AI 705 - Recommendation Systems

Movie Recommendation System

Yash Koushik (IMT2020033)
Surya Sastry (IMT2020079)
Dharmin (IMT2020127)
Karan Raj Pandey (IMT2021014)

Problem Statement

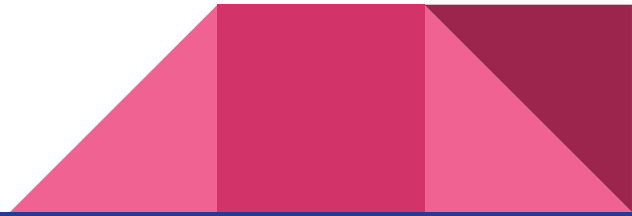
- The objective of this project is to build a movie recommendation system, where based on the existing reviews of a user, we should suggest new movies that the user will like. The dataset chosen for this project is the MovieLens 1M dataset.
- We are expected to implement the above using K-Means, SVD, and Collaborative Filtering using only numpy and pandas (from scratch).

About the Dataset: Movielens 1M

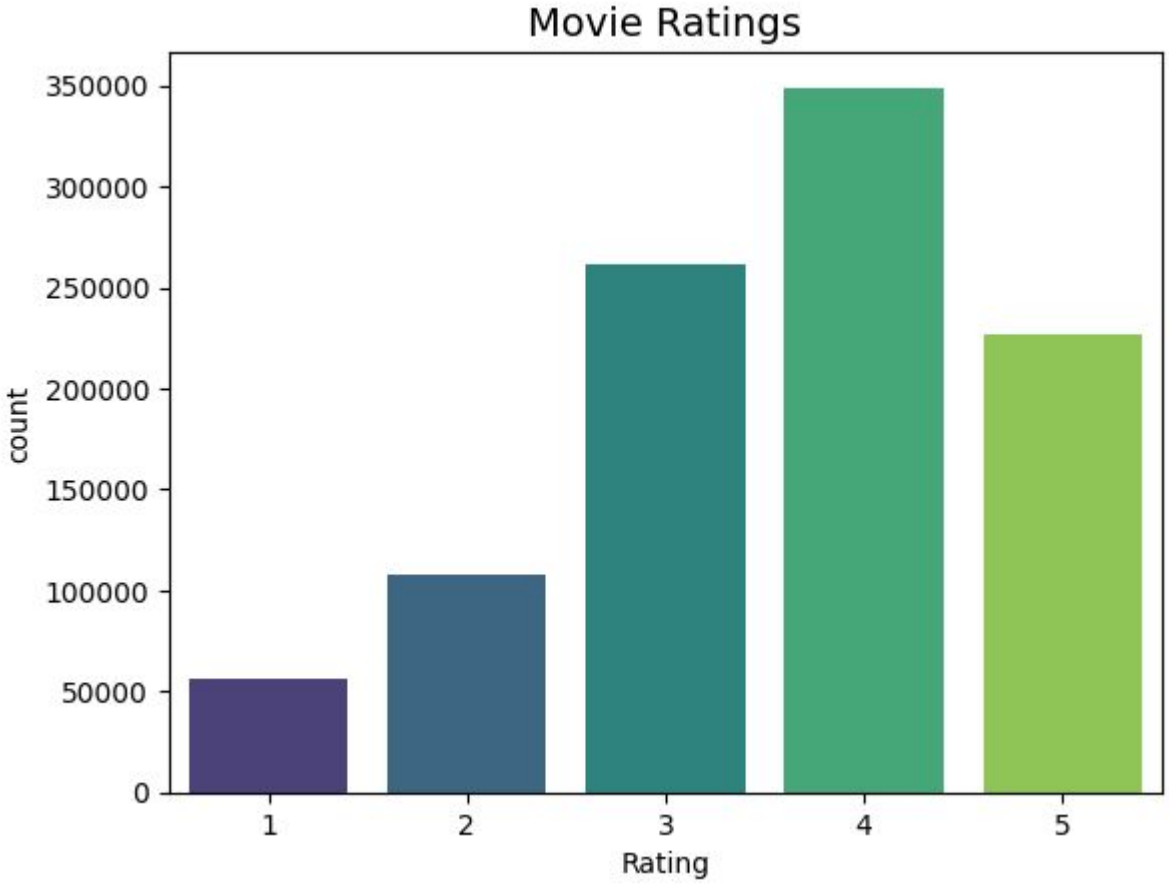
- The MovieLens dataset consists of approximately 1 million ratings of 4000 movies by over 6000 users on IMDB.
- The dataset is divided into three files, users.dat, movies.dat and ratings.dat.
- Ratings.dat consists of the ratings given by various users, users.dat consists of additional information about each user (Gender, Age, Occupation, Zipcode), and movies.dat consists of additional information about each movie (Title, Genre).

Exploratory Data Analysis

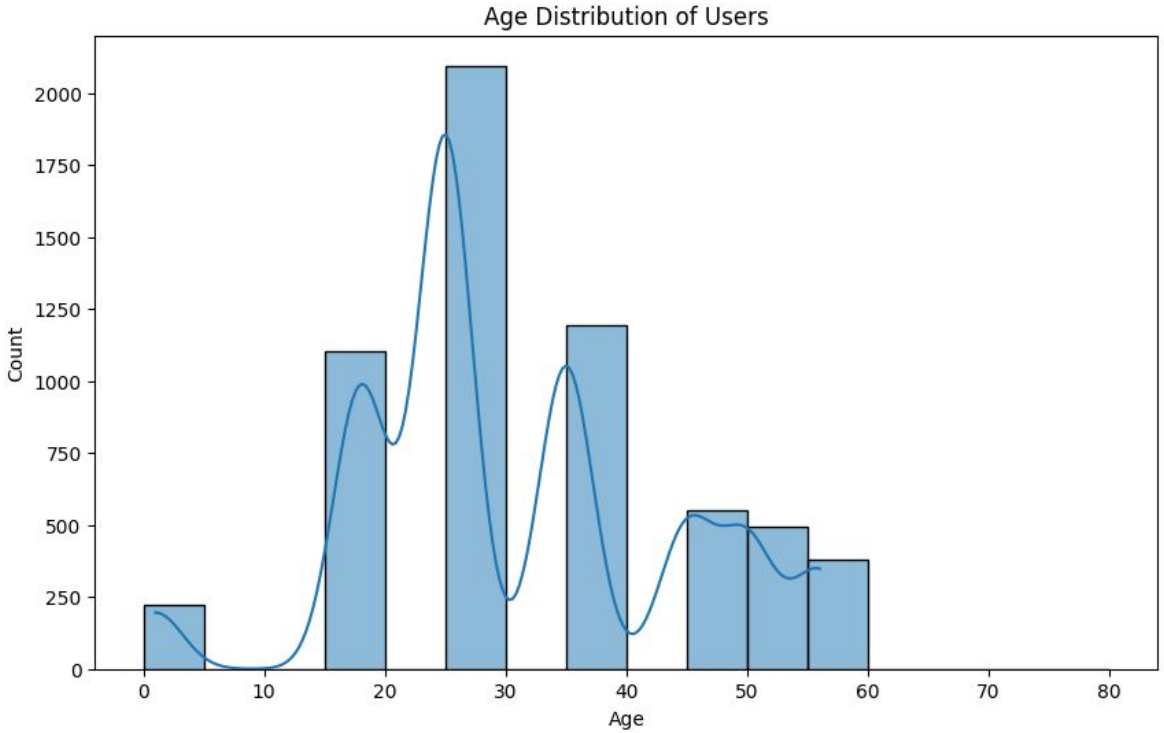
- Prior to constructing a robust recommendation model, it is crucial to thoroughly explore and comprehend the dataset along with its complexities.
- Exploratory Data Analysis (EDA) enables us to gain insights into patterns and relationships within the dataset. Furthermore, EDA offers explainability regarding the functionality and effectiveness of recommender models.



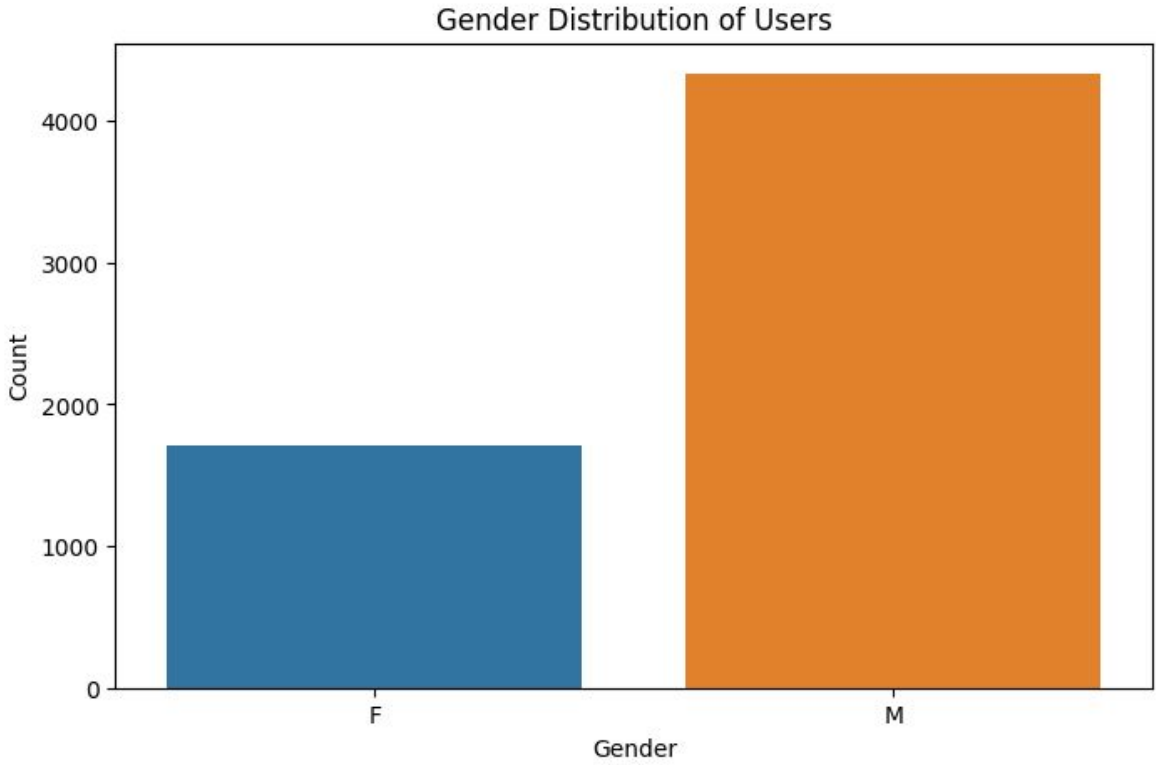
The plot shows the distribution of movie ratings, indicating that a significant portion of ratings are concentrated at higher values, suggesting a general positive sentiment towards movies in the dataset.



The age distribution of users indicates a predominantly younger demographic, with a peak in the late teens to early thirties. There's a decline in user counts with increasing age beyond the early forties.



The plot illustrates the gender distribution among users, showcasing the relative counts of male and female users. It suggests that the dataset has a larger representation of male over the other.



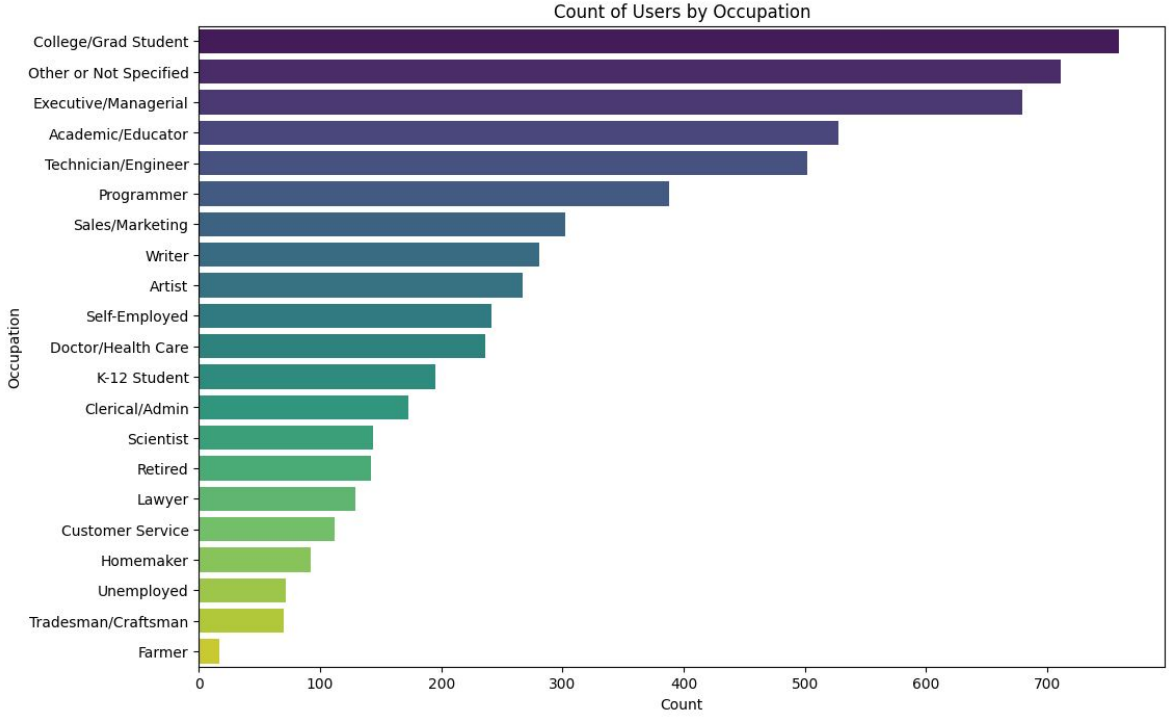
From the plot, it can be inferred that the most common occupations among users in the dataset are,

- College/Grad Student
- Other or Not Specified
- Executive/Managerial

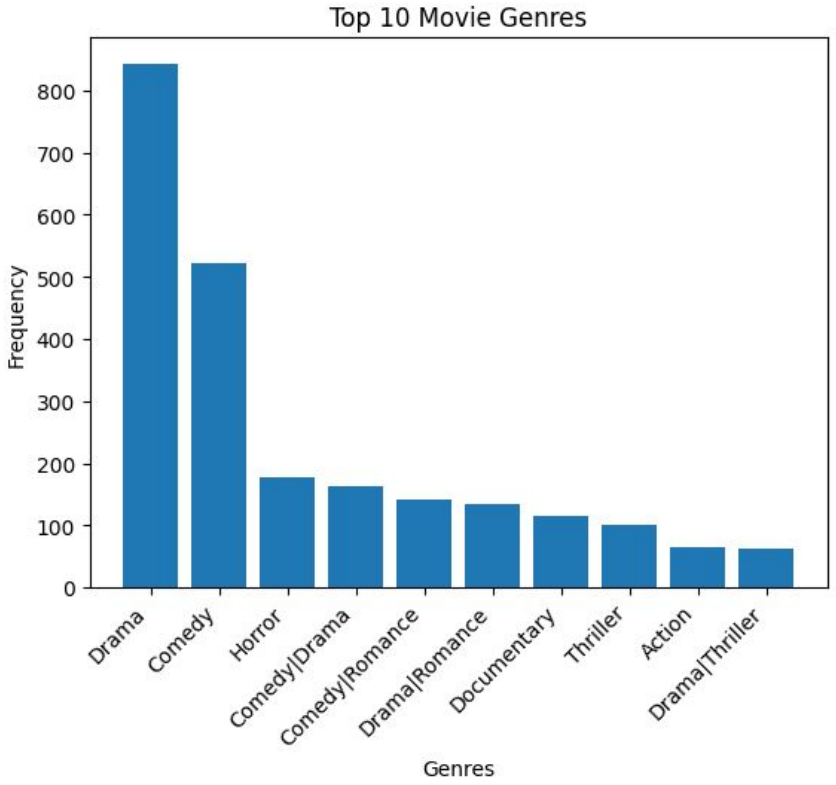
while occupations such as

- Homemaker
- Farmer

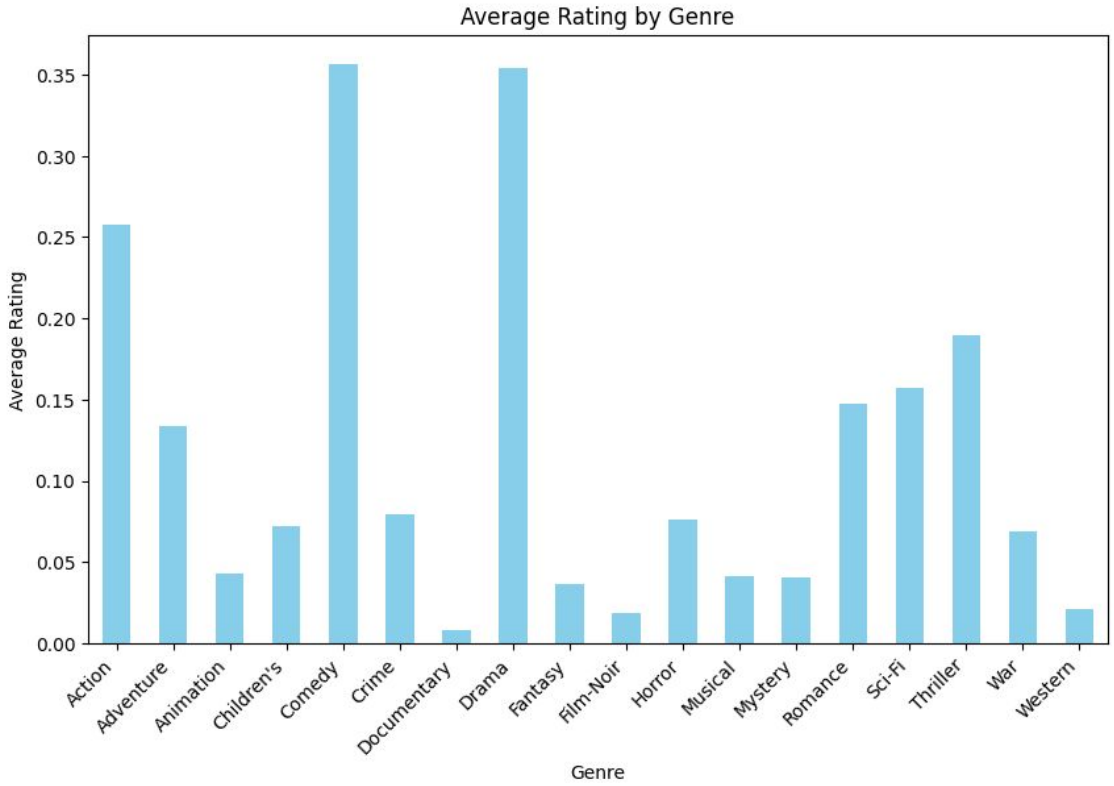
are relatively less common.



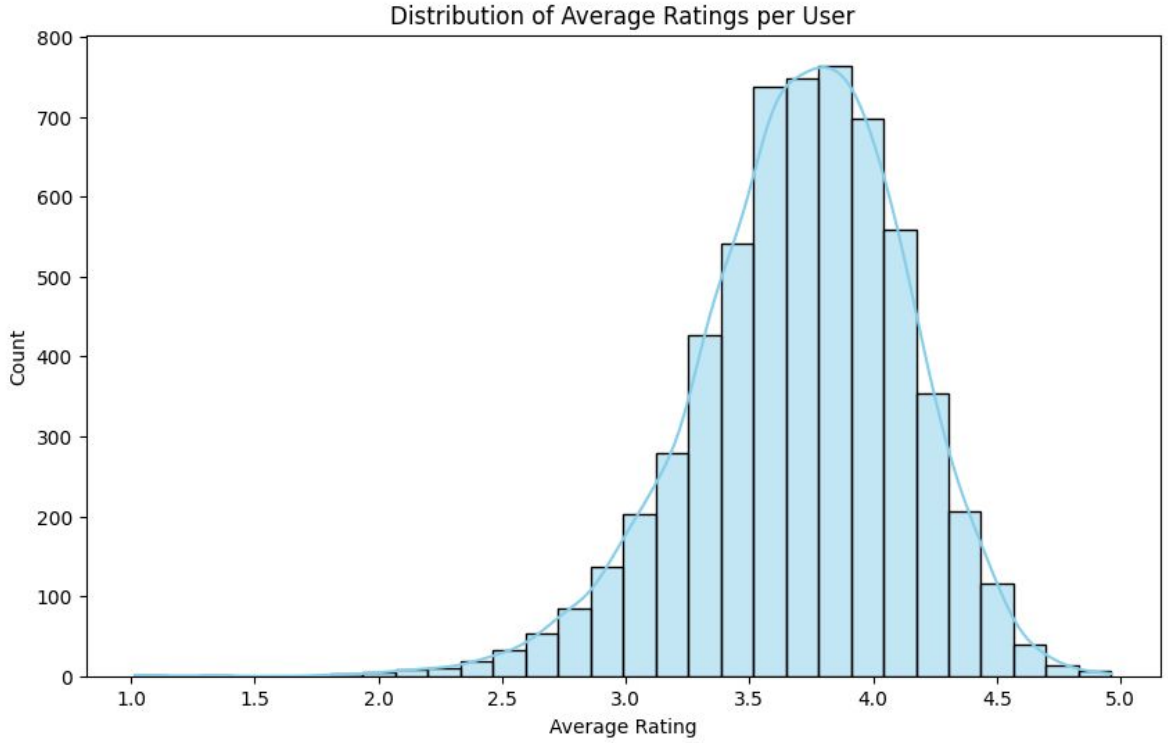
The bar chart shows the frequency distribution of the top 10 movie genres. It indicates that the most common genres among the movies in the dataset are displayed, with Drama being the most prevalent followed by Comedy and Horror.



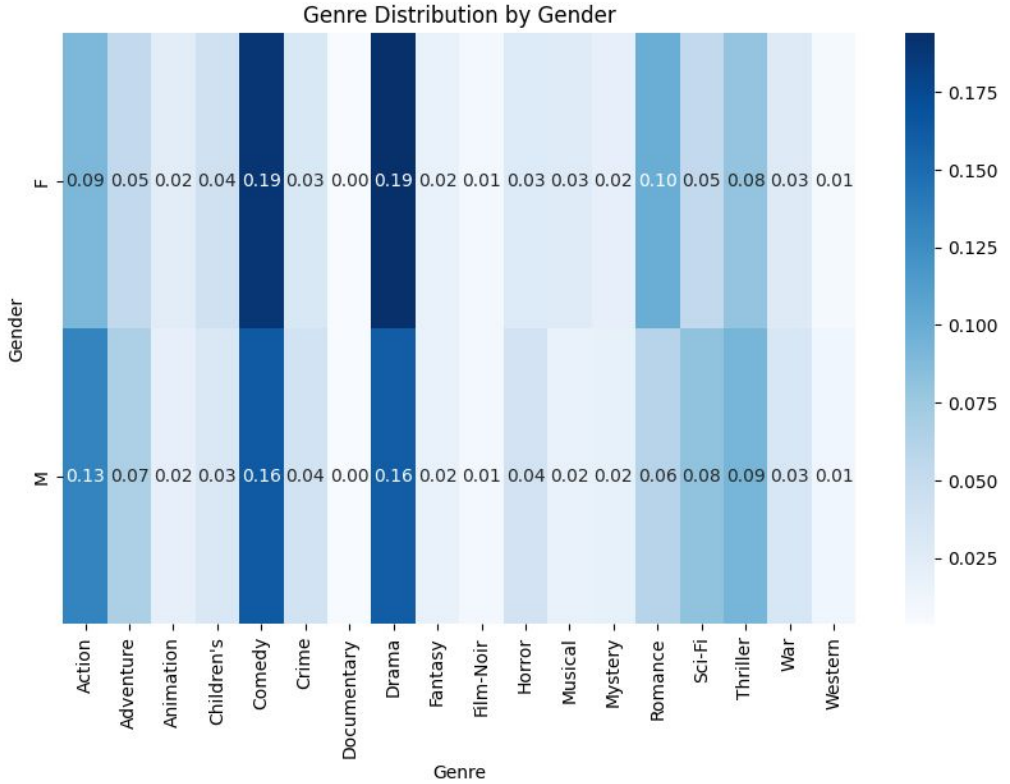
From the plot, we can infer that genres like Comedy, Drama, and Action tend to have higher average ratings compared to others, while genres like Documentary, Film-Noir and Western have relatively lower average ratings. Additionally, Adventure and Romance genres also show moderate average ratings.



- From the plot, we can infer that the majority of users tend to give higher ratings, as indicated by the peak around the higher average rating values.
- Additionally, there is a right-skewed distribution, suggesting that there are fewer users who give lower average ratings.
- This indicates a positive skewness in user rating behavior towards higher ratings.

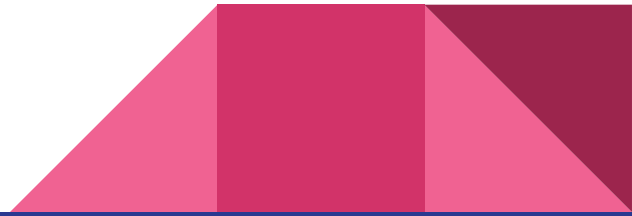


- Both genders show a preference for genres like Drama, Comedy, and Action.
- Females tend to have a higher proportion of Romance and Animation preferences compared to males.
- Males have a relatively higher interest in genres like Thriller and Sci-Fi compared to females.



Helper methods used in our Recommendation System

- K-Means:
  - Centroid Initialisation is done using 3 techniques:
    - Initialised randomly
    - K-Means plus plus
    - Naive Sharding
- Singular Value Decomposition (SVD):
  - All the methods are developed from scratch
  - We have also implemented reduced SVD from scratch.
- These methods are later used in our recommendation system.



Evaluation Metrics

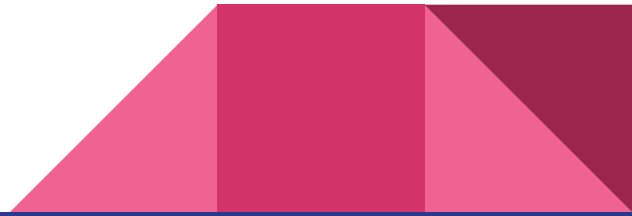
- For SVD+K-Means we have used MSE between actual genres and recommended genres which would later on help us in recommending movies based on the genre.

```
[222]
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, y_pred_arr)
```

```
0.040515285178019 81
```

Different Approaches used in the Project

- SVD + K-Means:
- Collaborative Filtering:
- Hybrid Recommendation:
- Temporal Recommendation:



Preprocessing for SVD + K-Means

- One-hot encoding of first digit of Zip-codes, which can act as an user similarity metric since we can recommend it area-wise.
- One-hot encoding of Occupation which can be used for user similarity metric too
- Standardization of Age so that it is easy to deal with skewed data
- Normalisation of Ratings so that it is beneficial when using k-means clustering algorithm.

SVD + K-Means

- For recommendations using SVD + K-Means, we made the dataset as a table with UserID as the index and genre, one hot encoding of zip-code, age, occupation and label-encoding of gender as the columns. There are 18 individual genres in the dataset.
- Each entry in this matrix is the user’s average rating for movies of that particular genre with UserID as the index.

Dataframe

```
[26]
df
```

Python

UserID	Gender	Age	Zip-code_1	Zip-code_2	Zip-code_3	Zip-code_4	Zip-code_5	Zip-code_6	Zip-code_7	Zip-code_8	...	Action	1940s	Animation	Comedy	Thriller	Mystery	1980s	Children's	Film-Noir	Drama
1	0	0.017857	0	0	0	1	0	0	0	0	...	[4.0, 5]	[2.5, 3]	[14.0, 18]	[11.0, 14]	[2.0, 3]	[]	[8.0, 10]	[16.25, 20]	[]	[18.0, 21]
2	1	1.000000	0	0	0	0	0	0	1	0	...	[35.0, 56]	[1.5, 2]	[]	[16.0, 25]	[19.25, 31]	[1.75, 3]	[15.25, 21]	[]	[0.75, 1]	[57.25, 79]
3	1	0.446429	0	0	0	0	1	0	0	0	...	[17.0, 23]	[]	[2.25, 3]	[20.75, 30]	[3.5, 5]	[0.5, 1]	[14.75, 20]	[2.25, 3]	[]	[6.0, 8]
4	1	0.803571	0	1	0	0	0	0	0	0	...	[15.0, 19]	[]	[]	[]	[2.5, 4]	[]	[4.25, 7]	[0.75, 1]	[]	[4.75, 6]
5	1	0.446429	0	0	0	0	1	0	0	0	...	[12.5, 31]	[1.0, 1]	[3.0, 4]	[33.75, 56]	[18.0, 39]	[4.25, 8]	[2.5, 7]	[4.25, 6]	[2.25, 3]	[54.5, 104]
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6036	0	0.446429	0	0	1	0	0	0	0	0	...	[77.0, 154]	[14.25, 18]	[24.75, 34]	[143.75, 261]	[78.75, 147]	[20.5, 34]	[113.5, 193]	[33.0, 54]	[13.0, 17]	[233.0, 372]
6037	0	0.803571	0	0	0	0	0	0	1	0	...	[18.5, 28]	[10.0, 14]	[0.75, 1]	[38.0, 59]	[46.0, 68]	[8.75, 13]	[26.25, 39]	[4.0, 6]	[5.5, 9]	[70.5, 98]
6038	0	1.000000	1	0	0	0	0	0	0	0	...	[1.0, 2]	[]	[2.0, 3]	[8.5, 12]	[]	[]	[4.0, 5]	[0.5, 1]	[]	[6.5, 9]
6039	0	0.803571	1	0	0	0	0	0	0	0	...	[6.0, 8]	[11.5, 15]	[8.5, 13]	[44.25, 65]	[11.0, 14]	[13.5, 17]	[9.75, 14]	[10.75, 17]	[5.25, 6]	[21.0, 28]
6040	1	0.446429	1	0	0	0	0	0	0	0	...	[20.75, 42]	[11.25, 12]	[2.0, 4]	[58.0, 102]	[30.0, 41]	[9.5, 11]	[67.75, 143]	[4.5, 6]	[7.5, 10]	[130.5, 185]

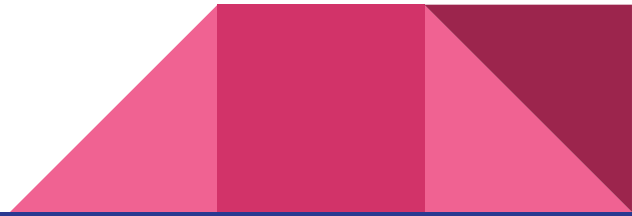
6040 rows × 60 columns

Preprocessing and Novelty

We also took the sum of ratings and number of ratings, as a tuple for each user which is represented by each entry.

Novelty:-

Incorporating zip-code, age and occupation as additional columns for predicting the movies is the novelty.



Preprocessing

NaN depicts that user hasn’t seen a single movie of that genre.

We replaced the NaN values using the mean of that group.

UserID	Gender	Age	Zip-code_1	Zip-code_2	Zip-code_3	Zip-code_4	Zip-code_5	Zip-code_6	Zip-code_7	Zip-code_8	...	Action	1940s	Animation	Comedy	Thriller	Mystery	1980s	Children's	Film-Noir	Drama
1	0	0.017857	0	0	0	1	0	0	0	0	...	0.8	0.833333	0.777778	0.785714	0.666667	NaN	0.8	0.8125	NaN	0.85714
2	1	1.000000	0	0	0	0	0	0	1	0	...	0.625	0.75	NaN	0.64	0.620968	0.583333	0.72619	NaN	0.75	0.72468
3	1	0.446429	0	0	0	0	1	0	0	0	...	0.73913	NaN	0.75	0.691667	0.7	0.5	0.7375	0.75	NaN	0.7
4	1	0.803571	0	1	0	0	0	0	0	0	...	0.789474	NaN	NaN	NaN	0.625	NaN	0.607143	0.75	NaN	0.79166
5	1	0.446429	0	0	0	0	1	0	0	0	...	0.403226	1.0	0.75	0.602679	0.461538	0.53125	0.357143	0.708333	0.75	0.52403
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6036	0	0.446429	0	0	1	0	0	0	0	0	...	0.5	0.791667	0.727941	0.550766	0.535714	0.602941	0.588083	0.611111	0.764706	0.62634
6037	0	0.803571	0	0	0	0	0	0	1	0	...	0.660714	0.714286	0.75	0.644068	0.676471	0.673077	0.673077	0.666667	0.611111	0.71938
6038	0	1.000000	1	0	0	0	0	0	0	0	...	0.5	NaN	0.666667	0.708333	NaN	NaN	0.8	0.5	NaN	0.72222
6039	0	0.803571	1	0	0	0	0	0	0	0	...	0.75	0.766667	0.653846	0.680769	0.785714	0.794118	0.696429	0.632353	0.875	0.7
6040	1	0.446429	1	0	0	0	0	0	0	0	...	0.494048	0.9375	0.5	0.568627	0.731707	0.863636	0.473776	0.75	0.75	0.70540

6040 rows × 60 columns

SVD

To understand composition and to spot trends we use SVD as SVD allows us to extract and untangle information. We also used Reduced SVD.

```
num_cluster=32
to_remove=9
threshold=0
```

```
from SVD import *
u,s,sigma,V_trans = ReducedSVD(df.to_numpy(),threshold,num_cluster)
df = pd.DataFrame(u@sigma@V_trans,columns=df.columns)
```

SVD+K-Means recommendation

These are the final recommendations after applying the SVD+K-Means on the train test split data.

```
[234]
top_movies=list(movies['Title'])[:5]
print("Top 5 movies are:")
for i in top_movies:
    print(i)
```

```
Top 5 movies are:
Star Wars: Episode V - The Empire Strikes Back (1980)
L.A. Confidential (1997)
Diva (1981)
Lady Vanishes, The (1938)
Transformers: The Movie, The (1986)
```

Error Analysis

We predicted the genres first and applied MSE on the genres, and used Affinity for predicting movies. As a novelty we even included years as a genre for recommending.

```
Top 5 PREDICTED recommended genres are:
1930s
1920s
1960s
War
1950s
```

```
Top 5 ACTUAL recommended genres are:
1930s
2000s
1950s
1920s
1940s
```

Preprocessing for Collaborative Filtering

- **Bayesian Average**:

$$r_i = (C \times m + \Sigma_{reviews}) / C + N$$

- We use Bayesian Average so as to prevent the less frequently rated movies form gaining the top spot.
- **Pearson's Correlation**: Pearson correlation offers advantages over cosine similarity by considering mean and standard deviation of ratings, addressing rating scale variations and user biases.

	similarityIndex	UserID	MovieID	Rating	weightedRating
0	0.082541	710	2987	5	0.412705
1	0.082541	710	1248	4	0.330164
2	0.082541	710	719	3	0.247623
3	0.082541	710	3937	3	0.247623
4	0.082541	710	3793	4	0.330164

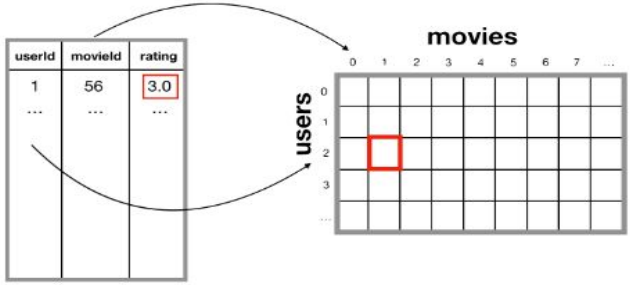
Handling The Cold Start Problem

- Collaborative filtering relies solely on user-item interactions within the utility matrix. The issue with this approach is that brand new users or items with no interactions get excluded from the recommendation system.
- We convert the **genres** column into binary features. Each genre will have its own column in the dataframe, and will be populated with 0 or 1.

	Comedy	Thriller	Animation	Musical	Mystery	Action	Adventure	Documentary	Sci-Fi	Romance	War	Drama	Horror	Film-Noir	Children's	Western	Fantasy	Crime
0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1	0
2	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- This matrix is populated with values between 0 and 1 which represent the degree of similarity between movies along the x and y axes.

Collaborative Filtering



- First we transform our dataframe into **user-item** matrix.
- **Item-Item** recommendation using *k-nearest neighbours:* We are going to find the *k* movies that have the most similar user engagement vectors for movie *i*.
- Evaluating sparsity: Dividing the number of stored elements by total number of elements.
- Output:

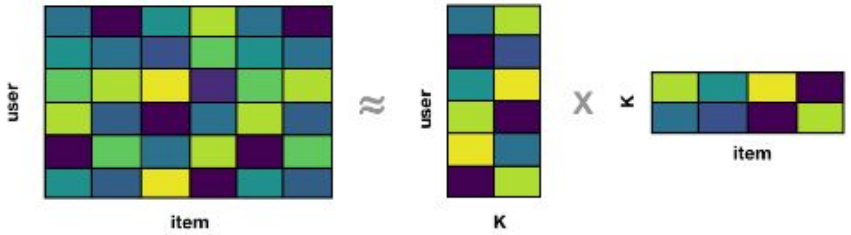
```
Because you watched Toy Story (1995):
Aladdin (1992)
Toy Story 2 (1999)
Lion King, The (1994)
Bug's Life, A (1998)
Beauty and the Beast (1991)
Wayne's World (1992)
Pleasantville (1998)
Babe (1995)
Mask, The (1994)
```

Dimensionality Reduction with Matrix Factorization

- Matrix factorization (MF) uncovers latent features in user-movie interactions, improving recommendation quality, especially for sparse data, by decomposing the user-item matrix into two factors.
  - user-factor matrix (n_users, k)
  - item-factor matrix (k, n_items)

- We reduced the dimensions down to n_components=20. We can think of each component representing a latent feature such as movie genre.

$$X_{mn} \approx P_{mk} \times Q_{nk}^T = \hat{X}$$




```
Because you watched Leaving Las Vegas (1995):
Dead Man Walking (1995)
Sling Blade (1996)
Shine (1996)
Crying Game, The (1992)
What's Eating Gilbert Grape (1993)
Insider, The (1999)
Some Folks Call It a Sling Blade (1993)
Malcolm X (1992)
Thelma & Louise (1991)
```

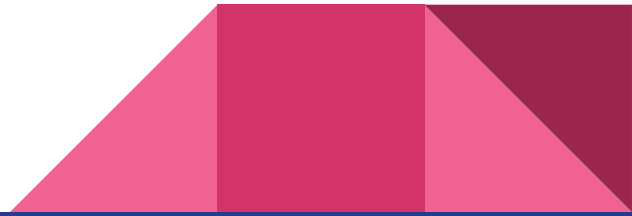
Novelties implemented

- HYBRID RECOMMENDATION

  It combines both collaborative filtering and content-based filtering techniques to generate movie recommendations based on a given movie title.

- TEMPORAL RECOMMENDATION

  It generates movie recommendations based on the current time of day and predefined temporal preferences for different genres.



HYBRID RECOMMENDATION

- This function, hybrid_recommendations, takes a movie title string and the number of recommendations to generate as input parameters.
- It first finds the closest match to the input movie title using movie_finder.
- Then, it retrieves the index of the movie from the movie_idx dictionary.
- Next, it computes collaborative filtering recommendations based on movie similarities (corr) using cosine similarity.
- Similarly, it computes content-based recommendations using the same cosine similarity matrix (corr) but sorts based on content similarity.
- Finally, it prints the recommendations from both approaches.

```
def hybrid_recommendations(title_string, n_recommendations=10):
    title = movie_finder(title_string, all_titles)
    idx = movie_idx[title]
    # Collaborative filtering
    sim_scores_collab = list(enumerate(corr[idx]))
    sim_scores_collab = sorted(sim_scores_collab, key=lambda x: x[1], reverse=True)
    sim_scores_collab = sim_scores_collab[1:(n_recommendations+1)]
    similar_movies_collab = [i[0] for i in sim_scores_collab]

    # Content-based filtering
    sim_scores_content = list(enumerate(corr[idx]))
    sim_scores_content = sorted(sim_scores_content, key=lambda x: x[1], reverse=True)
    sim_scores_content = sim_scores_content[1:(n_recommendations+1)]
    similar_movies_content = [i[0] for i in sim_scores_content]
```

OUTPUT:

- The function prints the given movie title and the recommendations separately for collaborative filtering and content-based filtering.

- It returns the recommended movie titles for both filtering approaches.

```
    print(f"Because you watched {title}:")
    print("Collaborative filtering recommendations:")
    print(df1['Title'].iloc[similar_movies_collab])
    print("\nContent-based filtering recommendations:")
    print(df1['Title'].iloc[similar_movies_content])

hybrid_recommendations('Toy Story 2', 5)
```

```
Because you watched Toy Story 2 (1999):
Collaborative filtering recommendations:
1050        Aladdin and the King of Thieves (1996)
2072                        American Tail, An (1986)
2073    American Tail: Fievel Goes West, An (1991)
2285                      Rugrats Movie, The (1998)
2286                           Bug's Life, A (1998)
Name: Title, dtype: object

Content-based filtering recommendations:
1050        Aladdin and the King of Thieves (1996)
2072                        American Tail, An (1986)
2073    American Tail: Fievel Goes West, An (1991)
2285                      Rugrats Movie, The (1998)
2286                           Bug's Life, A (1998)
Name: Title, dtype: object
```

TEMPORAL RECOMMENDATION

- This function retrieves the current time of day by accessing the hour from the current system time.
- Based on the hour, it categorizes the time into four periods: morning, afternoon, evening, and night.
- It returns the time of day as a string.

```python
from datetime import datetime

def get_time_of_day():
    current_time = datetime.now()
    hour = current_time.hour
    if 6 <= hour < 12:
        return 'morning'
    elif 12 <= hour < 18:
        return 'afternoon'
    elif 18 <= hour < 22:
        return 'evening'
    else:
        return 'night'
```

- This function, recommend_temporal, takes a user ID and the movies DataFrame as input parameters.
- It fetches the current time of day using the get_time_of_day function.
- Based on the time of day, it selects the predefined genre preferences from the temporal_preferences dictionary.
- Then, it filters movies from the movies DataFrame (df1) based on the preferred genres for the given time of day.
- Finally, it samples 10 recommended movies from the filtered list and returns them.

```python
def recommend_temporal(user_id, movies_df):
    # Get the current time of day
    time_of_day = get_time_of_day()

    # Define temporal preferences based on time of day
    temporal_preferences = {
        'morning': ['Drama', 'Romance', 'Comedy'],
        'afternoon': ['Action', 'Adventure', 'Thriller'],
        'evening': ['Drama', 'Comedy', 'Romance'],
        'night': ['Horror', 'Thriller', 'Mystery']
    }

    # Get genre preferences for the given time of day
    preferred_genres = temporal_preferences.get(time_of_day, [])

    # Filter movies based on preferred genres
    recommended_movies = df1[df1['Genres'].apply(lambda x: any(genre in x for genre in preferred_genres))]

    # Return a sample of recommended movies
    return recommended_movies.sample(n=10)
```

OUTPUT

- The function returns a DataFrame containing the recommended movies for the current time of day.
- The DataFrame includes columns for movie title and genres.
- These recommendations are based on the predefined temporal preferences for different times of the day.

```
Temporal Recommendations for the current time of day:
                                    Title                        Genres
2432                      October Sky (1999)                     [Drama]
1889                 Terms of Endearment (1983)             [Comedy, Drama]
2473  Lock, Stock & Two Smoking Barrels (1998)   [Comedy, Crime, Thriller]
582                         Home Alone (1990)         [Children's, Comedy]
3328           Great Muppet Caper, The (1981)         [Children's, Comedy]
2200                 Indecent Proposal (1993)                     [Drama]
1053              Damsel in Distress, A (1937)  [Comedy, Musical, Romance]
2173                 Grandview, U.S.A. (1984)                     [Drama]
2507                       Love, etc. (1996)                      [Drama]
1199                            Ran (1985)                [Drama, War]
```

```python
# Example usage of recommend_temporal
user_id = 1
recommended_temporal_movies = recommend_temporal(user_id, df1)
print(f"Temporal Recommendations for the current time of day:")
print(recommended_temporal_movies[['Title', 'Genres']])
```

Thank You