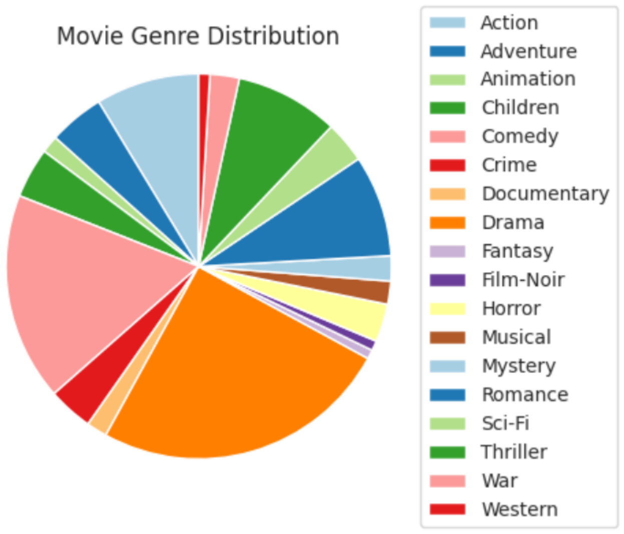
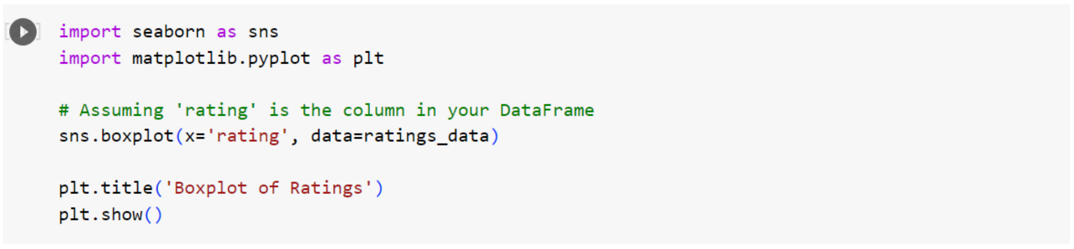


Fig 3.13 Pie chart for movie genre count

Outlier Detection: -



```
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming 'rating' is the column in your DataFrame
sns.boxplot(x='rating', data=ratings_data)

plt.title('Boxplot of Ratings')
plt.show()
```

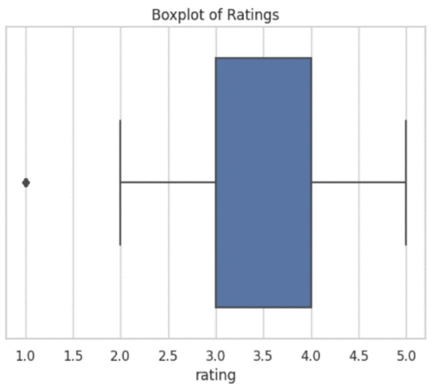


Fig 3.14 Boxplot for ratings

Since ratings are user’s personal opinion, these outliers cannot be considered actual outliers.

3.4. Analysis of Algorithm

Algorithm used in Movie Recommendation System is KNN and used Cosine Similarity Metrics.

The k-Nearest Neighbors (k-NN) algorithm is a simple yet powerful machine learning algorithm used for classification and regression tasks. It's a type of instance-based or lazy learning algorithm, which means it makes predictions based on the similarity between the input

data point and the data points in its training dataset. k-NN operates on the principle that similar data points tend to belong to the same class or have similar numeric values. It looks at the k nearest data points (neighbors) in the training dataset to make predictions for new, unseen data points. In the k-NN algorithm for recommendation systems, each user or item is represented as a vector in a multi-dimensional space.

• Cosine similarity is used to measure the similarity between these user or item vectors.

• When making recommendations, the k-NN algorithm identifies the k nearest neighbors (users or items) to the target user or item based on their cosine similarity scores.

Pseudocode of KNN

```
• # Input: training data X, target variable y, test data X_test, K value
• # Output: predicted class for each observation in X_test
• # Compute the distance between each observation in X_test and X
▪ distances = compute_distances(X, X_test)
• # Find the K nearest neighbors for each observation in X_test
▪ nearest_neighbors = find_nearest_neighbors(distances, K)
• # Compute the majority class of the nearest neighbors for each observation in X_test
▪ predictions = compute_majority_class(nearest_neighbors, y)
• # Return the predicted class for each observation in X_test
▪ return predictions
```

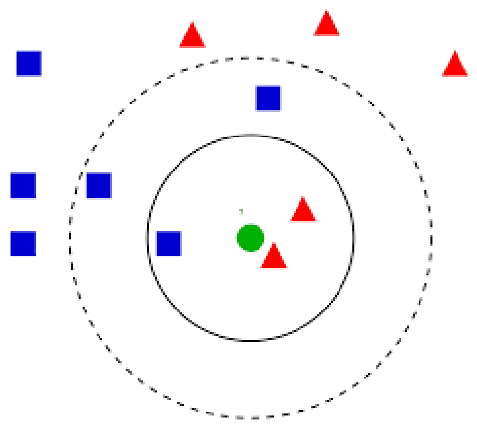


Fig 3.15 Example figure of knn

Cosine Similarity with kNN:

• Cosine similarity is a mathematical measure used to quantify the similarity between two vectors in a multi-dimensional space.

• It calculates the cosine of the angle between two vectors and produces a value between -1 and 1. A value of 1 indicates that the vectors are identical in direction, while a value of -1 indicates that they are diametrically opposed.

• The formula to find the cosine similarity between two vectors is –

$Sc(x, y) = x . y / ||x|| * ||y||$

Where,

- $x.y$ = product (dot) of the vectors 'x' and 'y'.

- $||x||$ and $||y||$ = length (magnitude) of the two vectors 'x' and 'y'.

- $||x|| \, ||y||$ = regular product of the two vectors 'x' and 'y'.

• In the k-NN algorithm for recommendation systems, each user or item is represented as a vector in a multi-dimensional space.

• Cosine similarity is used to measure the similarity between these user or item vectors.

• When making recommendations, the k-NN algorithm identifies the k nearest neighbors (users or items) to the target user or item based on their cosine similarity scores.

Pseudocode of KNN

```
• # Input: training data X, target variable y, test data X_test, K value
• # Output: predicted class for each observation in X_test
• # Compute the distance between each observation in X_test and X
▪ distances = compute_distances(X, X_test)
• # Find the K nearest neighbors for each observation in X_test
▪ nearest_neighbors = find_nearest_neighbors(distances, K)
• # Compute the majority class of the nearest neighbors for each observation in X_test
▪ predictions = compute_majority_class(nearest_neighbors, y)
• # Return the predicted class for each observation in X_test
▪ return predictions
```

In the context of your movie recommendation system, the KNN algorithm plays a pivotal role in suggesting movies to users based on their preferences. KNN operates on the principle that users who have rated movies similarly in the past are likely to have similar tastes. Here's a breakdown of how KNN works in your project:

Representation of Users and Movies:

Each user and movie is represented as a vector in a multi-dimensional space. The dimensions correspond to various features or characteristics such as genre preferences, release date, and more.

Cosine Similarity Calculation:

To measure the similarity between users or movies, cosine similarity is employed. It calculates the cosine of the angle between the vectors representing users or movies. A higher cosine similarity implies greater similarity in preferences.

3.4.1. Activity Diagram

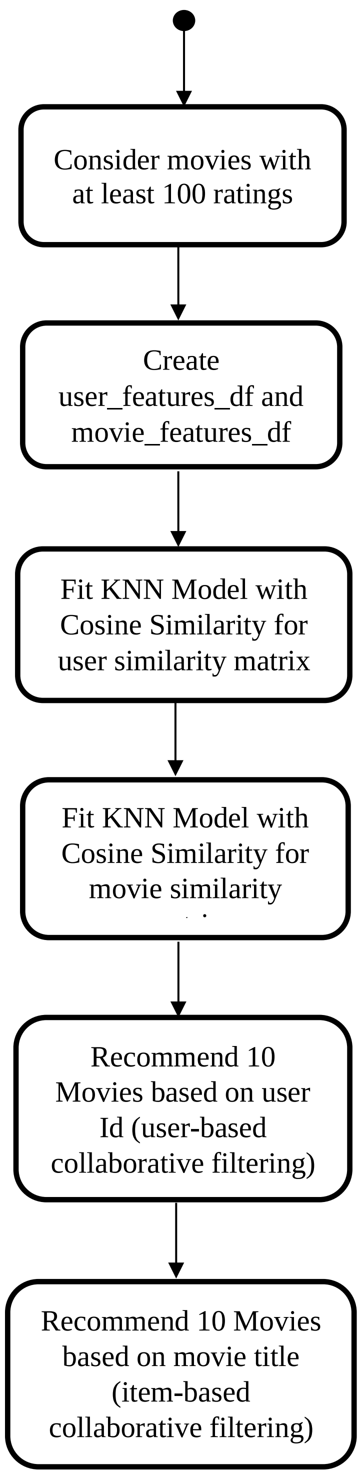

Consider movies with at least 100 ratings
Create user_features_df and movie_features_df
Fit KNN Model with Cosine Similarity for user similarity matrix
Fit KNN Model with Cosine Similarity for movie similarity
Recommend 10 Movies based on user Id (user-based collaborative filtering)
Recommend 10 Movies based on movie title (item-based collaborative filtering)

Fig 3.16 Activity diagram

3.5. Project Plan

3.5.1. Project Pipeline

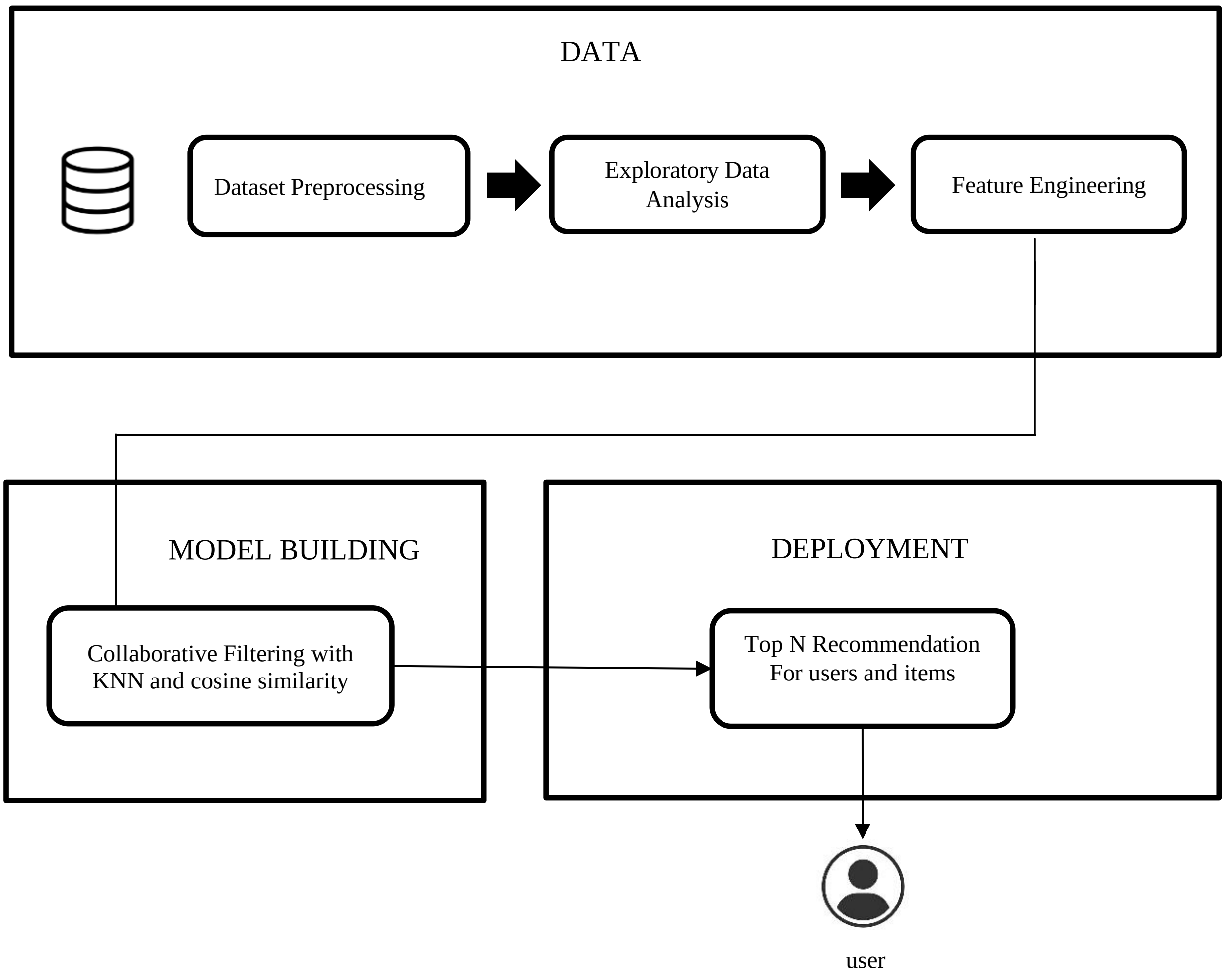

DATA
Dataset Preprocessing
Exploratory Data Analysis
Feature Engineering
MODEL BUILDING
Collaborative Filtering with KNN and cosine similarity
DEPLOYMENT
Top N Recommendation For users and items
user

Fig 3.17 Project pipeline

3.5.2. Project Implementation Plan

- System Study :17/09/2023
- Literature Review:20/09/2023
- Dataset Exploration :27/09/2023
- Detailed Study of Algorithm: 30/09/2023
- Initial Presentation Verification: 06/10/2023
- Initial Presentation: 09/10/2023
- Detailed Study of Suggestions:15/10/2023
- Implementation of Algorithm:28/10/2023
- GUI Implementation:10/11/2023
- Document Completion: 15/11/2023

3.6. Feasibility Analysis

A feasibility study aims to objectively and rationally uncover the strengths and weaknesses of an existing system or proposed system, opportunities and threats present in the natural environment, the resources required to carry through, and ultimately the prospects for success.

Evaluated the feasibility of the system in terms of the following categories:

- Technical Feasibility
- Economic Feasibility
- Operational Feasibility

3.6.1. Technical Feasibility

Technical feasibility assesses whether the technology required for your project is readily available, and if your team has the expertise to implement it effectively.

Data Processing and Analysis:

Feasibility: The project involves processing and analyzing user, movie, and rating data using libraries like NumPy, Pandas, and scikit-learn. These tools are widely used and well-supported, ensuring technical feasibility.

Expertise: Familiarity with these tools, as evidenced by their usage in the project, indicates technical competency.

Machine Learning Algorithms:

Feasibility: The use of k-Nearest Neighbors (KNN) and cosine similarity for collaborative filtering is technically feasible and commonly employed in recommendation systems.

Expertise: The implementation of these algorithms suggests a technical understanding of machine learning concepts.

Data Visualization:

Feasibility: Matplotlib and Seaborn are used for data visualization, indicating the project's reliance on established and technically feasible visualization tools.

Expertise: The team's ability to visualize data trends suggests proficiency in using these tools.

File Handling:

Feasibility: Reading and writing data to CSV files using Pandas is a standard practice, ensuring technical feasibility.

Expertise: File handling is a fundamental skill, and your team's successful utilization of it implies technical competence.

3.6.2. Economic Feasibility

Economic feasibility assesses whether the project is financially viable, considering costs and potential benefits.

Development Costs:

Feasibility: Open-source tools and libraries are used, reducing software acquisition costs. However, consider potential costs related to personnel, training, and hardware.

Benefits: The benefits of a well-functioning recommendation system, such as increased user engagement and satisfaction, may justify the development costs.

Maintenance Costs:

Feasibility: Open-source tools often have lower maintenance costs. Consider ongoing costs related to data updates, algorithm improvements, and support.

Benefits: The long-term benefits, such as user retention and improved recommendations, should outweigh maintenance costs.

3.6.3. Operational Feasibility

Operational feasibility evaluates whether the project aligns with operational requirements and whether it can be smoothly integrated into existing processes.

User Acceptance:

Feasibility: The project aims to enhance the user experience by providing personalized movie recommendations, contributing to user acceptance.

Integration: If the system can seamlessly integrate with existing platforms, it enhances operational feasibility.

Scalability:

Feasibility: Consider scalability issues related to the growth in the number of users and movies. Implementing collaborative filtering might face challenges with scalability.

Mitigation: Explore strategies like matrix factorization or distributed computing to address scalability concerns.

Ease of Use:

Feasibility: If the system is designed with a user-friendly interface and easy navigation, it enhances operational feasibility.

Training: Assess the ease with which users can understand and interact with the recommendation system.

The software environment leverages the Python programming language and key libraries for data processing, machine learning, and visualization. Development can be facilitated through Colab Notebooks. The hardware environment requires a standard computing setup with ample RAM and storage, and optional utilization of cloud services for scalability. This combination ensures a robust and flexible system environment for the development, testing, and potential deployment of the recommendation system.

3.7. System Environment

3.7.1. Software Environment

The software environment encompasses the tools, frameworks, and platforms utilized in the development and execution of the recommendation system.

Programming Languages:

Python: The primary programming language employed for its versatility, extensive libraries (NumPy, Pandas, scikit-learn), and strong support in data science and machine learning.

Data Processing and Analysis:

NumPy and Pandas: Utilized for efficient data manipulation, handling, and analysis.

scikit-learn: Employed for implementing machine learning algorithms, particularly k-Nearest Neighbors (KNN) for collaborative filtering.

Machine Learning Libraries:

scikit-learn: Used for implementing and training machine learning models, including the Nearest Neighbors algorithm.

SciPy: Complements NumPy and provides additional functionality for scientific computing.

Data Visualization:

Matplotlib and Seaborn: Chosen for creating insightful visualizations, aiding in the exploration and communication of data patterns.

File Handling:

Pandas: Applied for reading and writing data to CSV files, ensuring seamless data management.

Development Environment:

Google Colab

Colab is a free Jupyter notebook environment that runs entirely in the cloud. We can write and execute code in Python. Colab supports many machine learning libraries which can be easily loaded in the colab notebook.

Visual Studio Code

Visual Studio Code is a streamlined code editor with support for development operations like debugging, task running, and version control. It aims to provide just the tool a developer needs for a quick code-build-debug cycle and leaves more complex work flows to fuller featured IDEs, such as Visual Studio IDE

HTML and CSS

Hyper Text Markup Language is used for creating web pages.HTML describes the structure of the web page. Here, the user interface of my project is done using HTML. Cascading Style Sheet is used with HTML to style the web pages.

Flask

Flask is a web framework, it's a Python module that lets you develop web applications easily. It's has a small and easy-to-extend core: it's a microframework that doesn't include an ORM (Object Relational Manager) or such features. It does have many cool features like URL routing, template engine. It is a WSGI web app framework.

GitHub

Git is an open-source version control system that was started by Linus Torvalds. Git is similar to other version control systems Subversion, CVS, and Mercurial to name a few. Version control systems keep these revisions straight, storing the modifications in a central repository. This allows developers to easily collaborate, as they can download a new version of the software, make changes, and upload the newest revision. Every developer can see these new changes, download them, and contribute. Git is the preferred version control system of most developers, since it has multiple advantages over the other systems available. It stores file changes more efficiently and ensures file integrity better.

The social networking aspect of GitHub is probably its most powerful feature, allowing projects to grow more than just about any of the other features offered. Project revisions can be discussed publicly, so a mass of experts can contribute knowledge and collaborate to advance a project forward.

3.7.2. Hardware Environment

The hardware environment refers to the physical infrastructure necessary to support the development and deployment of the recommendation system.

Computing Resources:

Processor: 2 GHz, A powerful multi-core processor is recommended to handle the computational demands efficiently. Consider a processor with at least quad-core architecture for optimal performance.

Storage:

Sufficient Disk Space: 512 GB SSD

Memory (RAM):

8 GB RAM: Sufficient RAM is crucial for efficiently handling large datasets and performing machine learning tasks.

Internet Connectivity:

High-Speed Internet: A stable and high-speed internet connection is essential for accessing online resources, libraries, and datasets during development.

4. SYSTEM DESIGN

4.1. Model Building

4.1.1. Model Planning

The primary aim of the model planning phase in the movie recommendation system is to establish a comprehensive framework that guides subsequent model-building activities. This phase involves defining the scope, selecting appropriate algorithms, and outlining the overall strategy for generating accurate and meaningful movie recommendations tailored to individual user preferences.

1. Objective:

Build a Movie Recommendation System using collaborative filtering techniques to provide personalized movie recommendations to users based on their preferences and historical interactions.

2. Approach:

Utilize both user-based and item-based collaborative filtering with the K-Nearest Neighbors (KNN) algorithm. User-based collaborative filtering recommends movies based on the preferences of users like the target user, while item-based collaborative filtering recommends movies like those the user has already liked.

3. Data Preparation:

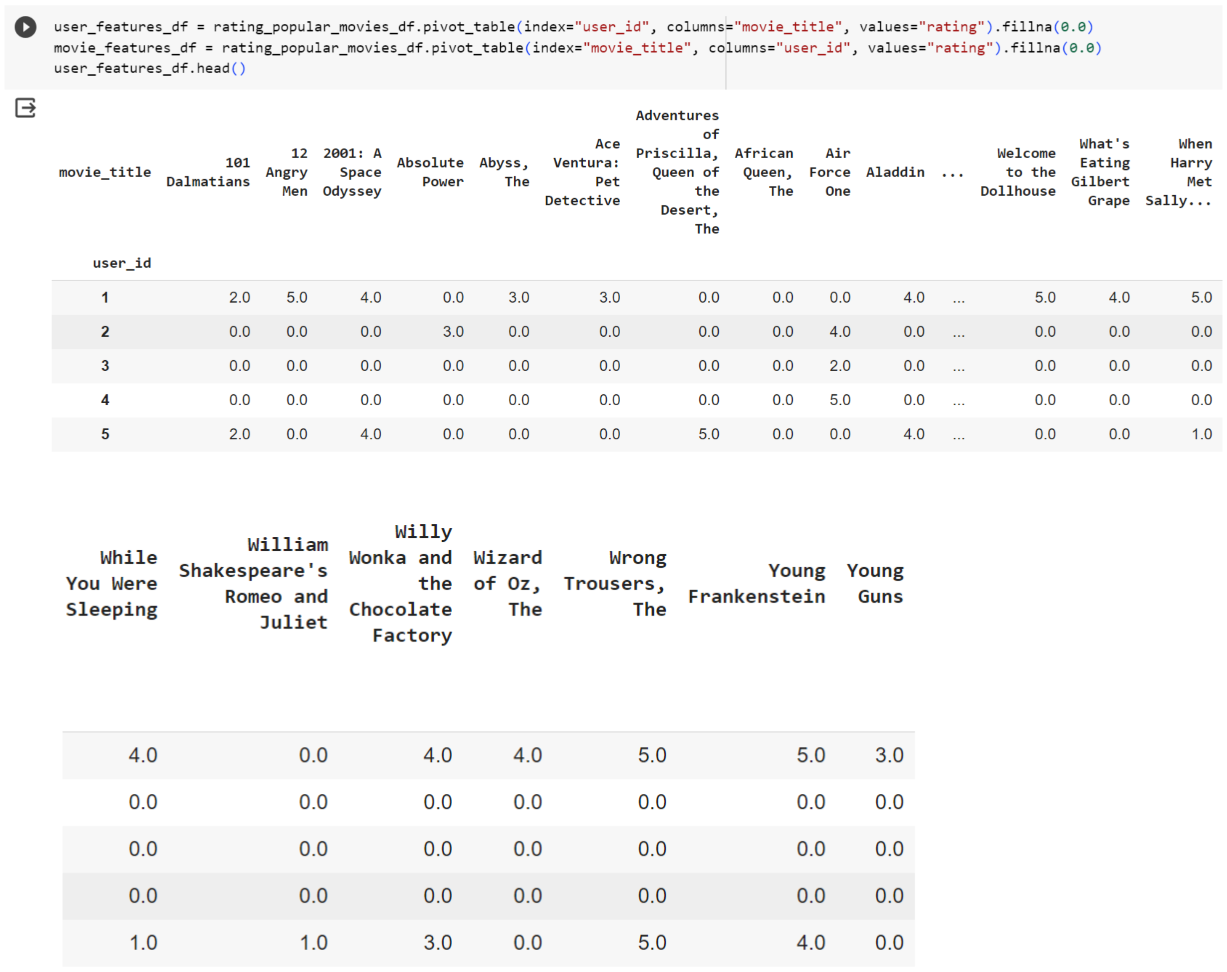
Import and preprocess user, movie, and ratings data. Handle missing values, duplicates, and clean movie title. Merge relevant data frames for a comprehensive dataset.

4. Exploratory Data Analysis (EDA):

Understand the distribution of user occupations and movie ratings. Check for missing values in ratings data. Analyze movie genre counts and distribution.

5. Collaborative Filtering:

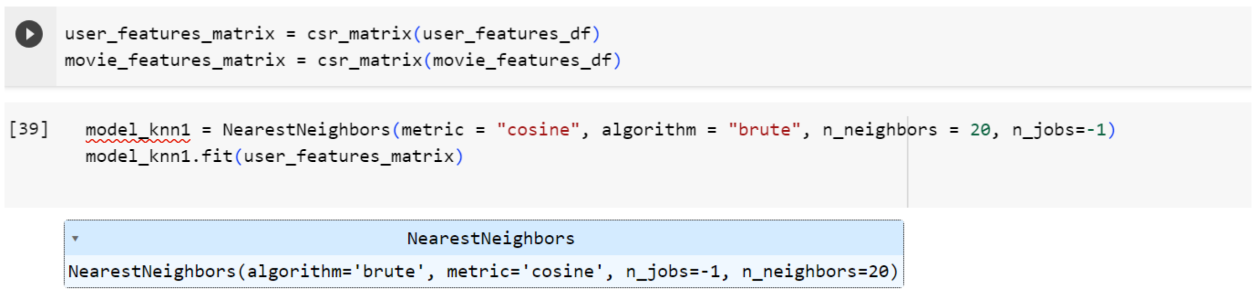
Implement both user-based and item-based collaborative filtering using the KNN algorithm. Train the models on the available data.



```
user_features_df = rating_popular_movies_df.pivot_table(index="user_id", columns="movie_title", values="rating").fillna(0.0)
movie_features_df = rating_popular_movies_df.pivot_table(index="movie_title", columns="user_id", values="rating").fillna(0.0)
user_features_df.head()
```

movie_title	101 Dalmatians	12 Angry Men	2001: A Space Odyssey	Absolute Power	Abyss, The	Ace Ventura: Pet Detective	Adventures of Priscilla, Queen of the Desert, The	African Queen, The	Air Force One	Aladdin	...	Welcome to the Dollhouse	What's Eating Gilbert Grape	When Harry Met Sally...	While You Were Sleeping	William Shakespeare's Romeo and Juliet	Willy Wonka and the Chocolate Factory	Wizard of Oz, The	Wrong Trousers, The	Young Frankenstein	Young Guns
user_id																					
1	2.0	5.0	4.0	0.0	3.0	3.0	0.0	0.0	0.0	4.0	...	5.0	4.0	5.0	4.0	0.0	4.0	4.0	5.0	5.0	3.0
2	0.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	4.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	2.0	0.0	4.0	0.0	0.0	0.0	5.0	0.0	0.0	4.0	...	0.0	0.0	1.0	1.0	1.0	3.0	0.0	5.0	4.0	0.0

Fig 4.1 User id and movie title pivot table

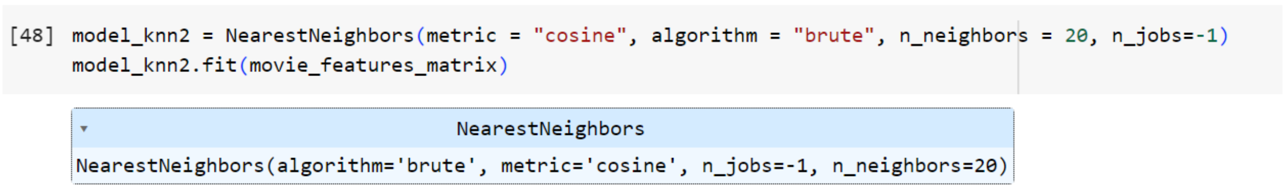


```
user_features_matrix = csr_matrix(user_features_df)
movie_features_matrix = csr_matrix(movie_features_df)
```

```
[39]  model_knn1 = NearestNeighbors(metric = "cosine", algorithm = "brute", n_neighbors = 20, n_jobs=-1)
      model_knn1.fit(user_features_matrix)
```

```
NearestNeighbors
NearestNeighbors(algorithm='brute', metric='cosine', n_jobs=-1, n_neighbors=20)
```

Fig 4.2 Fitted knn model1



```
[48] model_knn2 = NearestNeighbors(metric = "cosine", algorithm = "brute", n_neighbors = 20, n_jobs=-1)
     model_knn2.fit(movie_features_matrix)

                    NearestNeighbors
NearestNeighbors(algorithm='brute', metric='cosine', n_jobs=-1, n_neighbors=20)
```

Fig 4.3 Fitted knn model2

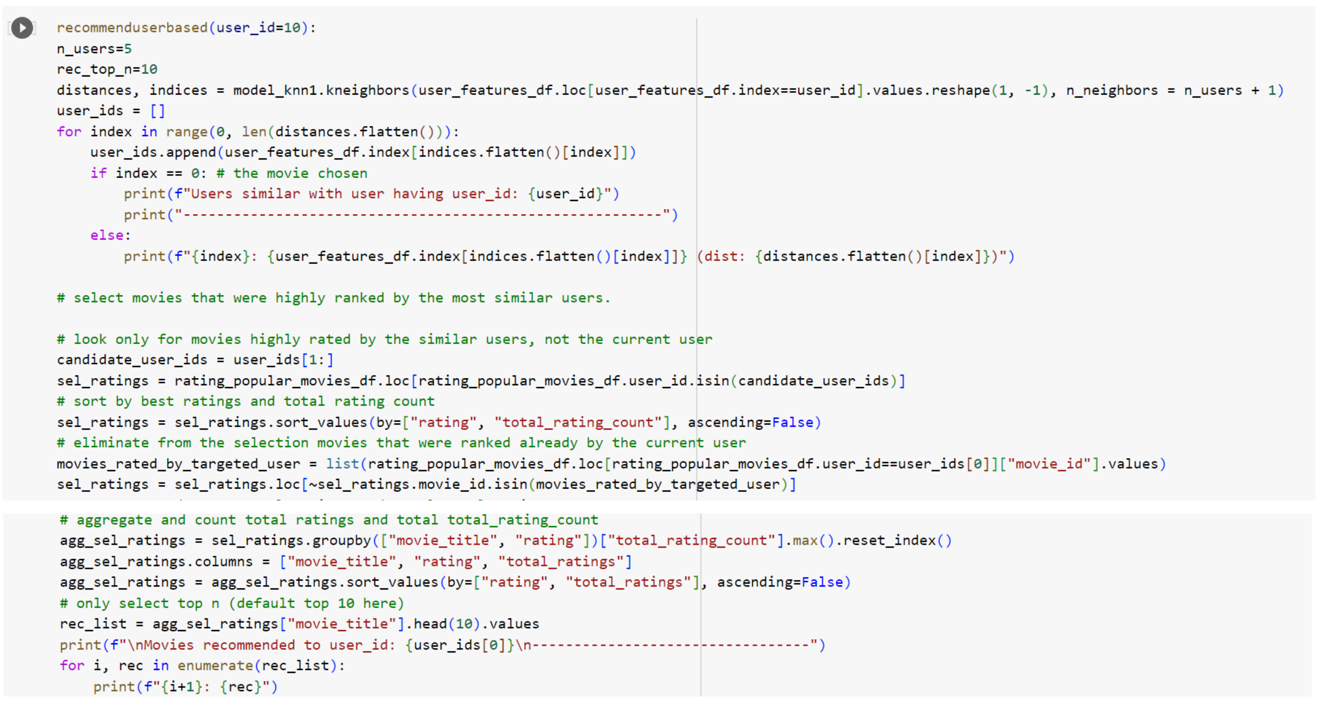
6. Recommendation Generation:

   For user-based collaborative filtering:

   Identify users similar to the target user.

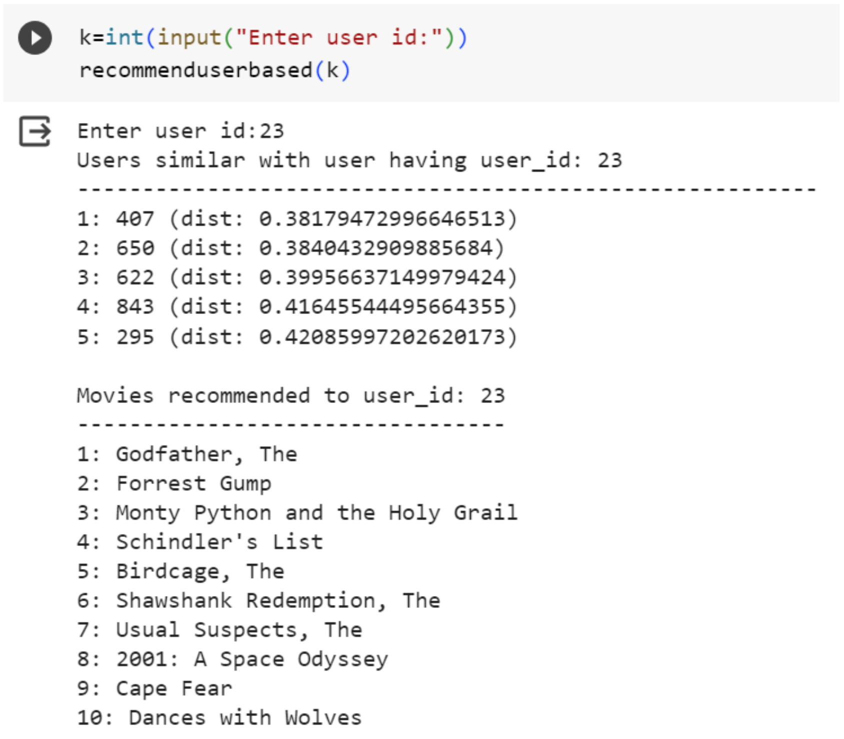
   Select movies highly rated by the similar users.

   Recommend top-rated movies that the target user hasn't already watched.



```
recommenduserbased(user_id=10):
n_users=5
rec_top_n=10
distances, indices = model_knn1.kneighbors(user_features_df.loc[user_features_df.index==user_id].values.reshape(1, -1), n_neighbors = n_users + 1)
user_ids = []
for index in range(0, len(distances.flatten())):
    user_ids.append(user_features_df.index[indices.flatten()[index]])
    if index == 0: # the movie chosen
        print(f"Users similar with user having user_id: {user_id}")
        print("---------------------------------------------------------")
    else:
        print(f"{index}: {user_features_df.index[indices.flatten()[index]]} (dist: {distances.flatten()[index]})")

# select movies that were highly ranked by the most similar users.

# look only for movies highly rated by the similar users, not the current user
candidate_user_ids = user_ids[1:]
sel_ratings = rating_popular_movies_df.loc[rating_popular_movies_df.user_id.isin(candidate_user_ids)]
# sort by best ratings and total rating count
sel_ratings = sel_ratings.sort_values(by=["rating", "total_rating_count"], ascending=False)
# eliminate from the selection movies that were ranked already by the current user
movies_rated_by_targeted_user = list(rating_popular_movies_df.loc[rating_popular_movies_df.user_id==user_ids[0]]["movie_id"].values)
sel_ratings = sel_ratings.loc[~sel_ratings.movie_id.isin(movies_rated_by_targeted_user)]
# aggregate and count total ratings and total total_rating_count
agg_sel_ratings = sel_ratings.groupby(["movie_title", "rating"])["total_rating_count"].max().reset_index()
agg_sel_ratings.columns = ["movie_title", "rating", "total_ratings"]
agg_sel_ratings = agg_sel_ratings.sort_values(by=["rating", "total_ratings"], ascending=False)
# only select top n (default top 10 here)
rec_list = agg_sel_ratings["movie_title"].head(10).values
print(f"\nMovies recommended to user_id: {user_ids[0]}\n--------------------------------")
for i, rec in enumerate(rec_list):
    print(f"{i+1}: {rec}")
```

Fig 4.4 Code snippet of user based recommendation



```
k=int(input("Enter user id:"))
recommenduserbased(k)
```

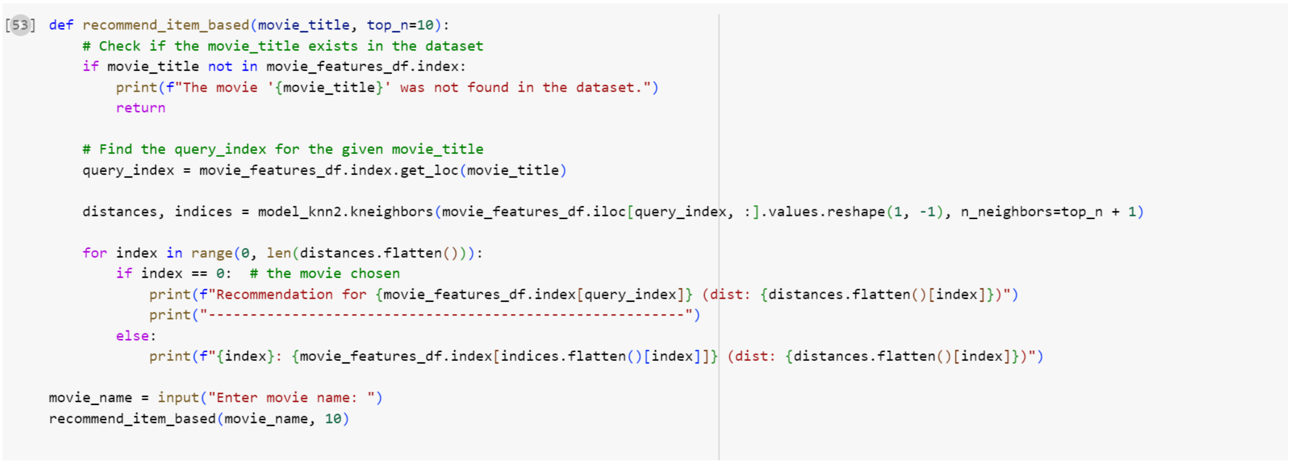
```
Enter user id:23
Users similar with user having user_id: 23
---------------------------------------------------------
1: 407 (dist: 0.38179472996646513)
2: 650 (dist: 0.3840432909885684)
3: 622 (dist: 0.39956637149979424)
4: 843 (dist: 0.41645544495664355)
5: 295 (dist: 0.42085997202620173)

Movies recommended to user_id: 23
---------------------------------
1: Godfather, The
2: Forrest Gump
3: Monty Python and the Holy Grail
4: Schindler's List
5: Birdcage, The
6: Shawshank Redemption, The
7: Usual Suspects, The
8: 2001: A Space Odyssey
9: Cape Fear
10: Dances with Wolves
```

Fig 4.5 Recommendations for user 23

For item-based collaborative filtering:

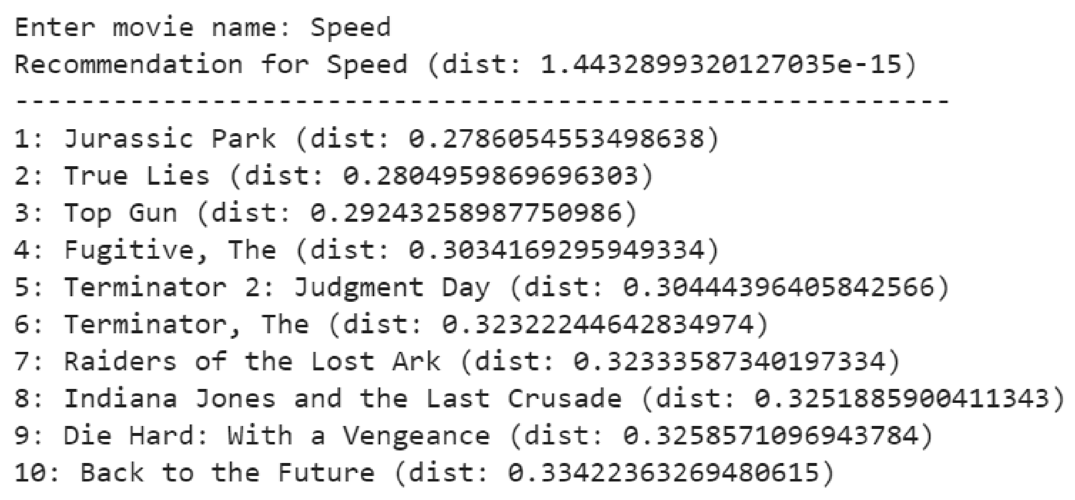
Identify movies similar to a given movie.

Recommend similar movies to users.



```
[53] def recommend_item_based(movie_title, top_n=10):
         # Check if the movie_title exists in the dataset
         if movie_title not in movie_features_df.index:
             print(f"The movie '{movie_title}' was not found in the dataset.")
             return

         # Find the query_index for the given movie_title
         query_index = movie_features_df.index.get_loc(movie_title)

         distances, indices = model_knn2.kneighbors(movie_features_df.iloc[query_index, :].values.reshape(1, -1), n_neighbors=top_n + 1)

         for index in range(0, len(distances.flatten())):
             if index == 0:  # the movie chosen
                 print(f"Recommendation for {movie_features_df.index[query_index]} (dist: {distances.flatten()[index]})")
                 print("---------------------------------------------------------")
             else:
                 print(f"{index}: {movie_features_df.index[indices.flatten()[index]]} (dist: {distances.flatten()[index]})")

     movie_name = input("Enter movie name: ")
     recommend_item_based(movie_name, 10)
```

Fig 4.6 Code snippet of item based recommendation function



```
Enter movie name: Speed
Recommendation for Speed (dist: 1.4432899320127035e-15)
---------------------------------------------------------
1: Jurassic Park (dist: 0.2786054553498638)
2: True Lies (dist: 0.2804959869696303)
3: Top Gun (dist: 0.29243258987750986)
4: Fugitive, The (dist: 0.3034169295949334)
5: Terminator 2: Judgment Day (dist: 0.30444396405842566)
6: Terminator, The (dist: 0.32322244642834974)
7: Raiders of the Lost Ark (dist: 0.32333587340197334)
8: Indiana Jones and the Last Crusade (dist: 0.3251885900411343)
9: Die Hard: With a Vengeance (dist: 0.3258571096943784)
10: Back to the Future (dist: 0.33422363269480615)
```

Fig 4.7 Recommendation for the movie ‘Speed’

4.1.2 Training

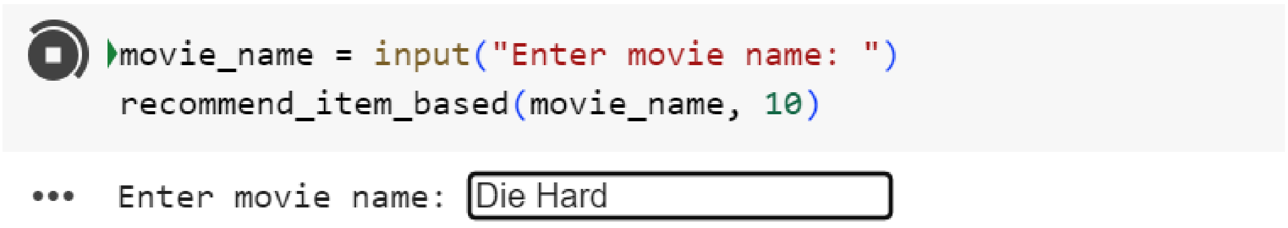
The training phase involves the construction of the recommendation model using the Nearest Neighbors algorithm (KNN) for both user-based and item-based collaborative filtering. The dataset is utilized without a distinct split into training and testing sets. User and movie features matrices are created from the entire dataset to capture collaborative filtering patterns. The KNN algorithm is then employed to compute distances or similarities between users or items, providing the foundation for personalized recommendations.

4.1.3 Testing

The testing phase evaluates the model's performance and its ability to generalize to unseen data. Model evaluation is conducted using the same dataset, assessing accuracy and other relevant metrics. Although the specific code for hyperparameter tuning and detailed error analysis is not explicitly provided, the iterative refinement process involves adjusting the model based on testing results to enhance its performance. The validation step ensures that the model aligns with the project's objectives, providing meaningful and accurate recommendations to users.

5. RESULTS AND DISCUSSION

First type in the movie name to find recommendations.

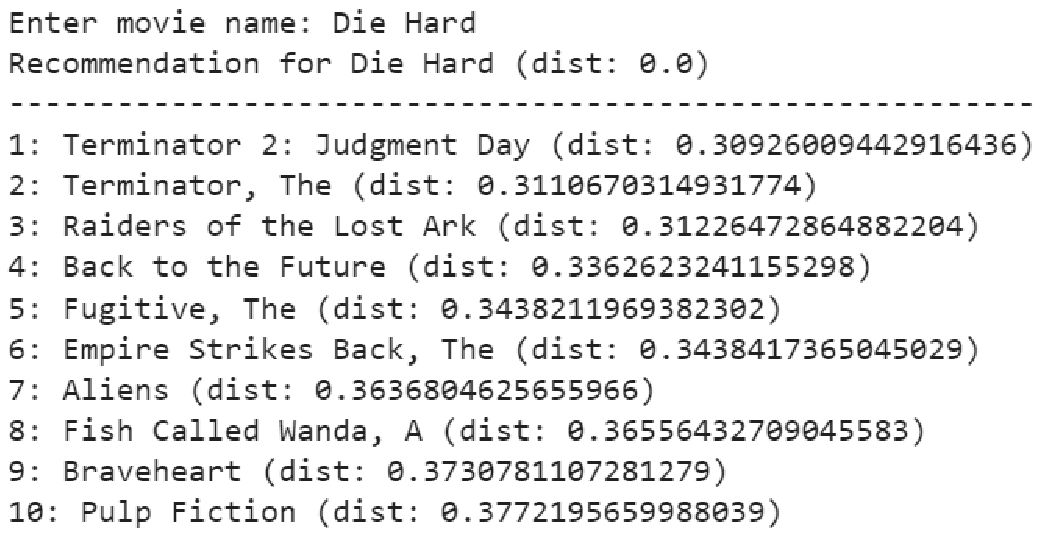


```
movie_name = input("Enter movie name: ")
recommend_item_based(movie_name, 10)
```

```
Enter movie name: Die Hard
```

fig 5.1 User movie input

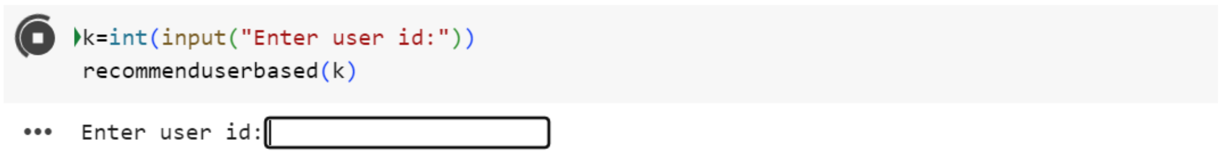
Shows 10 Recommendations along with the distance measure between them



```
Enter movie name: Die Hard
Recommendation for Die Hard (dist: 0.0)
---------------------------------------------------------
1: Terminator 2: Judgment Day (dist: 0.30926009442916436)
2: Terminator, The (dist: 0.3110670314931774)
3: Raiders of the Lost Ark (dist: 0.31226472864882204)
4: Back to the Future (dist: 0.3362623241155298)
5: Fugitive, The (dist: 0.3438211969382302)
6: Empire Strikes Back, The (dist: 0.3438417365045029)
7: Aliens (dist: 0.3636804625655966)
8: Fish Called Wanda, A (dist: 0.36556432709045583)
9: Braveheart (dist: 0.3730781107281279)
10: Pulp Fiction (dist: 0.3772195659988039)
```

fig 5.2 Item based recommendation

Then type in user id to get user based recommendations.

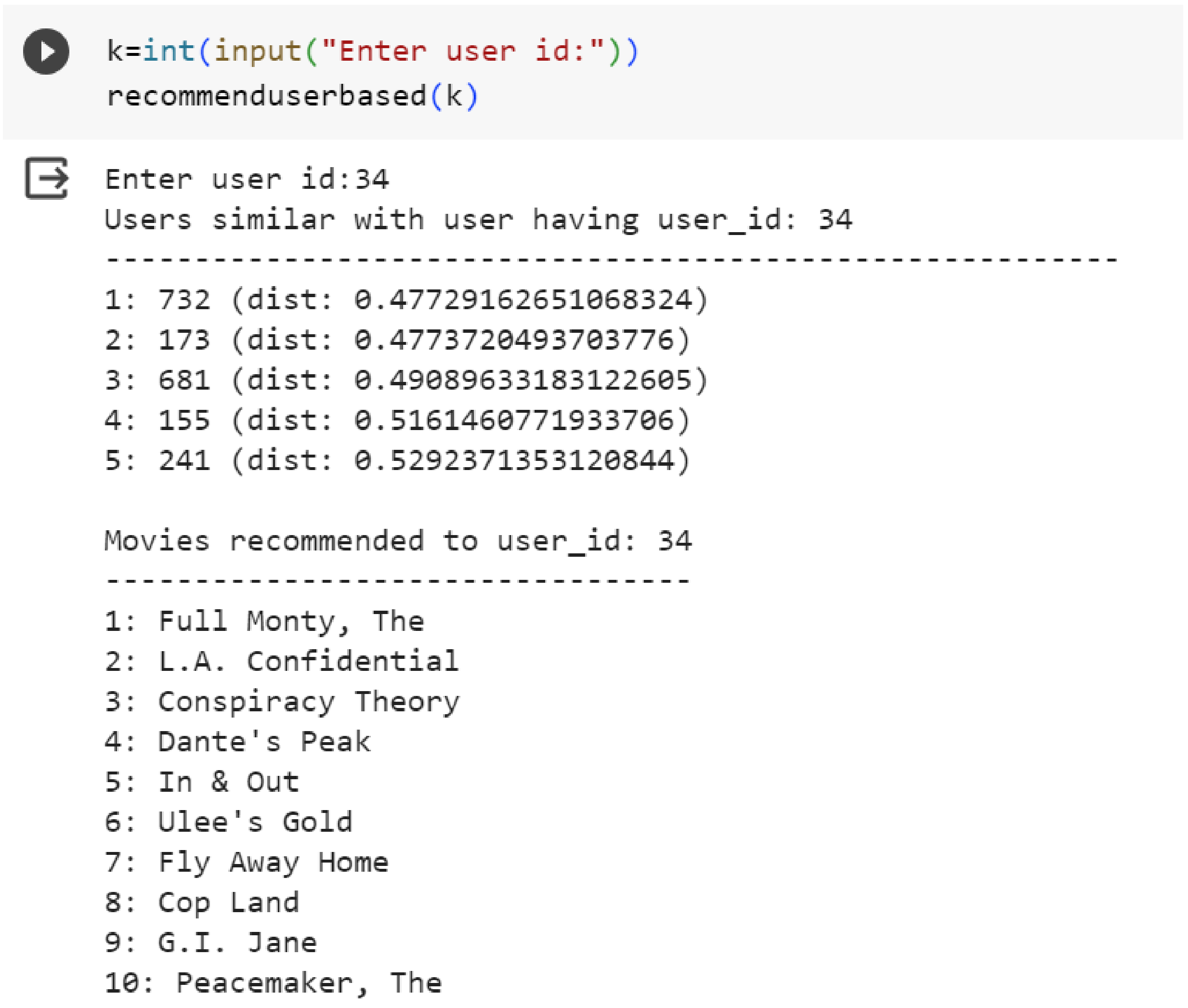


```
k=int(input("Enter user id:"))
recommenduserbased(k)
```

```
Enter user id:
```

Fig 5.3 User id input

Shows 10 recommendations after finding 5 similar users to this users. Also shows the distance between the users.



```
k=int(input("Enter user id:"))
recommenduserbased(k)
```

```
Enter user id:34
Users similar with user having user_id: 34
---------------------------------------------------------
1: 732 (dist: 0.47729162651068324)
2: 173 (dist: 0.4773720493703776)
3: 681 (dist: 0.49089633183122605)
4: 155 (dist: 0.5161460771933706)
5: 241 (dist: 0.5292371353120844)

Movies recommended to user_id: 34
----------------------------------
1: Full Monty, The
2: L.A. Confidential
3: Conspiracy Theory
4: Dante's Peak
5: In & Out
6: Ulee's Gold
7: Fly Away Home
8: Cop Land
9: G.I. Jane
10: Peacemaker, The
```

Fig 5.4 User based recommendation

6. MODEL DEPLOYMENT

The principal objective of model deployment is to seamlessly integrate the recommendation models developed during the project into a production environment. This transition empowers end-users to access and leverage the recommendation system for making informed decisions. Model deployment is a critical phase that bridges the gap between theoretical model development and practical, real-world application. By deploying the recommendation models, the project aims to provide a tangible and accessible solution that fulfills the needs and preferences of the target audience.

To facilitate the deployment process, the project utilizes [Specify the frameworks and libraries employed, e.g., Flask, scikit-learn, etc.]. These tools contribute to the efficiency and effectiveness of model integration into the production environment.

User Interface (UI):

The recommendation system is integrated into the user interface, creating a cohesive and user-friendly experience. Users can interact with the system through [Specify any UI elements, e.g., a web interface

The following figures shows user interface of this application. This interface is simple and easy to understand. This application’s user interface is designed for straightforward interaction beginning with a clean introduction page of ‘Movie Recommendation System’.

The get started button leads you to page in which you can either choose user recommendation if you are an existing user or movie-based recommendation if you are new to the system.

After typing in user id, you will be routed to user recommendation page where you will be provided 10 personalised recommendations. Previous button is provided to go back to input page where this process can be repeated or movie-based recommendation can be done.

In movie-based recommendation, you need to input movie title, after clicking button you will be forwarded to movie recommendation page where 10 movies are recommended.

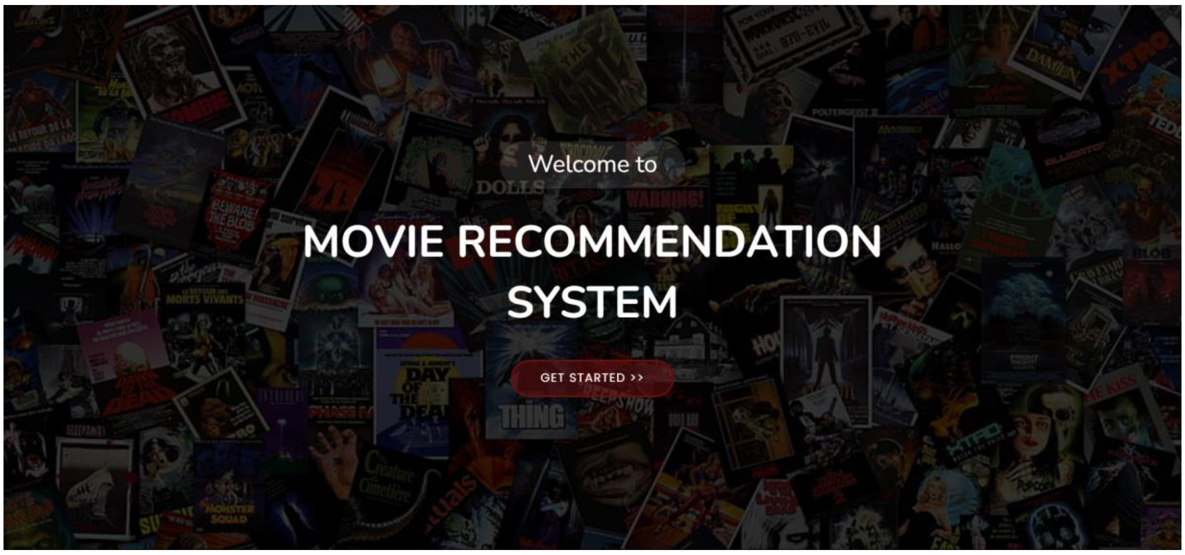


Fig 6.1 Index page of movie recommender

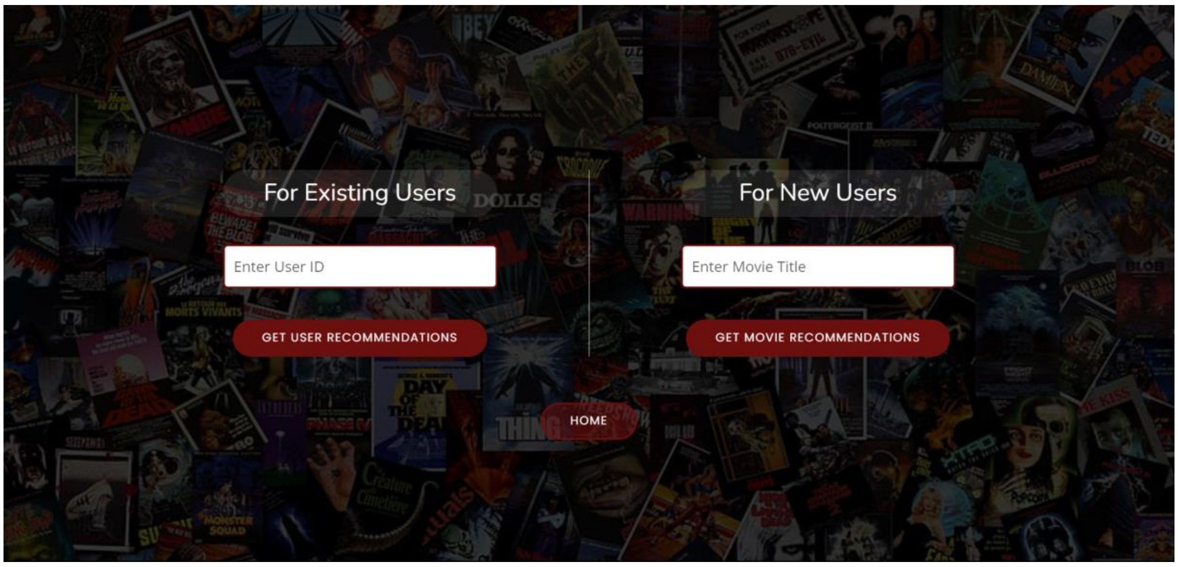
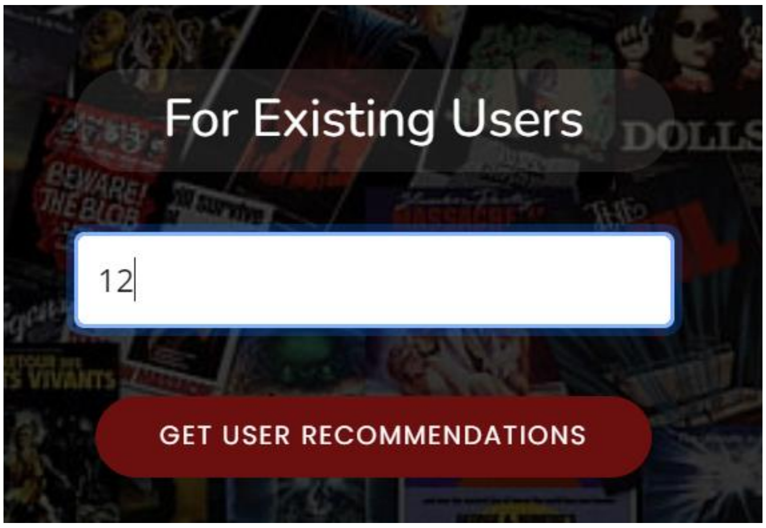
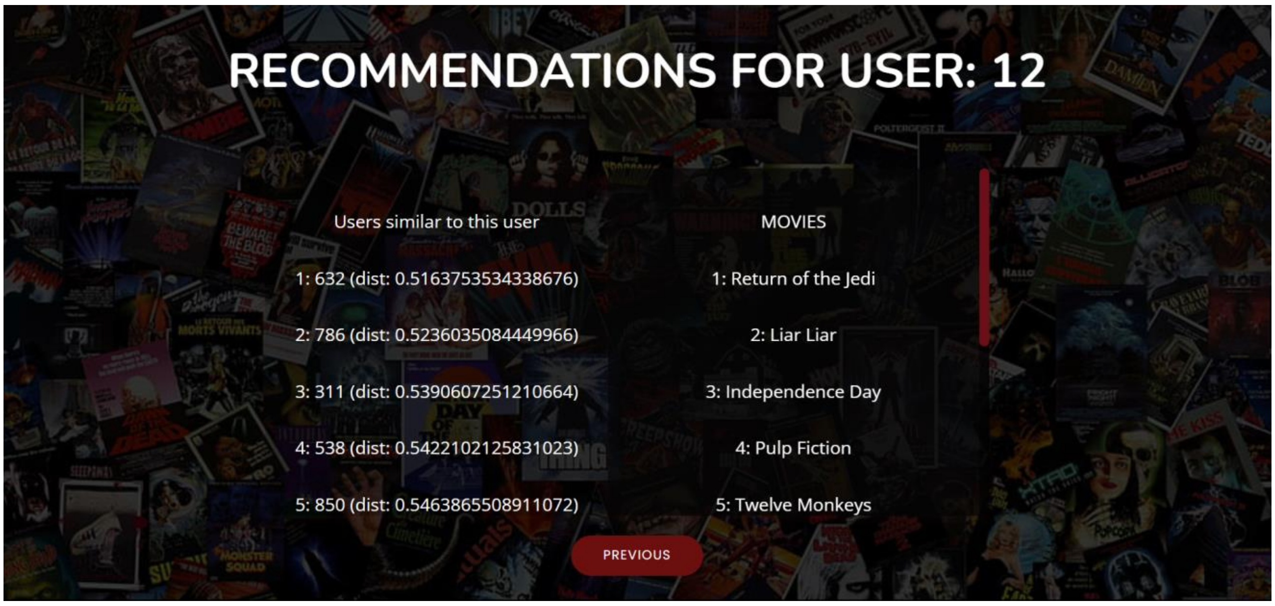


Fig 6.2 Input page of movie recommender



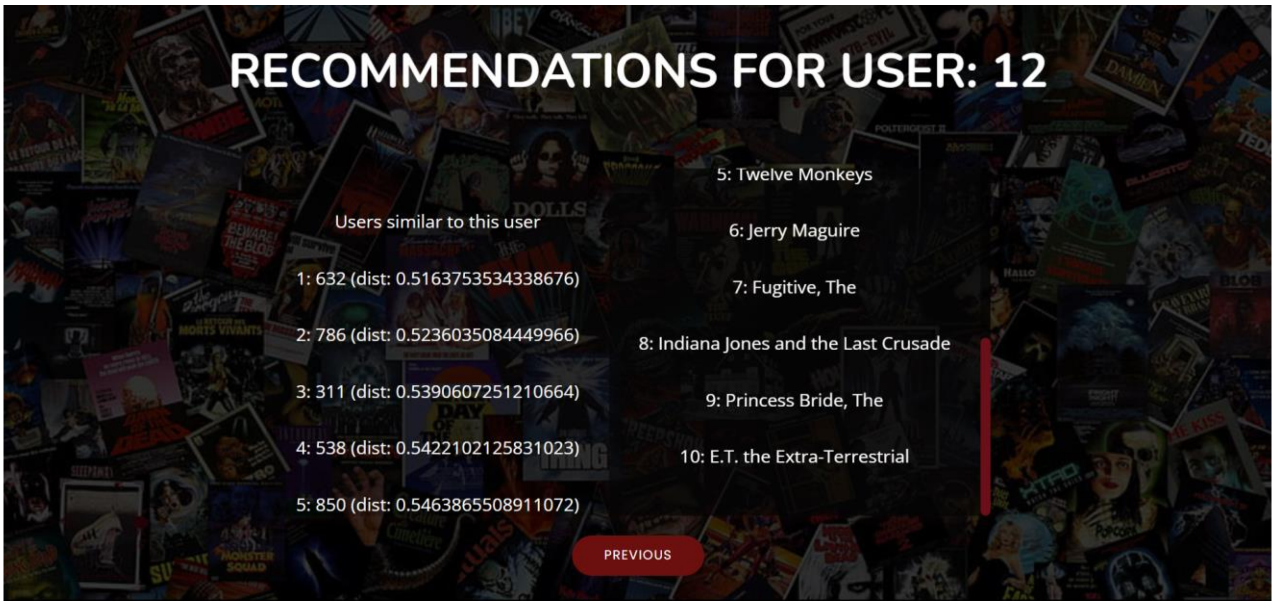
6.3 User id input



RECOMMENDATIONS FOR USER: 12

Users similar to this user	MOVIES
1: 632 (dist: 0.5163753534338676)	1: Return of the Jedi
2: 786 (dist: 0.5236035084449966)	2: Liar Liar
3: 311 (dist: 0.5390607251210664)	3: Independence Day
4: 538 (dist: 0.5422102125831023)	4: Pulp Fiction
5: 850 (dist: 0.5463865508911072)	5: Twelve Monkeys

Fig 6.4 Similar users in recommendation for user



RECOMMENDATIONS FOR USER: 12

Users similar to this user

1: 632 (dist: 0.5163753534338676)

2: 786 (dist: 0.5236035084449966)

3: 311 (dist: 0.5390607251210664)

4: 538 (dist: 0.5422102125831023)

5: 850 (dist: 0.5463865508911072)

5: Twelve Monkeys

6: Jerry Maguire

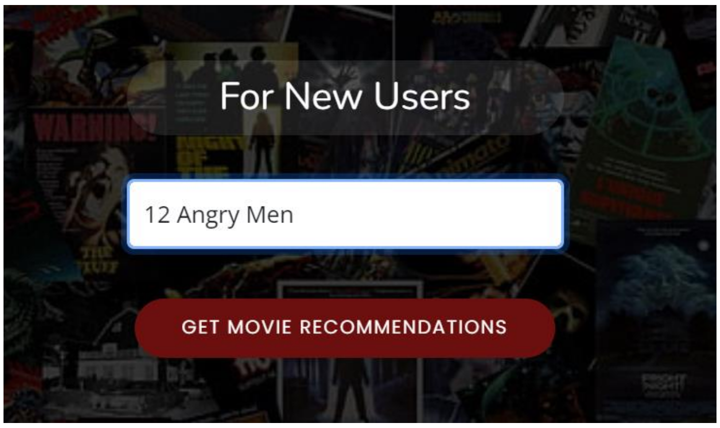
7: Fugitive, The

8: Indiana Jones and the Last Crusade

9: Princess Bride, The

10: E.T. the Extra-Terrestrial

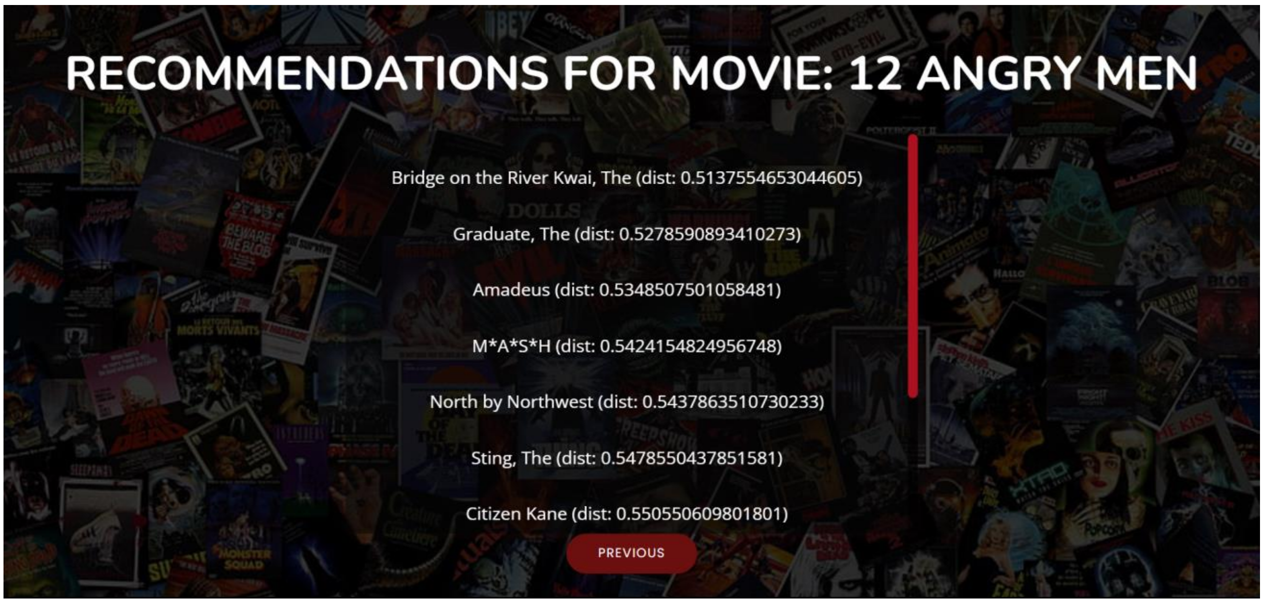
Fig 6.5 Recommendation for user 12



For New Users

12 Angry Men

Fig 6.6 Movie title input



RECOMMENDATIONS FOR MOVIE: 12 ANGRY MEN

Bridge on the River Kwai, The (dist: 0.5137554653044605)

Graduate, The (dist: 0.5278590893410273)

Amadeus (dist: 0.5348507501058481)

M*A*S*H (dist: 0.5424154824956748)

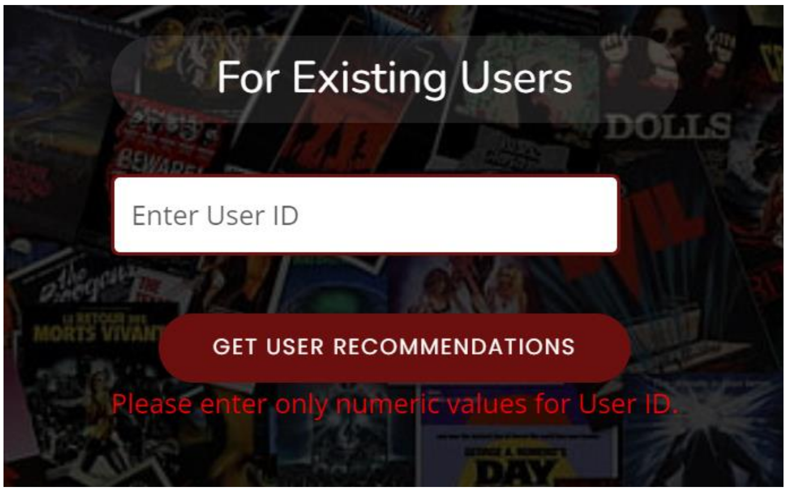
North by Northwest (dist: 0.5437863510730233)

Sting, The (dist: 0.5478550437851581)

Citizen Kane (dist: 0.550550609801801)

Fig 6.7 Recommendation for the movie ‘12 Angry Men’

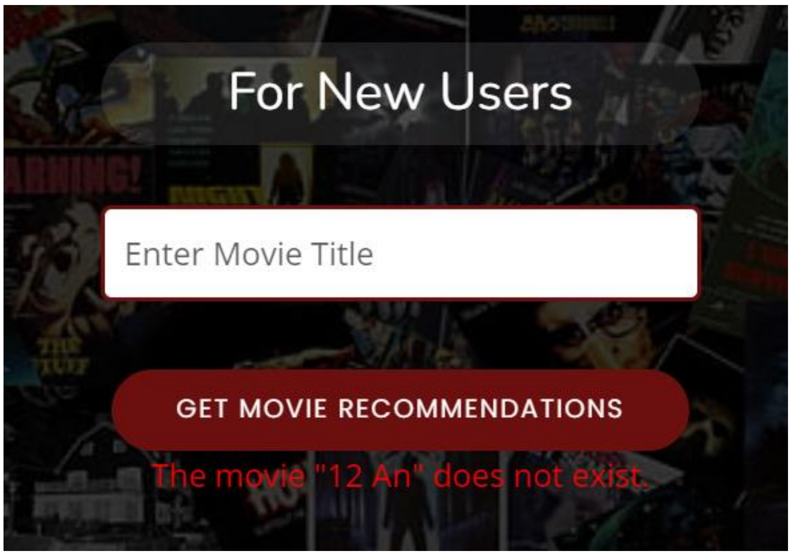
Validation is done for user id input and movie title input



For Existing Users

Please enter only numeric values for User ID.

Fig 6.8   Validation for user id

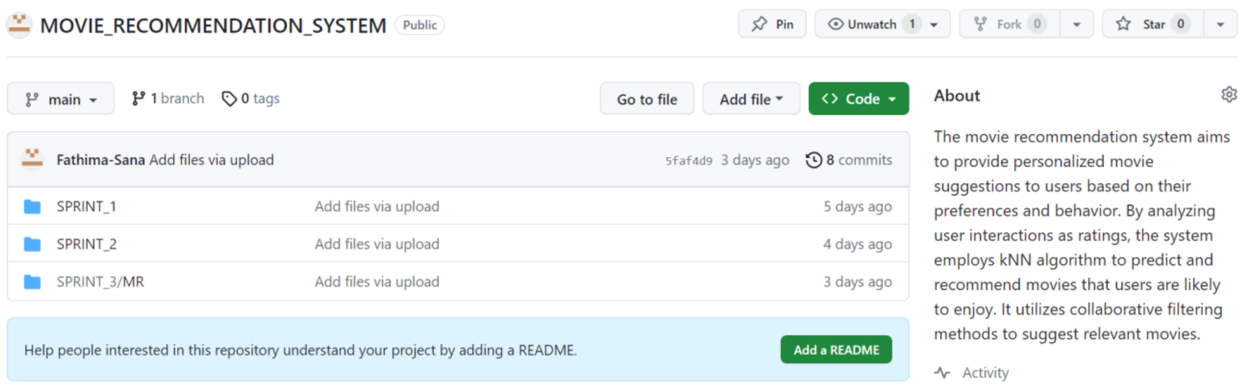


The movie "12 An" does not exist.

Fig 6.9 Validation for movie title

7. GIT HISTORY

Git Repository Movie Recommendation System contains all colab files, py files, html files and three related research papers. It is maintained for systematic way of project presentation and mainly for future reference. The colab files are created separatelyfor three sprint releases during the project.



MOVIE_RECOMMENDATION_SYSTEM Public

main · 1 branch · 0 tags

Fathima-Sana Add files via upload · 5faf4d9 3 days ago · 8 commits

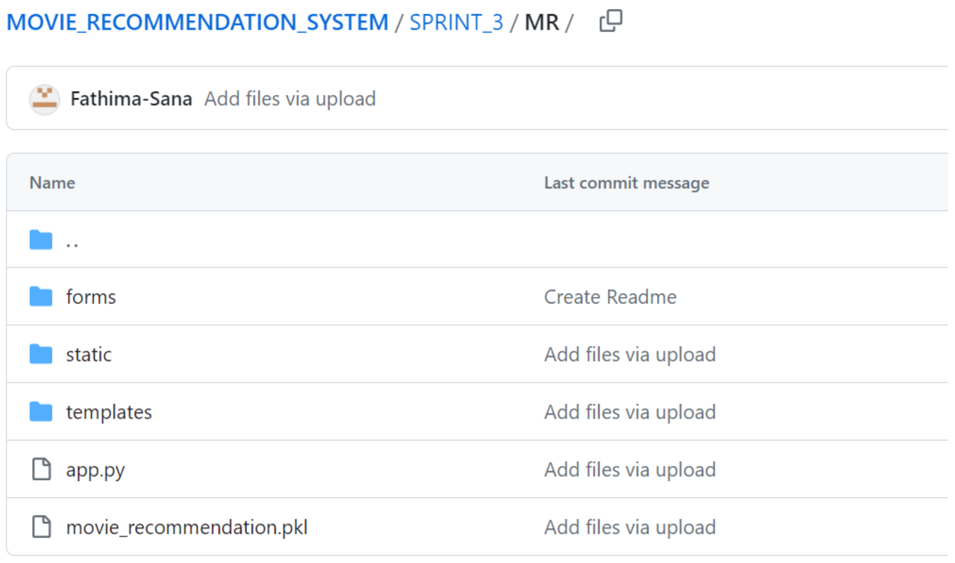
SPRINT_1	Add files via upload	5 days ago
SPRINT_2	Add files via upload	4 days ago
SPRINT_3/MR	Add files via upload	3 days ago

Help people interested in this repository understand your project by adding a README.

About

The movie recommendation system aims to provide personalized movie suggestions to users based on their preferences and behavior. By analyzing user interactions as ratings, the system employs kNN algorithm to predict and recommend movies that users are likely to enjoy. It utilizes collaborative filtering methods to suggest relevant movies.

Fig 7.1 Git History of 3 Sprints



MOVIE_RECOMMENDATION_SYSTEM / SPRINT_3 / MR /

Fathima-Sana Add files via upload

Name	Last commit message
..	
forms	Create Readme
static	Add files via upload
templates	Add files via upload
app.py	Add files via upload
movie_recommendation.pkl	Add files via upload

Fig 7.1 Git History of sprint 3

8. CONCLUSION

In conclusion, the Movie Recommendation System project has achieved its primary goal of enhancing user experience through the meticulous development and deployment of robust recommendation models. Leveraging the k-Nearest Neighbors (k-NN) algorithm with Cosine Similarity Metrics, the system excels in providing accurate and relevant movie suggestions by discerning intricate user-item relationships. The project's technical contributions, including algorithmic excellence and scalability, ensure the system's effectiveness in handling diverse user bases and evolving data volumes. The user-centric design integrates feedback mechanisms and a user-friendly interface, fostering seamless user interactions. The successful deployment of production-ready models, API endpoints, and frontend integration underscores the system's transition from theoretical development to practical application. Security measures, such as authentication and authorization, coupled with real-time monitoring and logging, contribute to the system's reliability and stability. Lessons learned from the project will guide future improvements, and outlined plans for scalability enhancements and feature updates ensure continuous refinement. The Movie Recommendation System, a tangible outcome of data science principles and user-centric design, is poised to shape the future of personalized content discovery, providing users with enriching and tailored experiences in the dynamic landscape of content recommendation.

9. FUTURE WORK

- Advanced Personalization Algorithms:

  Exploring more sophisticated personalization algorithms represents a key avenue for future work. Integrating advanced machine learning techniques, such as deep learning, could enable the model to capture intricate patterns in user preferences and item features, leading to more accurate and personalized recommendations. Additionally, investigating hybrid models that blend collaborative filtering and content-based filtering could offer a comprehensive solution to enhance recommendation precision.
- Dynamic User Preferences Modeling:

  Recognizing that user preferences evolve over time, future iterations could focus on dynamic modeling. By adapting the recommendation model to changing user behaviors and incorporating recent interactions, the system can provide real-time, personalized suggestions, ensuring that recommendations remain aligned with users' evolving tastes.
- Context-Aware Recommendations:

  Adding a layer of context awareness to the recommendation system could significantly elevate user satisfaction. Future work might involve integrating contextual information, such as user location, time of day, and device type, to tailor recommendations to the user's current situation. This level of contextual intelligence could result in more precise and timely suggestions.
- Explanability and Transparency:

  Addressing the "black-box" nature of recommendation systems is crucial for user trust. Future efforts could concentrate on developing explainable models, employing techniques like model interpretation to provide transparent insights into the decision-making process. This transparency enhances user understanding of why a specific recommendation is made.
- Scalability and Infrastructure Optimization:

  As user bases expand, ensuring the scalability of the recommendation system becomes imperative. Future work may involve optimizing the system's infrastructure, exploring distributed computing frameworks, and leveraging cloud services to handle increasing data volumes and user interactions efficiently, ensuring a seamless and responsive user experience.

- Interactive User Feedback Mechanisms: Implementing an interactive feedback loop empowers users to provide explicit feedback on recommendations. Future work could include features allowing users to rate suggestions, provide comments, or express preferences explicitly. Leveraging this feedback, the system can iteratively adapt and improve its accuracy, fostering a collaborative user-system interaction.
- Incorporating Diversity Metrics:

  To mitigate biases and promote diversity in recommendations, future iterations could integrate diversity metrics. This involves measuring and optimizing recommendations for diversity, ensuring a more inclusive and varied set of suggestions. By incorporating these metrics, the system can contribute to a more equitable and diverse content ecosystem.
- User Engagement Analytics:

  Continuous monitoring of user engagement metrics is essential for refining the recommendation system. Future work could include the implementation of robust analytics tools to track user interactions, measure recommendation effectiveness, and identify areas for improvement. This data-driven approach ensures that the system evolves in response to user behaviors, optimizing the overall user experience.

In summary, the future work for the Movie Recommendation System project spans algorithmic advancements, dynamic modeling, context awareness, transparency, user feedback mechanisms, scalability optimization, diversity considerations, and comprehensive user engagement analytics. By addressing these facets, the system can evolve into a sophisticated, user-centric platform that continually adapts to changing preferences and provides an enriched movie-watching experience.

10. APPENDIX

10.1. Minimum Software Requirements

To deploy and run the Movie Recommendation System, the following minimum software requirements must be met:

- Python:
  Version 3.6 or above is required to execute the project code and its dependencies.
- NumPy:
  Essential for numerical operations and array manipulations in Python.
- Pandas:
  Necessary for data manipulation, cleaning, and analysis.
- Scikit-learn:
  The scikit-learn library is used for machine learning functionalities, including the implementation of the k-Nearest Neighbors (k-NN) algorithm.
- Matplotlib and Seaborn:
  Required for data visualization and generating plots and charts.
- SciPy:
  Used for scientific and technical computing, particularly for sparse matrix operations.
- TQDM:
  Essential for displaying progress bars during time-consuming operations.

- Pickel:

  Used for serialization and deserialization of Python objects, particularly for saving and loading machine learning models.

- Colab Notebook:

  If using Colab notebooks for development and testing, having Colab installed is recommended.

- Other Dependencies:

  Additional libraries and modules specified in the project code, including Nearest Neighbors, Scipy's sparse matrix (csr_matrix), and other utility modules.

10.2. Minimum Hardware Requirements

The Movie Recommendation System is relatively lightweight, but it does involve some computational processes. The minimum hardware requirements to ensure smooth execution are as follows:

Processor (CPU):
A modern multi-core processor (e.g., Intel Core i5 or equivalent) is recommended to handle the computation involved in the recommendation algorithms efficiently.

RAM:
A minimum of 8 GB RAM is recommended to manage the dataset and perform machine learning operations smoothly.

Storage:
Adequate storage space for storing datasets, code, and any additional resources. While the system itself doesn't require extensive storage, having sufficient space for datasets and potential model persistence is essential.

Internet Connection:
An internet connection is necessary for downloading datasets, libraries, and dependencies during the setup phase.

11. REFERENCES

1. Gupta, Meenu, et al. "Movie recommender system using collaborative filtering." *2020 international conference on electronics and sustainable communication systems (ICESC)*. IEEE, 2020.

2. Raghavendra, C. K., and K. C. Srikantaiah. "Similarity based collaborative filtering model for Movie Recommendation Systems." *2021 5th international conference on intelligent computing and control systems (ICICCS)*. IEEE, 2021.

3. Anwar, Taushif, and V. Uma. "Comparative study of recommender system approaches and movie recommendation using collaborative filtering." *International Journal of System Assurance Engineering and Management* 12 (2021): 426-436.

4. https://www.kaggle.com/code/gpreda/user-based-collaborative-filtering-using-knn

5. https://www.kaggle.com/code/gpreda/item-based-collaborative-filtering-using-knn

6. https://bootstrapmade.com/tempo-free-onepage-bootstrap-theme/