BUILDING A MOVIE RECOMMENDATION SYSTEM

Photo by Tyson Moultrie on Unsplash

Project by:

1. Mwangi Wambugu
2. Grace Mutuku
3. Rony Muriithi
4. Esther Nyawera
5. John Kioko
6. Heri Kimotho
7. Peter Otieno

BUSINESS UNDERSTANDING

MOVIE INDUSTRY OVERVIEW

The film industry, a global juggernaut, annually amasses billions in revenue. As per Statista, in 2019 alone, the global box office accumulated a staggering $42.5 billion.
Film production is the creative hub where movies, TV shows, and other visual content come to life, involving stakeholders such as studios, independent producers, and streaming platforms.
The distribution segment markets and transports these creations to theaters, streaming services, and other outlets. Exhibition refers to the screening of movies and TV shows in cinemas
Movies possess a universal charm, forging connections among individuals from diverse backgrounds. Despite this collective appeal, our personal cinematic tastes exhibit uniqueness, spanning various genres such as thrillers, romance, or sci-fi, and often centering around preferred actors and directors.

STREAMING SERVICES INDUSTRY OVERVIEW

Streaming services deliver digital content, including movies, TV shows, documentaries, and original productions, directly to users over the internet. Streaming services have expanded globally, making content accessible to audiences worldwide.
Streaming platforms often provide personalized recommendations based on user preferences and viewing history.
Examples of widely known streaming platforms include:

1. Netflix
2. Hulu
3. Amazon Prime Video
4. Showmax
5. Disney+

BUSINESS PROBLEM

In the world of streaming services, with the rise in users, competition among streaming services is at an all time high. With a high demand in content, the streaming services need to keep up with growing tastes in the consumer base and they need to push content that is relevant and up to their consumers standards.

Filamu Streaming Services aims to stay competitive in this industry by utilizing the data gotten from GroupLens research lab at the University of Minnesota to build a better and more advanced recommendation system.

Crafting a universal formula for movies that would captivate every individual proves challenging; however, through meticulous analysis of user data and movie-related data,valuable insights are extracted so as to provide users with movie recommendations that would best suit their preferences.

OBJECTIVES

1. To analyse the data and find the most frequently watched movies.
2. To analyse the data and find the most frequently watched genres based on the data provided.
3. Create a recommendation system that makes movie recommendations based on user ratings by suggesting products that resonate with user's past ratings.

PROJECT OVERVIEW

RECOMMENDATION SYSTEMS

A Recommendation System is a filtration program whose prime goal is to predict a user’s preference towards a domain-specific item or item based on their past behavior, preferences, or the behavior of similar users. Recommendation systems are widely used in various online platforms to enhance user experience, increase engagement, and help users discover relevant and personalized content. In our case, this domain-specific item is a movie.

Therefore the main focus of our recommendation system is to filter and predict only those movies which a user would prefer given previous rating he or she had made on another movies.

Types of Recommendation Systems

1. **Content Based Filtering**

   Content-based filtering is a recommendation strategy that suggests items similar to those a user has previously liked. It calculates similarity (often using cosine similarity) between the user’s preferences and item attributes, such as lead actors, directors, and genres. For example, if a user enjoys ‘The Prestige’, the system recommends movies with ‘Christian Bale’, ‘Thriller’ genre, or films by ‘Christopher Nolan’.

   However, content-based filtering has drawbacks. It limits exposure to different products, preventing users from exploring a variety of items. This can hinder business expansion as users might not try out new types of products

2. **Collaborative Filtering**

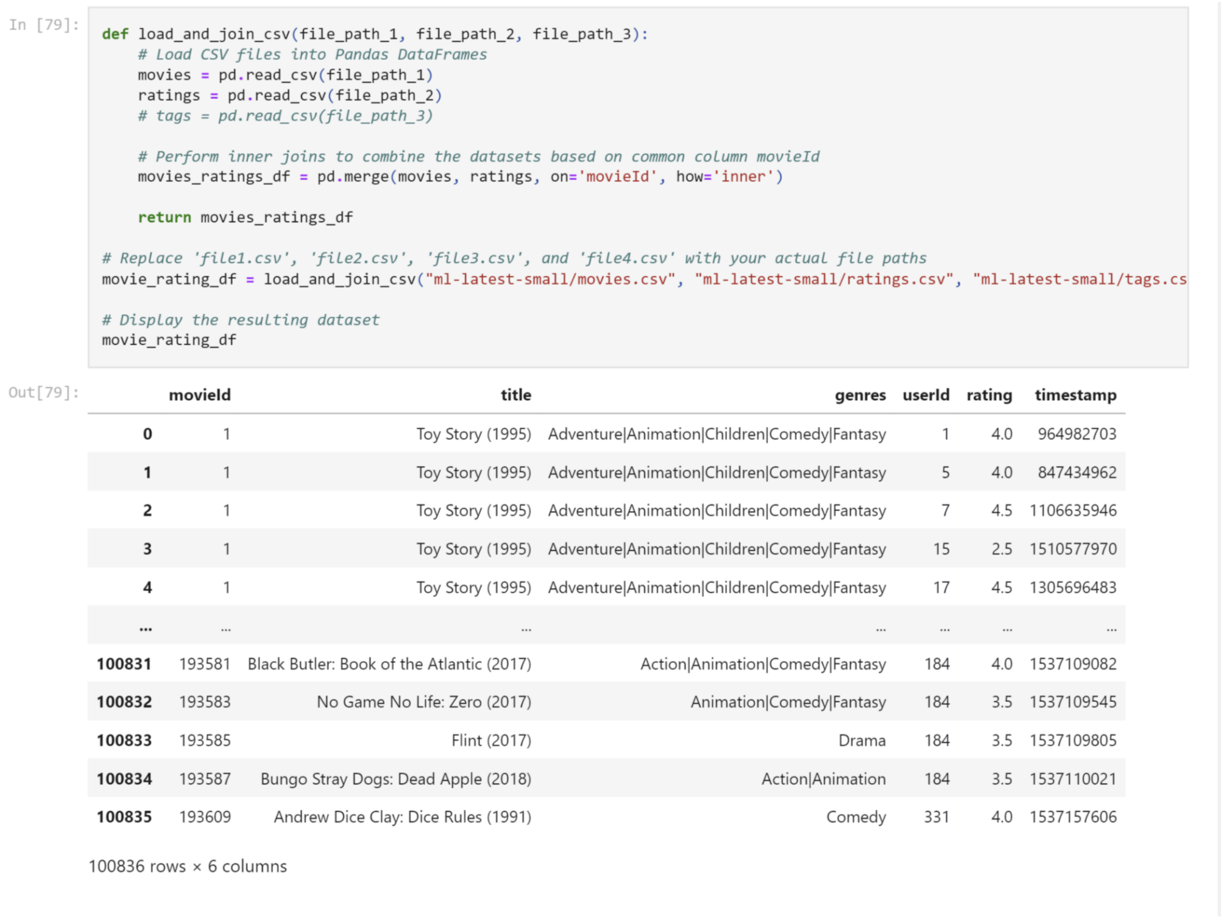
   Collaborative filtering is a recommendation strategy that considers the user’s behavior and compares it with other users in the database. It uses the history of all users to influence the recommendation algorithm. Unlike content-based filtering, collaborative filtering relies on the interactions of multiple users with items to generate suggestions. It doesn’t solely depend on one user’s data for modeling. There are various approaches to implement collaborative filtering, but the key concept is the collective influence of multiple users on the recommendation outcome.

OUTLINE OF THE PROJECT

1. Data Preparation
2. Data Understanding
3. Exploratory Data Analysis
4. Modeling
5. Model Prediction
6. Conclusion
7. Recommendation
8. Next Steps
9. Deployment

1. DATA PREPARATION

Using the pandas library, we loaded the data into our jupyter notebook.



```
In [79]:
def load_and_join_csv(file_path_1, file_path_2, file_path_3):
    # Load CSV files into Pandas DataFrames
    movies = pd.read_csv(file_path_1)
    ratings = pd.read_csv(file_path_2)
    # tags = pd.read_csv(file_path_3)

    # Perform inner joins to combine the datasets based on common column movieId
    movies_ratings_df = pd.merge(movies, ratings, on='movieId', how='inner')

    return movies_ratings_df

# Replace 'file1.csv', 'file2.csv', 'file3.csv', and 'file4.csv' with your actual file paths
movie_rating_df = load_and_join_csv("ml-latest-small/movies.csv", "ml-latest-small/ratings.csv", "ml-latest-small/tags.cs

# Display the resulting dataset
movie_rating_df
```

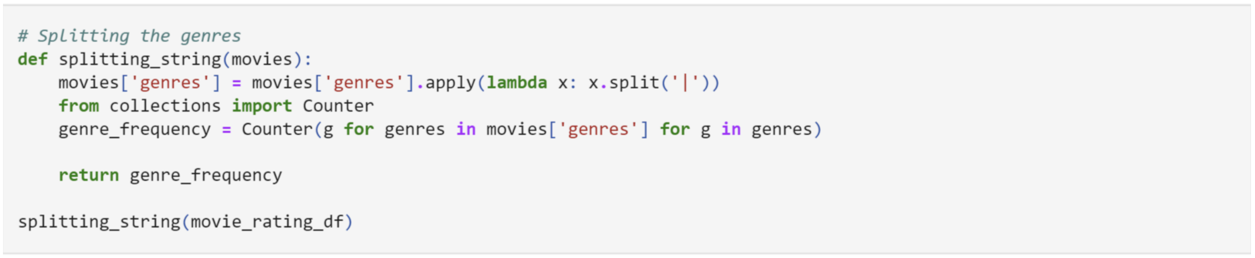
Out[79]:

	movieId	title	genres	userId	rating	timestamp
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	1	4.0	964982703
1	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	5	4.0	847434962
2	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	7	4.5	1106635946
3	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	15	2.5	1510577970
4	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	17	4.5	1305696483
...	...	...	...	...	...	...
100831	193581	Black Butler: Book of the Atlantic (2017)	Action|Animation|Comedy|Fantasy	184	4.0	1537109082
100832	193583	No Game No Life: Zero (2017)	Animation|Comedy|Fantasy	184	3.5	1537109545
100833	193585	Flint (2017)	Drama	184	3.5	1537109805
100834	193587	Bungo Stray Dogs: Dead Apple (2018)	Action|Animation	184	3.5	1537110021
100835	193609	Andrew Dice Clay: Dice Rules (1991)	Comedy	331	4.0	1537157606

100836 rows × 6 columns

Multiple datasets that had a common column, movieId, were merged into a pandas dataframe so as to make the data more comprehensible.

The values in the “genre” column were a list which would pose a problem in correctly analyzing each different genre. A function to split the list into individual genres was created.



```
# Splitting the genres
def splitting_string(movies):
    movies['genres'] = movies['genres'].apply(lambda x: x.split('|'))
    from collections import Counter
    genre_frequency = Counter(g for genres in movies['genres'] for g in genres)

    return genre_frequency

splitting_string(movie_rating_df)
```

There were 0 null values and 0 duplicate entries in the now almost complete dataset.

2. DATA UNDERSTANDING

UNDERSTANDING THE FEATURES

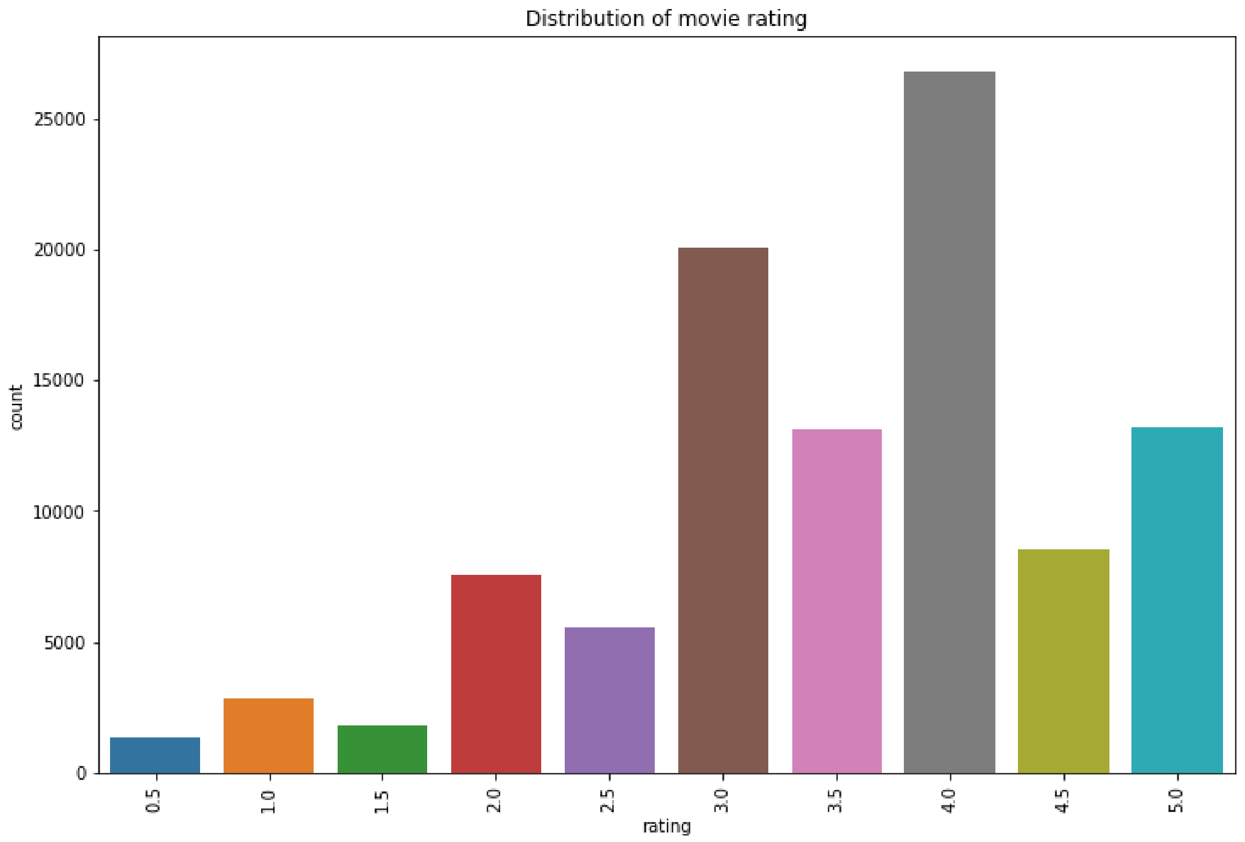
1. **MovieId -** represents a unique identifier of the movie
2. **Title -** name of the film containing the coinciding movieId
3. **Genres -** a stylistic or thematic category for motion pictures based on similarities either in the narrative elements, aesthetic approach, or the emotional response to the film
4. **UserId -** a unique customer identifier by which an advertiser chooses to identify a user visiting their website.
5. **Rating -** a measurement of the quality or success of something
6. **Timestamp -** a digital record of the time of occurrence of a particular event.

3. EXPLORATORY DATA ANALYSIS

3.1 UNIVARIATE DATA ANALYSIS

Univariate analysis is the simplest form of analyzing data. “Uni” means “one”, so in other words your data has only one variable. It doesn’t deal with causes or relationships (unlike regression) and it’s major purpose is to describe; It takes data, summarizes that data and finds patterns in the data.

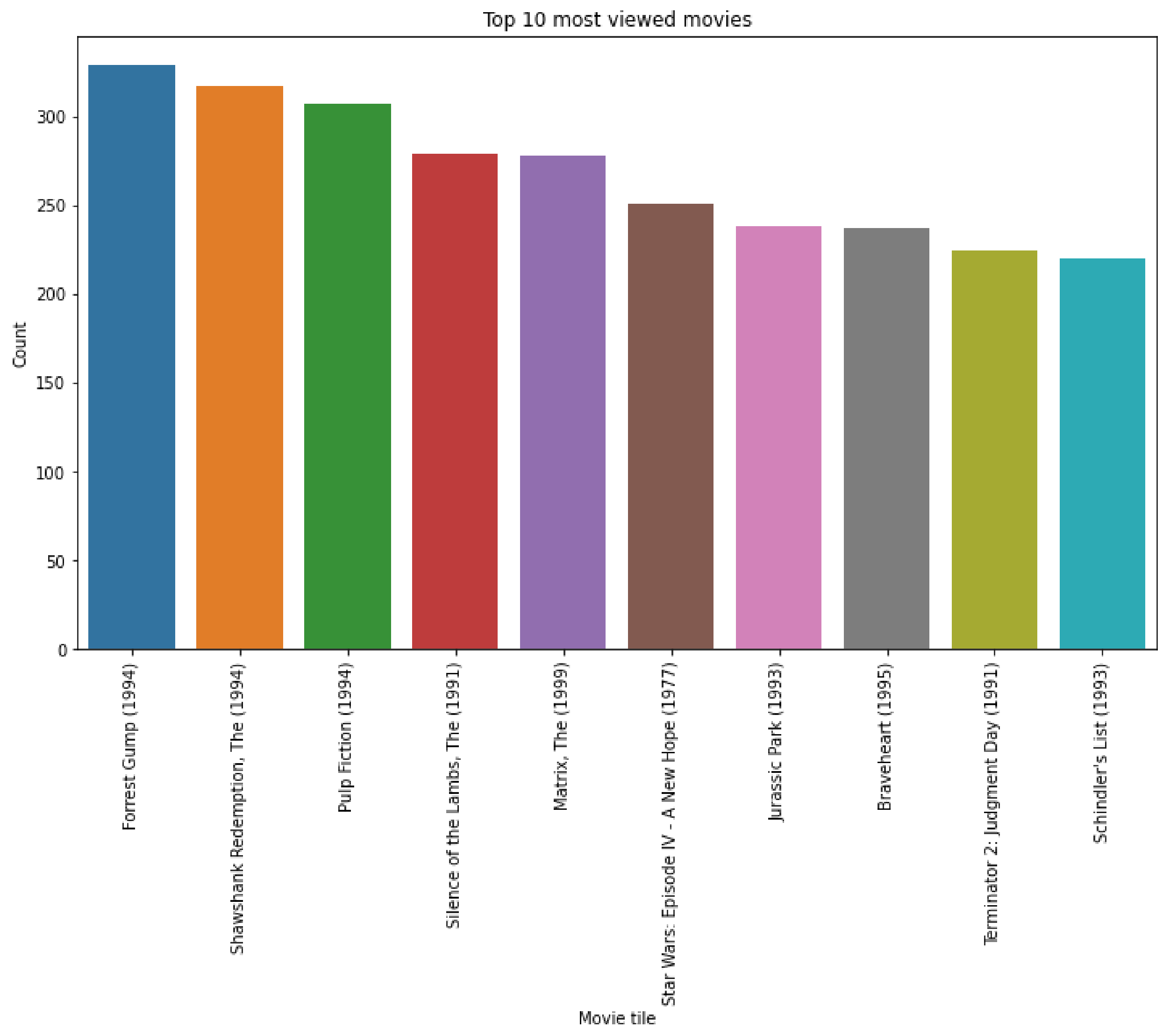
DISTRIBUTION OF MOVIE RATINGS



The following conclusions are drawn from the count plot above, which displays the distribution of movie ratings:

1. The users are generally generous with their ratings as a majority of the rating are above 3.0. The most frequent rating was a rating of 4.0 with a bit over 25,000 movies.
2. The second most frequent rating was 3.0 with about 20,000 movies.
3. The least most common rating was 0.5

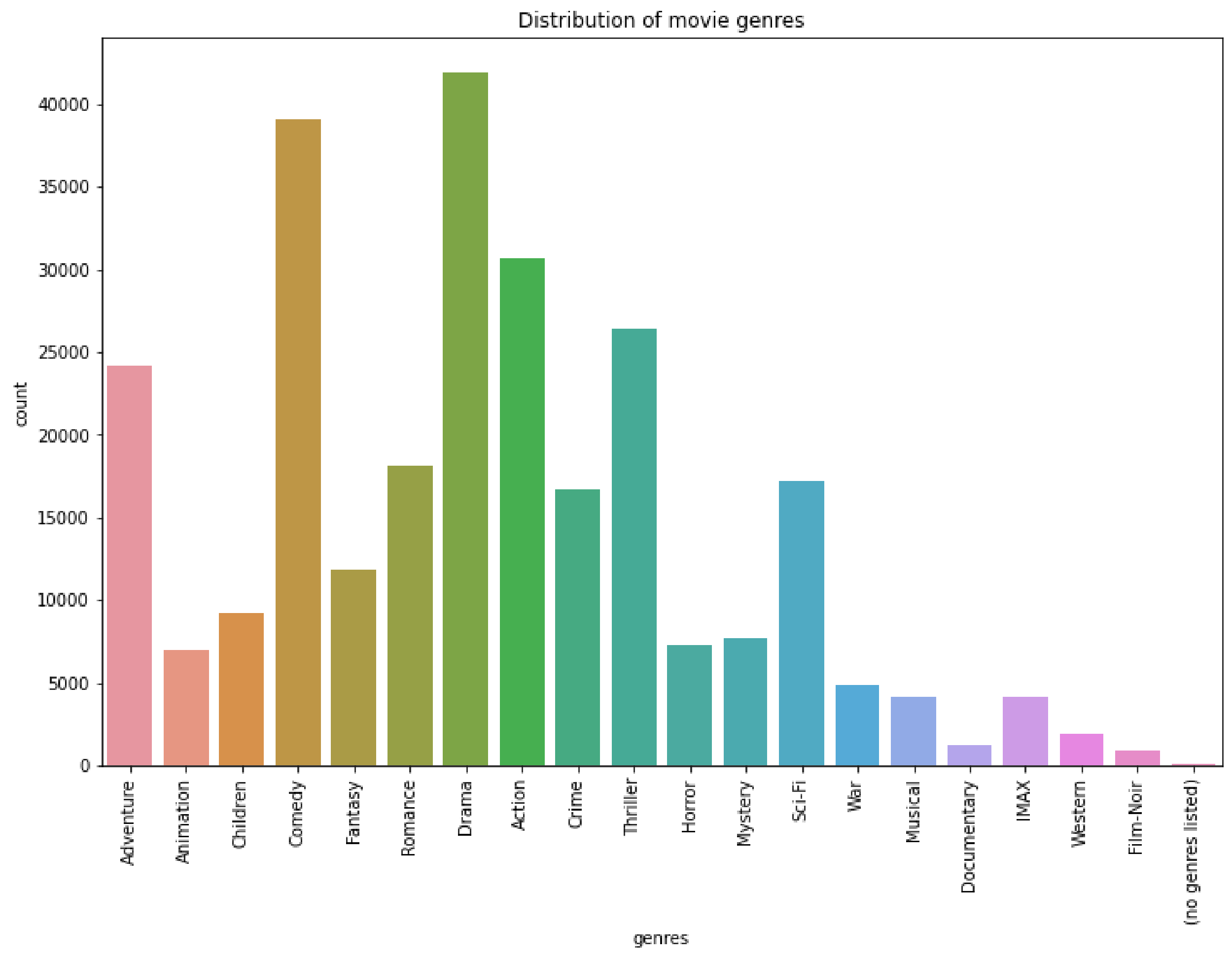
TOP 10 MOST VIEWED MOVIES



The count plot above shows the most viewed movie titles. The top 10 movies are:

1. Forrest Gump(1994)
2. The Shawshank Redemption(1994)
3. Pulp Fiction(1994)
4. The Silence of The Lambs(1991)
5. The Matrix(1999)
6. Star Wars: Episode IV - A New Hope(1977)
7. Jurassic Park(1993)
8. Braveheart(1995)
9. Terminator 2: Judgement Day(1991)
10. Schindler’s List(1993)

DISTRIBUTION OF MOVIE GENRES



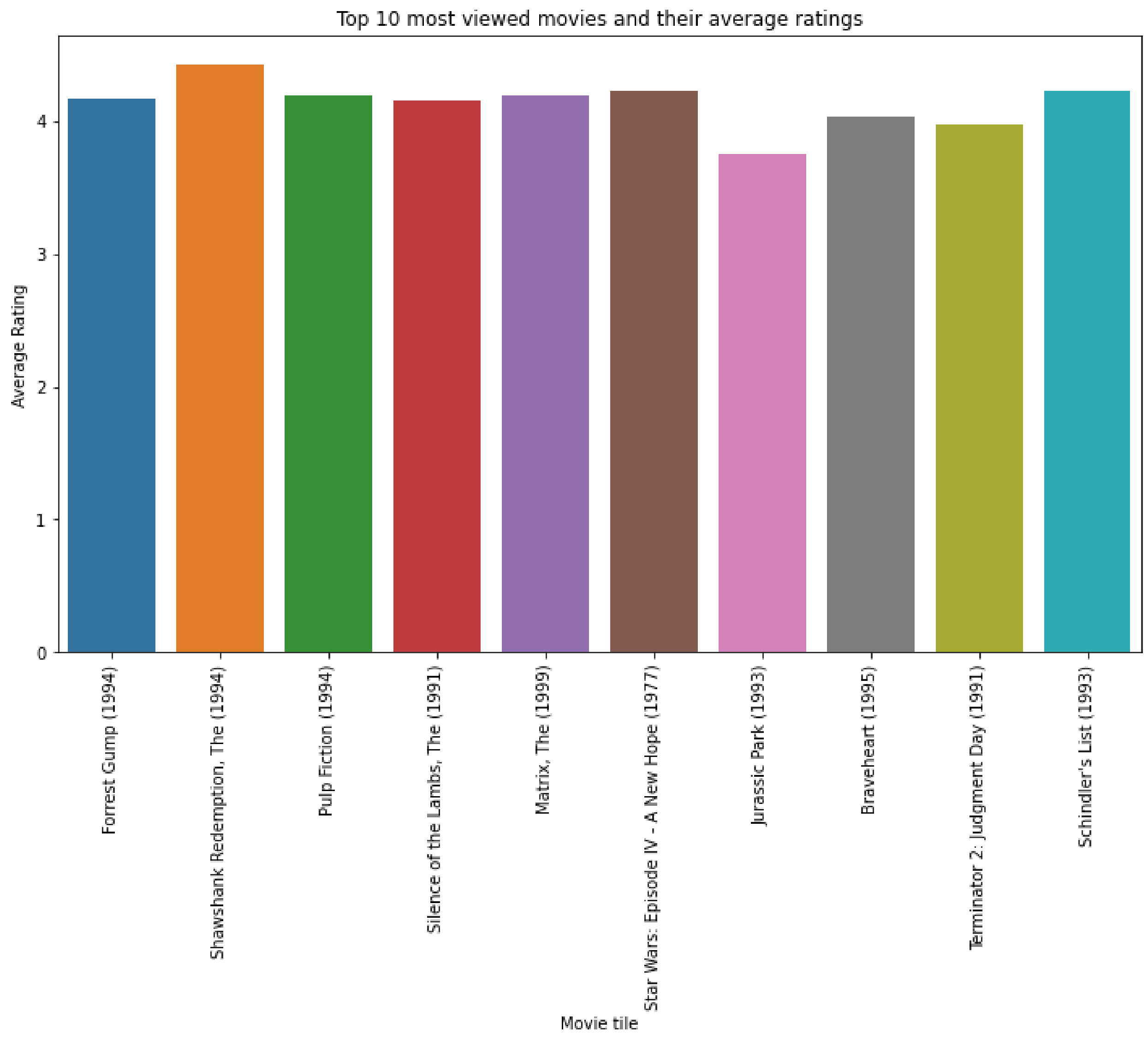
The bar plot shows the distribution of all the movie genres in the dataset. The 10 top genres are listed below starting from the most common one:

- Drama
- Comedy
- Action
- Thriller
- Adventure
- Romance
- Sci-Fi
- Crime
- Fantasy
- Children

3.2 BIVARIATE ANALYSIS

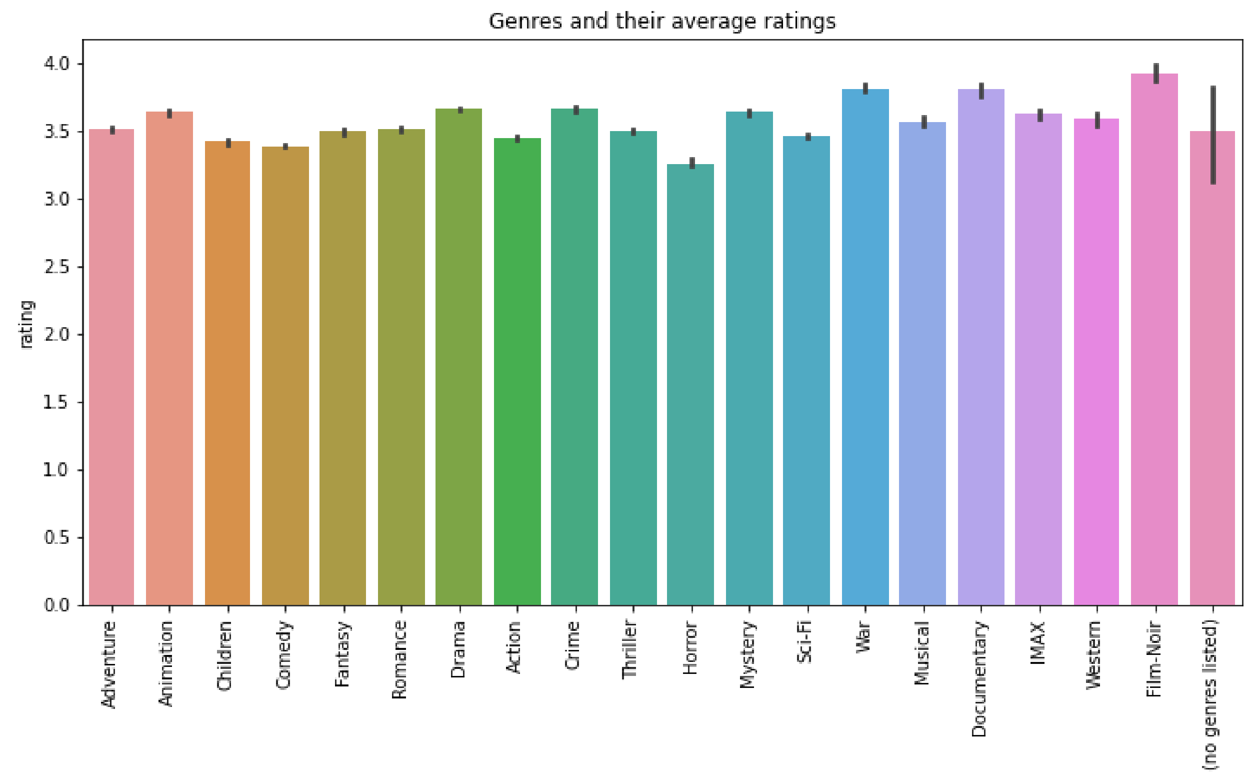
Bivariate analyses are conducted to determine whether a statistical association exists between two variables, the degree of association if one does exist, and whether one variable may be predicted from another.

TOP 10 MOST VIEWED MOVIES AND THEIR AVERAGE RATINGS

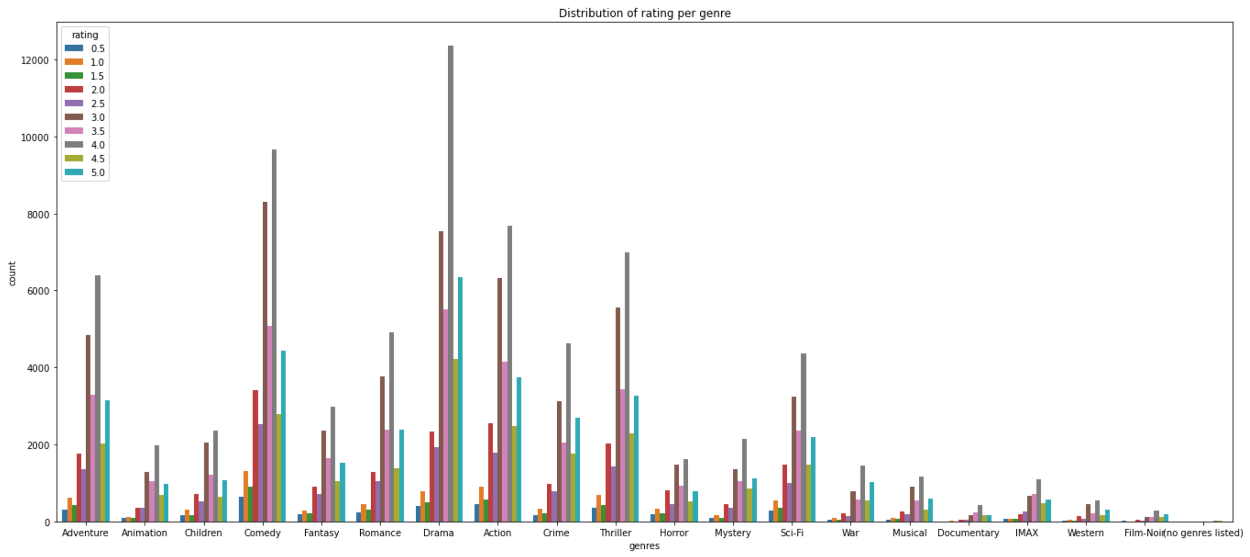


The bar graph above shows the top 10 most viewed movies and there average ratings. The average ratings range from 3.5 to 4.5 for the top 10 movies.

GENRES AND THEIR AVERAGE RATINGS



From the bar chart it can be seen that all the genres have an average rating of above 3. Film Noir has the highest average rating followed by Documentary.



All the genres have most of their movies rated 4.0, with relatively fewer users giving a rating of either 1 and 2 to the movies.

Drama, Comedy, Action, Thriller and Adventure appear to have the highest count of ratings indicating higher user engagement.

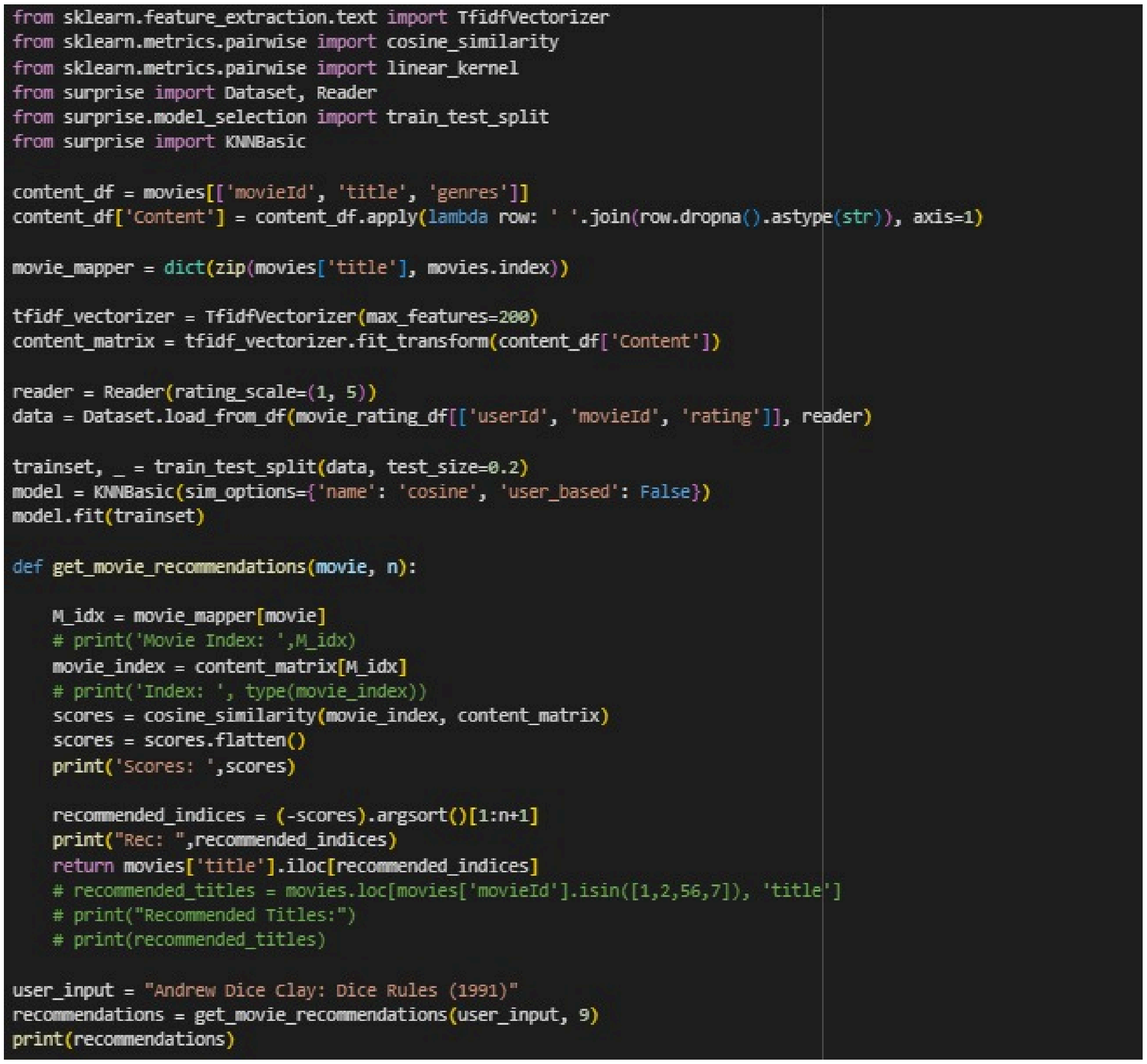
Film Noir, Western and Documentary genres seem to have the lowest count of ratings indicating low user engagement

4. MODELING

4. 1 CONTENT-BASED RECOMMENDATION SYSTEM

The content-based recommendation relies on analyzing and comparing the characteristics of movies to make personalized suggestions. These characteristics often include genres, keywords, or other metadata associated with each movie. The idea is to recommend films that are similar to those the user has shown interest in before, focusing on the content features of the items.

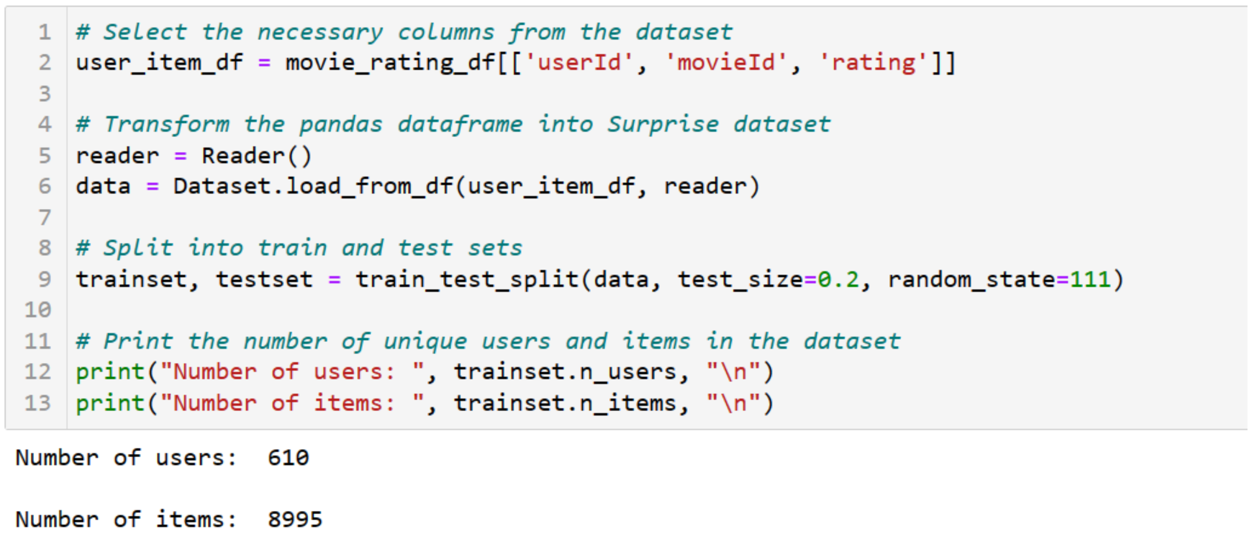
The file below employs content-based recommendation principles to suggest movies based on their textual features, specifically titles and genres. The use of TF-IDF and cosine similarity facilitates the calculation of similarity scores for personalized movie recommendations. Additionally, there is a collaborative filtering section using Surprise, showcasing a hybrid recommendation system.



```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics.pairwise import linear_kernel
from surprise import Dataset, Reader
from surprise.model_selection import train_test_split
from surprise import KNNBasic

content_df = movies[['movieId', 'title', 'genres']]
content_df['Content'] = content_df.apply(lambda row: ' '.join(row.dropna().astype(str)), axis=1)

movie_mapper = dict(zip(movies['title'], movies.index))

tfidf_vectorizer = TfidfVectorizer(max_features=200)
content_matrix = tfidf_vectorizer.fit_transform(content_df['Content'])

reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(movie_rating_df[['userId', 'movieId', 'rating']], reader)

trainset, _ = train_test_split(data, test_size=0.2)
model = KNNBasic(sim_options={'name': 'cosine', 'user_based': False})
model.fit(trainset)

def get_movie_recommendations(movie, n):

    M_idx = movie_mapper[movie]
    # print('Movie Index: ',M_idx)
    movie_index = content_matrix[M_idx]
    # print('Index: ', type(movie_index))
    scores = cosine_similarity(movie_index, content_matrix)
    scores = scores.flatten()
    print('Scores: ',scores)

    recommended_indices = (-scores).argsort()[1:n+1]
    print("Rec: ",recommended_indices)
    return movies['title'].iloc[recommended_indices]
    # recommended_titles = movies.loc[movies['movieId'].isin([1,2,56,7]), 'title']
    # print("Recommended Titles:")
    # print(recommended_titles)

user_input = "Andrew Dice Clay: Dice Rules (1991)"
recommendations = get_movie_recommendations(user_input, 9)
print(recommendations)
```

4.2 USER-BASED COLLABORATIVE RECOMMENDATION SYSTEM

The technique used is a user-based collaborative filtering which relies on the idea that individuals with similar preferences tend to enjoy similar items or content.



```python
# Select the necessary columns from the dataset
user_item_df = movie_rating_df[['userId', 'movieId', 'rating']]

# Transform the pandas dataframe into Surprise dataset
reader = Reader()
data = Dataset.load_from_df(user_item_df, reader)

# Split into train and test sets
trainset, testset = train_test_split(data, test_size=0.2, random_state=111)

# Print the number of unique users and items in the dataset
print("Number of users: ", trainset.n_users, "\n")
print("Number of items: ", trainset.n_items, "\n")
```

```
Number of users:  610

Number of items:  8995
```

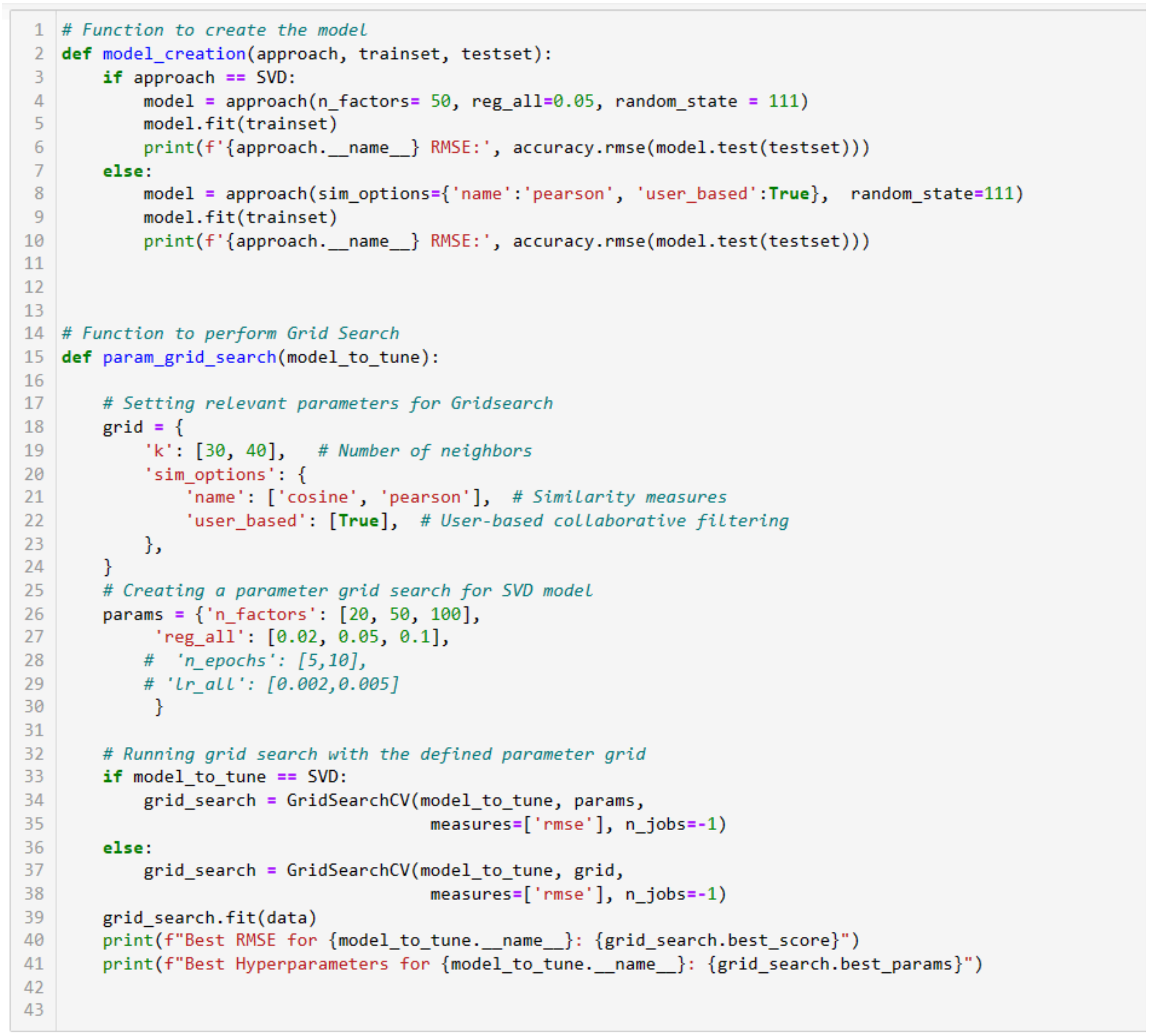
Relevant columns from the pandas dataframe were selected and transformed into a surprise dataset then split into a 80:20 trainset and testset, with a random state of 111 to ensure reproducibility .

The ‘userId’, ‘movieId’ and ‘rating’ columns are extracted from the movie_rating_df dataset.

Modeling Functions

The functions in this cell are defined inorder to create and tune collaborative filtering models. The functions are:

- ‘Model_creation’ - It builds and fits collaborative filtering models (KNN and SVD) with optional hyperparameters
- ‘Param_grid_search’ - It sets up a grid search for hyperparameter tuning



```python
# Function to create the model
def model_creation(approach, trainset, testset):
    if approach == SVD:
        model = approach(n_factors= 50, reg_all=0.05, random_state = 111)
        model.fit(trainset)
        print(f'{approach.__name__} RMSE:', accuracy.rmse(model.test(testset)))
    else:
        model = approach(sim_options={'name':'pearson', 'user_based':True},  random_state=111)
        model.fit(trainset)
        print(f'{approach.__name__} RMSE:', accuracy.rmse(model.test(testset)))



# Function to perform Grid Search
def param_grid_search(model_to_tune):

    # Setting relevant parameters for Gridsearch
    grid = {
        'k': [30, 40],    # Number of neighbors
        'sim_options': {
            'name': ['cosine', 'pearson'],  # Similarity measures
            'user_based': [True],  # User-based collaborative filtering
        },
    }
    # Creating a parameter grid search for SVD model
    params = {'n_factors': [20, 50, 100],
             'reg_all': [0.02, 0.05, 0.1],
        #  'n_epochs': [5,10],
        # 'lr_all': [0.002,0.005]
             }

    # Running grid search with the defined parameter grid
    if model_to_tune == SVD:
        grid_search = GridSearchCV(model_to_tune, params,
                                   measures=['rmse'], n_jobs=-1)
    else:
        grid_search = GridSearchCV(model_to_tune, grid,
                                   measures=['rmse'], n_jobs=-1)
    grid_search.fit(data)
    print(f"Best RMSE for {model_to_tune.__name__}: {grid_search.best_score}")
    print(f"Best Hyperparameters for {model_to_tune.__name__}: {grid_search.best_params}")
```

The functions in the above cell automate the process of creating a model and hyperparameter tuning for collaborative models, providing an efficient way to experiment with different algorithms and settings.

Model Building and Tuning

Here different models are constructed using surprise library's algorithms, then evaluated and tuned according to collaborative filtering model methods.

Models : The models include

- KNNBasic

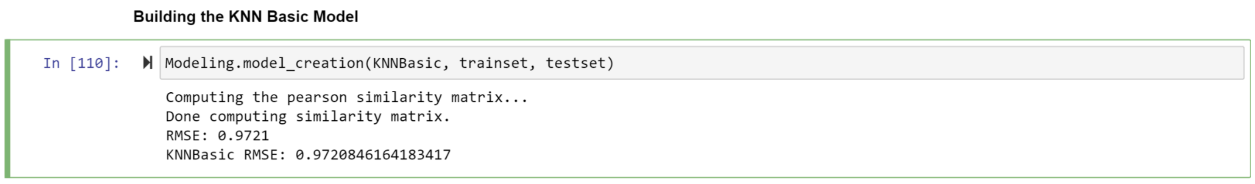
- KNNBaseline
- KNNWithMeans
- SVD

Evaluation Metric: Root Mean Squared Error **(**RMSE)
RMSE is a measure of the differences between values predicted by a model or an estimator and the values observed. In the context of collaborative filtering and recommendation systems, RMSE is used to evaluate the accuracy of the model's predictions. A lower RMSE value indicates that the model's predictions are closer to the actual observed values, hence indicating better accuracy.

RMSE is chosen as the evaluation metric since it is easy to interpret and it gives more weight to accuracy. It considers the squared differences between predicted and actual values. It is also widely used in recommender systems, making our results easily comparable with other studies.

MODEL 1: KNN BASIC MODEL (Baseline Model)

Building KNN Basic Model



Building the KNN Basic Model

```
In [110]:  Modeling.model_creation(KNNBasic, trainset, testset)

           Computing the pearson similarity matrix...
           Done computing similarity matrix.
           RMSE: 0.9721
           KNNBasic RMSE: 0.9720846164183417
```

The initial RMSE (Root Mean Square Error) for the KNNBasic model is 0.9721. In this case, an RMSE of 0.9721 indicates that, on average, the KNNBasic model's predictions are off by approximately 0.9721 units from the actual ratings in the dataset.

Tuning of the model is necessary to improve the performance of the KNNBasic model for the specific recommendation task at hand.

Tuning the KNNBasic Model

Grid search was performed to find optimal hyperparameters for the KNNBasic model.

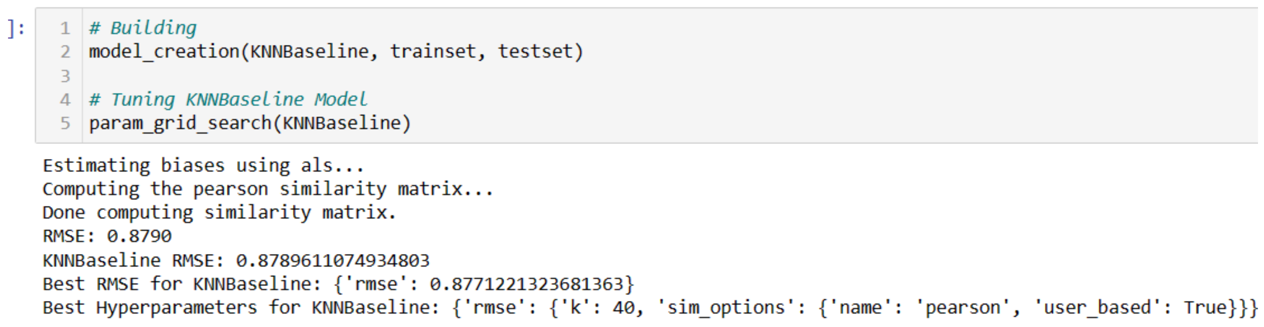
- The best hyperparameters include 40 neighbors with cosine similarity for user-based collaborative filtering.
- The tuned RMSE for KNNBasic is 0.9724, which is slightly worse than the initial model from the build one which was 0.9721

MODEL 2 : KNNBaseline Model

The KNNBaseline algorithm, extends the functionality of the KNNBasic algorithm by incorporating a baseline prediction into the similarity calculation.

KNNBaseline offers improved accuracy, robustness to sparsity, better handling of biases, and effective mitigation of the cold start problem due to the incorporation of baseline predictions. This makes KNNBaseline a preferred choice in many recommendation system scenarios, particularly when dealing with real-world, sparse, and dynamic datasets.

Building KNNBaseline Model



```
]:
# Building
model_creation(KNNBaseline, trainset, testset)

# Tuning KNNBaseline Model
param_grid_search(KNNBaseline)
```

```
Estimating biases using als...
Computing the pearson similarity matrix...
Done computing similarity matrix.
RMSE: 0.8790
KNNBaseline RMSE: 0.8789611074934803
Best RMSE for KNNBaseline: {'rmse': 0.8771221323681363}
Best Hyperparameters for KNNBaseline: {'rmse': {'k': 40, 'sim_options': {'name': 'pearson', 'user_based': True}}}
```

The KNNBaseline Model gives an RMSE of 0.8790 which is quite an improvement from the baseline model.

Tuning the KNNBaseline Model

Grid search for KNNBaseline is performed to fine-tune hyperparameters.

- The best hyperparameters include 30 neighbors with cosine similarity for user-based collaborative filtering.
- The tuned RMSE for KNNBaseline is 0.8779, further improving the model from the baseline build which had 0.8790

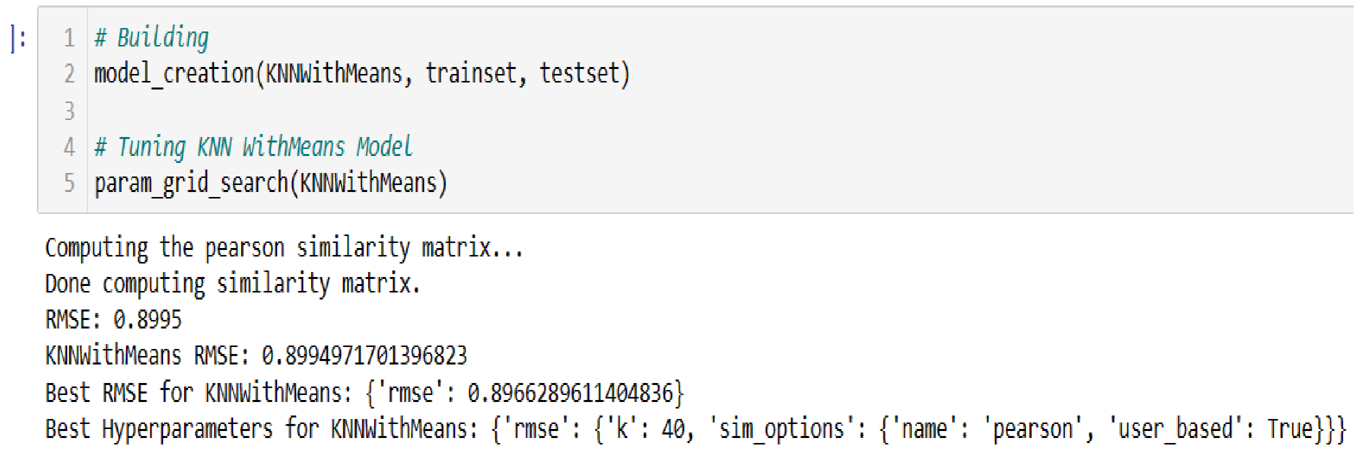
MODEL 3 : KNNWithMeans Model

In KNNWithMeans, the collaborative filtering is performed using the k-nearest neighbors approach, where the similarity between users is calculated based on their ratings.

Additionally, KNNWithMeans takes into account the mean ratings of each user or item when making predictions. This means that the model not only considers the similarities between users, but also adjusts for the average rating given by each user or received by each item.

This model is useful when there are significant variations in the rating tendencies of different users or items, and it aims to normalize these tendencies when making predictions.

Building KNNWithMeans Model



```
# Building
model_creation(KNNWithMeans, trainset, testset)

# Tuning KNN WithMeans Model
param_grid_search(KNNWithMeans)
```

```
Computing the pearson similarity matrix...
Done computing similarity matrix.
RMSE: 0.8995
KNNWithMeans RMSE: 0.8994971701396823
Best RMSE for KNNWithMeans: {'rmse': 0.8966289611404836}
Best Hyperparameters for KNNWithMeans: {'rmse': {'k': 40, 'sim_options': {'name': 'pearson', 'user_based': True}}}
```

The KNNWithMeans Model gives an RMSE of 0.8995 which is worse than the KNNBaseline Model.

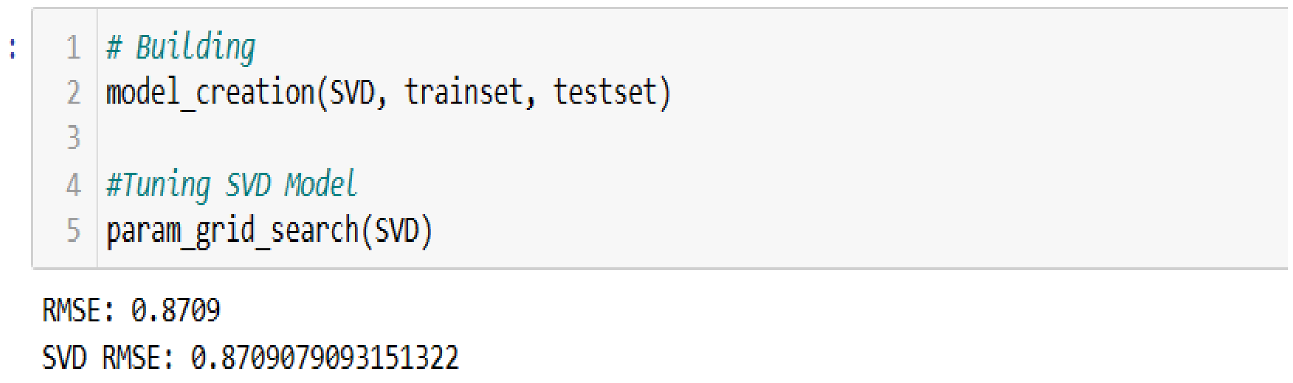
Tuning the KNNWithMeans Model
Grid search is performed to find optimal hyperparameters for the KNNWithMeans model.

- The best hyperparameters include 40 neighbors with Pearson similarity for user-based collaborative filtering.
- The tuned RMSE for KNNWithMeans is 0.8953, showing improvement over the initial model which had 0.8995.

MODEL 4 : Singular Value Decomposition (SVD) Model

The Singular Value Decomposition (SVD) model is a widely used collaborative filtering algorithm in recommendation systems. It is based on matrix factorization and has been popularized by winning the Netflix Prize competition.

Building SVD Model



```
# Building
model_creation(SVD, trainset, testset)

#Tuning SVD Model
param_grid_search(SVD)
```

```
RMSE: 0.8709
SVD RMSE: 0.8709079093151322
```

The SVD Model gives an RMSE of 0.8709 which is better than any of the above models.

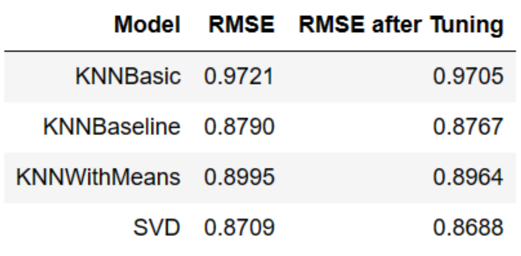
Tuning the SVD Model

Grid search is performed to find optimal hyperparameters for the SVD model.

- The best hyperparameters include 100 n-factors and a regularization term of 0.05.
- The RMSE for the tuned SVD is 0.8691, slightly improving the model.

Final Model Selection and Evaluation

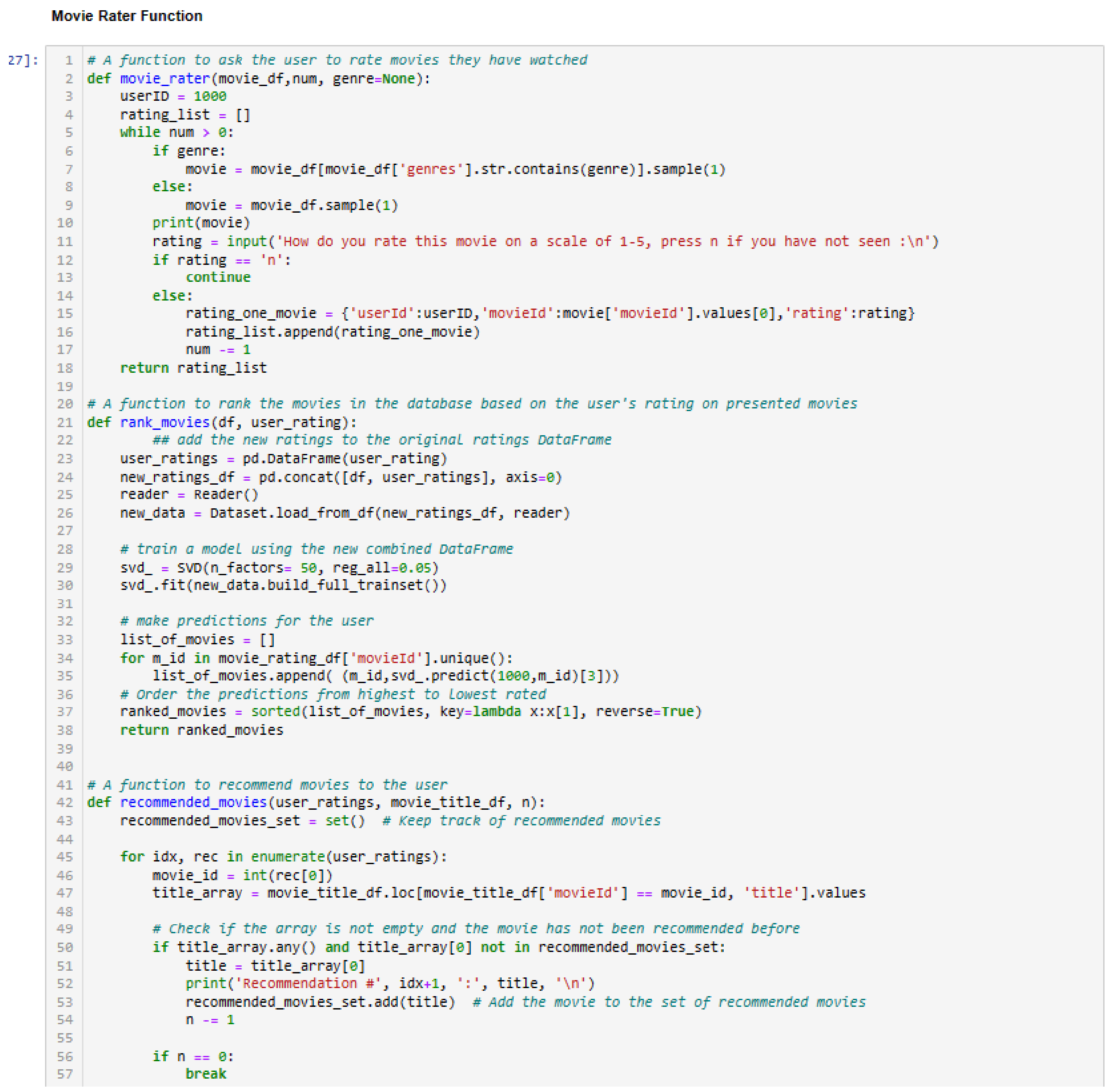
The tuned SVD Model is selected as the model because of its superior performance which will aid in training the datasets.



Model	RMSE	RMSE after Tuning
KNNBasic	0.9721	0.9705
KNNBaseline	0.8790	0.8767
KNNWithMeans	0.8995	0.8964
SVD	0.8709	0.8688

5. MODEL PREDICTION

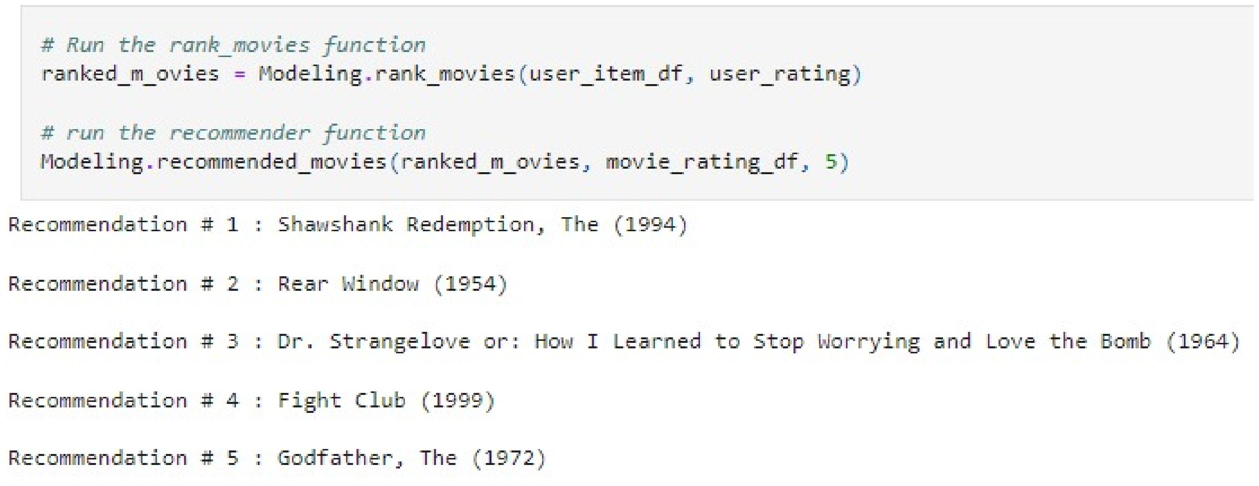
The functions, *movie_rater, rank_movies* and *recommended_movies*, help the user to interactively rate movies



Movie Rater Function

```
27]:
# A function to ask the user to rate movies they have watched
def movie_rater(movie_df,num, genre=None):
    userID = 1000
    rating_list = []
    while num > 0:
        if genre:
            movie = movie_df[movie_df['genres'].str.contains(genre)].sample(1)
        else:
            movie = movie_df.sample(1)
        print(movie)
        rating = input('How do you rate this movie on a scale of 1-5, press n if you have not seen :\n')
        if rating == 'n':
            continue
        else:
            rating_one_movie = {'userId':userID,'movieId':movie['movieId'].values[0],'rating':rating}
            rating_list.append(rating_one_movie)
            num -= 1
    return rating_list

# A function to rank the movies in the database based on the user's rating on presented movies
def rank_movies(df, user_rating):
        ## add the new ratings to the original ratings DataFrame
    user_ratings = pd.DataFrame(user_rating)
    new_ratings_df = pd.concat([df, user_ratings], axis=0)
    reader = Reader()
    new_data = Dataset.load_from_df(new_ratings_df, reader)

    # train a model using the new combined DataFrame
    svd_ = SVD(n_factors= 50, reg_all=0.05)
    svd_.fit(new_data.build_full_trainset())

    # make predictions for the user
    list_of_movies = []
    for m_id in movie_rating_df['movieId'].unique():
        list_of_movies.append( (m_id,svd_.predict(1000,m_id)[3]))
    # Order the predictions from highest to lowest rated
    ranked_movies = sorted(list_of_movies, key=lambda x:x[1], reverse=True)
    return ranked_movies


# A function to recommend movies to the user
def recommended_movies(user_ratings, movie_title_df, n):
    recommended_movies_set = set()  # Keep track of recommended movies

    for idx, rec in enumerate(user_ratings):
        movie_id = int(rec[0])
        title_array = movie_title_df.loc[movie_title_df['movieId'] == movie_id, 'title'].values

        # Check if the array is not empty and the movie has not been recommended before
        if title_array.any() and title_array[0] not in recommended_movies_set:
            title = title_array[0]
            print('Recommendation #', idx+1, ':', title, '\n')
            recommended_movies_set.add(title)  # Add the movie to the set of recommended movies
            n -= 1

        if n == 0:
            break
```

This code utilizes user ratings to generate personalized movie recommendations using the collaborative filtering model.



```python
# Run the rank_movies function
ranked_m_ovies = Modeling.rank_movies(user_item_df, user_rating)

# run the recommender function
Modeling.recommended_movies(ranked_m_ovies, movie_rating_df, 5)
```

```
Recommendation # 1 : Shawshank Redemption, The (1994)

Recommendation # 2 : Rear Window (1954)

Recommendation # 3 : Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1964)

Recommendation # 4 : Fight Club (1999)

Recommendation # 5 : Godfather, The (1972)
```

- The recommended_movies function is used to display top movie recommendations based on the user's preferences and new ratings.

Insights from the Outputs

- The SVD model consistently outperforms the other models, justifying its selection as the final model.
- Hyperparameter tuning improves model performance.
- User ratings influence the movie recommendations, making the system personalized.

6. CONCLUSION

- The users are generally generous with their ratings as a majority of the rating are above 3.0. The most frequent rating was a rating of 4.0 with a bit over 25,000 movies.
- The second most frequent rating was 3.0 with about 20,000 movies.
- The least most common rating was 0.5
- The count plot above shows the most viewed movie titles. The top 3 movies are: Forrest Gump(1994),The Shawshank Redemption(1994) and Pulp Fiction(1994)
- The bar plot shows the distribution of all the movie genres in the dataset. The 3 top genres are listed below starting from the most common one:Drama,Comedy and Action
- The bar graph above shows the top 10 most viewed movies and there average ratings. The average ratings range from 3.5 to 4.5 for the top 10 movies.

- From the bar chart it can be seen that all the genres have an average rating of above 3. Film Noir has the highest average rating followed by Documentary.
- All the genres have most of their movies rated 4.0, with relatively fewer users giving a rating of either 1 and 2 to the movies.
- Drama, Comedy, Action, Thriller and Adventure appear to have the highest count of ratings indicating higher user engagement.
- Film Noir, Western and Documentary genres seem to have the lowest count of ratings indicating low user engagement

Grid search was performed to find optimal hyperparameters for the KNNBasic model.

- The best hyperparameters include 40 neighbors with cosine similarity for user-based collaborative filtering.
- The tuned RMSE for KNNBasic is 0.9724, which is slightly worse than the initial model from the build one which was 0.9721

Grid search for KNNBaseline is performed to fine-tune hyperparameters.

- The best hyperparameters include 30 neighbors with cosine similarity for user-based collaborative filtering.
- The tuned RMSE for KNNBaseline is 0.8779, further improving the model from the baseline build which had 0.8790

Grid search is performed to find optimal hyperparameters for the KNNWithMeans model.

- The best hyperparameters include 40 neighbors with Pearson similarity for user-based collaborative filtering.
- The tuned RMSE for KNNWithMeans is 0.8953, showing improvement over the initial model which had 0.8995.

Grid search is performed to find optimal hyperparameters for the SVD model.

- The best hyperparameters include 100 n-factors and a regularization term of 0.05.
- The RMSE for the tuned SVD is 0.8691, slightly improving the model.

7. RECOMMENDATIONS

Filamu should consider displaying the following movies as the most popular films in the their database: Forrest Gump(1994), Hoop Dreams(1994), Pulp Fiction(1994)

Filamu should consider having more movies in the following four genres: Drama, Comedy, Action and Thriller, since they are the most popular.

When deployed, the developed recommendation system will assist Filamu in suggesting films that users might like. This is due to the fact that the system will display to the users films similar to those they have already watched and enjoyed, as well as films that like users have enjoyed.

8. NEXT STEPS

Continuously monitor and retrain the deployed model with new data to ensure consistent predictive quality in real-world scenarios.

Enhance scalability by implementing distributed computing frameworks like Spark to handle large datasets more efficiently.

Create more comprehensive user and item profiles by incorporating features such as descriptions, demographics, and historical behavior.

9. DEPLOYMENT

The sytem was deployed using streamlit.