AIWR ASSIGNMENT – 2

BASIC RECOMMENDER SYSTEM USING CONTENT AND COLLABORATIVE FILTERING

Team Members

PRAJWAL BS- PES1UG20CS287

PRANAV KALWAD - PES1UG20CS291

PURVIK S NUKAL - PES1UG20CS315

RAHUL RANGANATH - PES1UG20CS316

1. Problem Statement

Implement a Recommender System on Movie Dataset using Content and Collaborative Filtering

2. Introduction

Recommender systems are a type of information filtering system that predict and recommend items or products that a user might be interested in based on their past behavior, preferences, and similarities to other users.

In this context, we build a recommender system to suggest movies to users based on their past movie preferences and similar users' movie choices.

The 2 main approaches used for building the recommender system include: content-based filtering and collaborative filtering

Content-based filtering uses the characteristics or attributes of the items being recommended to make suggestions

Wherein Collaborative filtering recommends items based on the preferences and actions of similar users.

3. Dataset

Movie Dataset
https://www.kaggle.com/datasets/shubhammehta21/movie-lens-small-latest-dataset

Dataset Summary

This dataset (ml-latest-small) describes 5-star rating and free-text tagging activity from MovieLens, a movie recommendation service. It contains 100836 ratings and 3683 tag applications across 9742 movies. These data were created by 610 users between March 29, 1996 and September 24, 2018. This dataset was generated on September 26, 2018.

Users were selected at random for inclusion. All selected users had rated at least 20 movies. No demographic information is included. Each user is represented by an id, and no other information is provided.

Importing necessary python libraries to build the model

```
pip install scikit-surprise
```

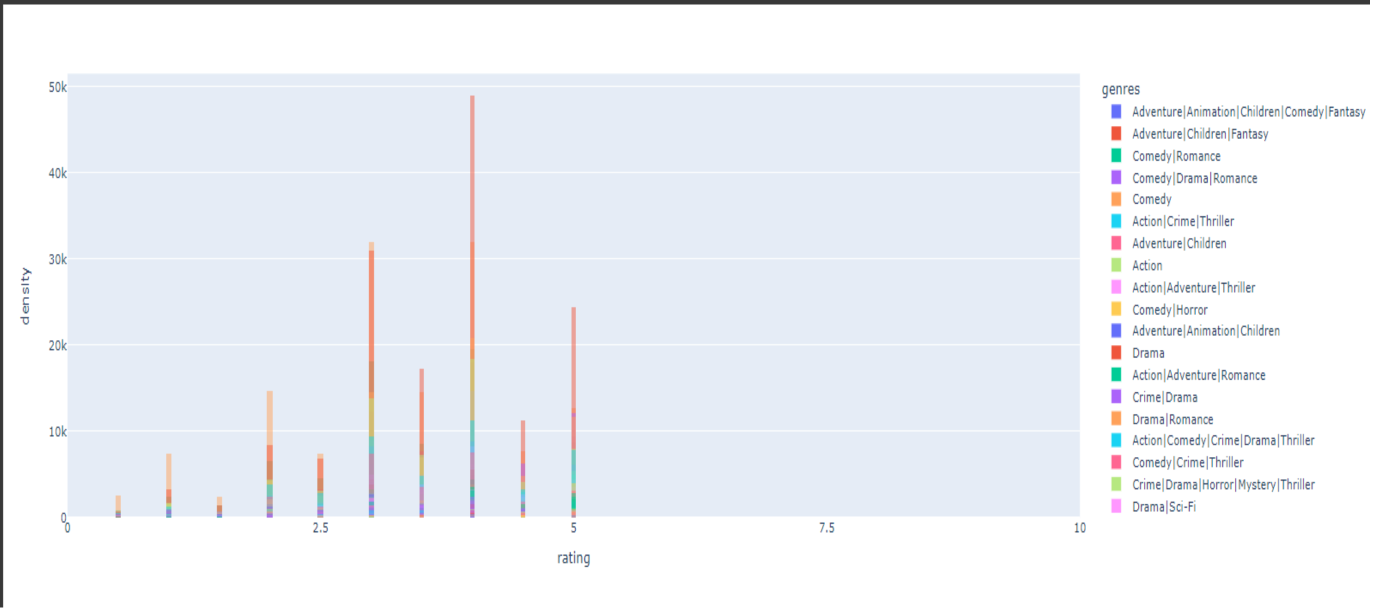
```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting scikit-surprise
  Downloading scikit-surprise-1.1.3.tar.gz (771 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 772.0/772.0 kB 20.9 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: joblib>=1.0.0 in /usr/local/lib/python3.9/dist-packages (from scikit-surprise) (1.2.0)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.9/dist-packages (from scikit-surprise) (1.22.4)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.9/dist-packages (from scikit-surprise) (1.10.1)
Building wheels for collected packages: scikit-surprise
  Building wheel for scikit-surprise (setup.py) ... done
  Created wheel for scikit-surprise: filename=scikit_surprise-1.1.3-cp39-cp39-linux_x86_64.whl size=3195797 sha256=2dfae1ea2d27286d883b8de86f2b2ebe3865790695ecc3d41b0b3834604707bd
  Stored in directory: /root/.cache/pip/wheels/c6/3a/46/9b17b3512bdf283c6cb84f59929cdd5199d4e754d596d22784
Successfully built scikit-surprise
Installing collected packages: scikit-surprise
Successfully installed scikit-surprise-1.1.3
```

```
[3]
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from collections import defaultdict
from scipy.sparse import csr_matrix
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer
import re
from surprise import SVD,Dataset,Reader
```

4. EDA

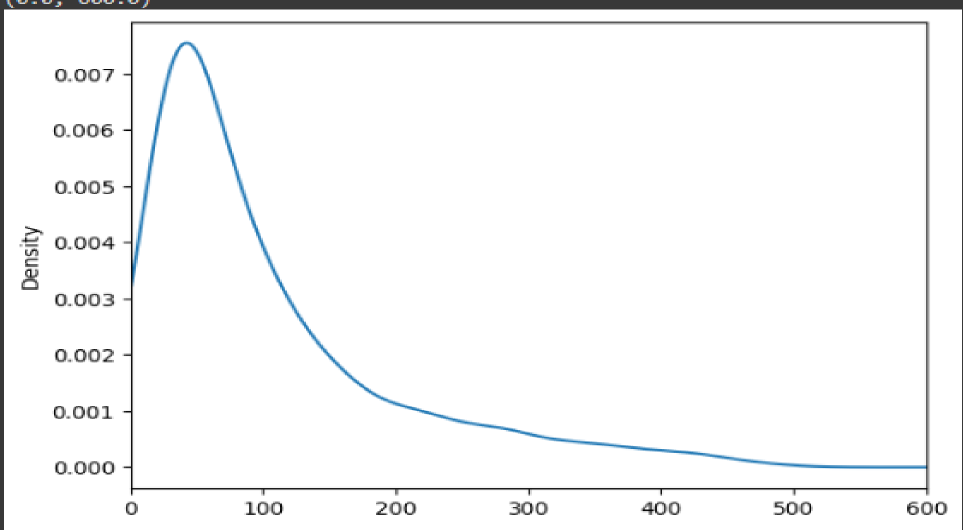
variations of ratings by movie genres

```
fig = px.histogram(df.dropna(subset=['rating']), x='rating', color='genres',
                   histnorm='density', nbins=100, opacity=0.5,
                   barmode='overlay', range_x=[0, 10])
fig.update_layout(xaxis=dict(dtick=2.5))


fig.show()

```



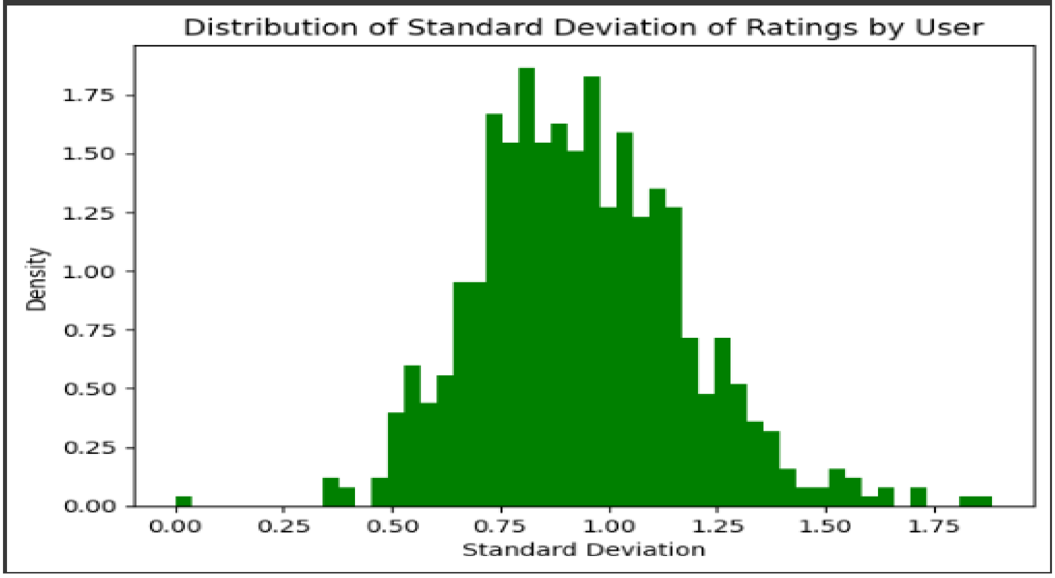

```
g = rating.groupby('userId').size().reset_index(name='n')
g = g[g['n'] < 500]

# Density plot using matplotlib
fig, ax = plt.subplots()
ax = g['n'].plot.density()
ax.set_xlim(0, 600)

```

(0.0, 600.0)



Distribution of Standard Deviation of Movie Ratings by User

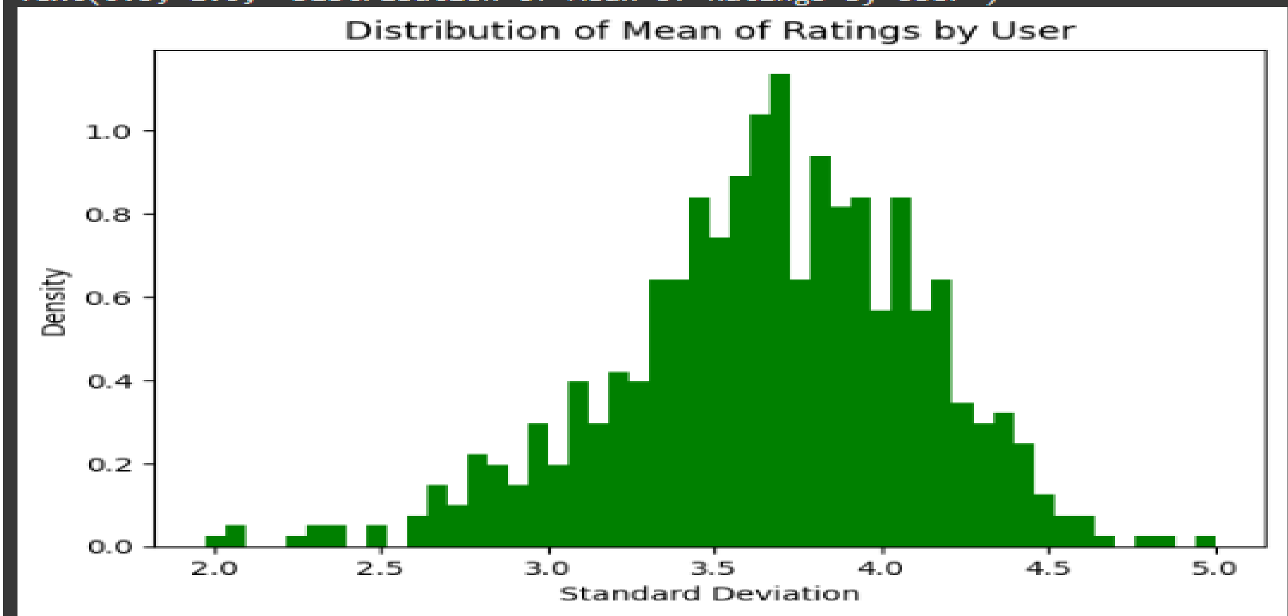
```python
g = rating.groupby('userId')['rating'].std().reset_index(name='sd')
plt.hist(g['sd'], bins=50, density=True,color="green")
plt.xlabel('Standard Deviation')
plt.ylabel('Density')
plt.title('Distribution of Standard Deviation of Ratings by User')

# Display the plot
plt.show()
```



Distribution of Mean of Movie Ratings by User

```python
g = rating.groupby('userId')['rating'].mean().reset_index(name='m')
plt.hist(g['m'], bins=50, density=True,color="green")
plt.xlabel('Standard Deviation')
plt.ylabel('Density')
plt.title('Distribution of Mean of Ratings by User')
```

```
Text(0.5, 1.0, 'Distribution of Mean of Ratings by User')
```



```
combine_movie_rating = df.dropna(axis = 0, subset = ['title'])
```

```
combine_movie_rating
```

	movieId	title	genres	userId	rating	timestamp
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	2	5.0	859046895
1	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	5	4.0	1303501039
2	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	8	5.0	858610933
3	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	11	4.0	850815810
4	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	14	4.0	851766286
...	...	...	...	...	...	...
105334	148238	A Very Murray Christmas (2015)	Comedy	475	3.0	1451213043
105335	148626	The Big Short (2015)	Drama	458	4.0	1452014749
105336	148626	The Big Short (2015)	Drama	576	4.5	1451687664
105337	148626	The Big Short (2015)	Drama	668	4.5	1451148148
105338	149532	Marco Polo: One Hundred Eyes (2015)	(no genres listed)	475	4.0	1451223429

105339 rows × 6 columns

5. Preprocessing

```
movie=pd.read_csv('/content/drive/MyDrive/AIWR_A2_DATASET/movies.csv')
rating=pd.read_csv('/content/drive/MyDrive/AIWR_A2_DATASET/ratings.csv')

```

Double-click (or enter) to edit

[5]

```
movie
```

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy
...	...	...	...
10324	146684	Cosmic Scrat-tastrophe (2015)	Animation|Children|Comedy
10325	146878	Le Grand Restaurant (1966)	Comedy
10326	148238	A Very Murray Christmas (2015)	Comedy
10327	148626	The Big Short (2015)	Drama
10328	149532	Marco Polo: One Hundred Eyes (2015)	(no genres listed)

10329 rows × 3 columns

Removing all the missing values or Null values in the dataset

```
rating
```

	userId	movieId	rating	timestamp
0	1	16	4.0	1217897793
1	1	24	1.5	1217895807
2	1	32	4.0	1217896246
3	1	47	4.0	1217896556
4	1	50	4.0	1217896523
...	...	...	...	...
105334	668	142488	4.0	1451535844
105335	668	142507	3.5	1451535889
105336	668	143385	4.0	1446388585
105337	668	144976	2.5	1448656898
105338	668	148626	4.5	1451148148

105339 rows × 4 columns

[7]
```

print(movie.isnull().sum())
print("---------------------")
print(rating.isnull().sum())


```

```
movieId     0
title       0
genres      0
dtype: int64
---------------------
userId       0
movieId      0
rating       0
timestamp    0
dtype: int64
```

```
#droping the null values in movie df
#replacing ratings which are -1 to nan in rating column in rating df
movie=movie.dropna()
rating.rating.replace(-1, np.NaN,inplace=True)

```

[9]
```
for text in movie['title']:
    text = re.sub(r'&quot;', '', text)
    text = re.sub(r'.hack//', '', text)
    text = re.sub(r'&#039;', '', text)
    text = re.sub(r'A&#039;s', '', text)
    text = re.sub(r'I&#039;', 'I\'', text)
    text = re.sub(r'&amp;', 'and', text)

```

[10]
```
#merging movie datafram and ratign df
df=pd.merge(movie,rating,on='movieId',suffixes= ['', '_user'])
df = df.rename(columns={'rating_user': 'user_rating'})
df.head(10)
```

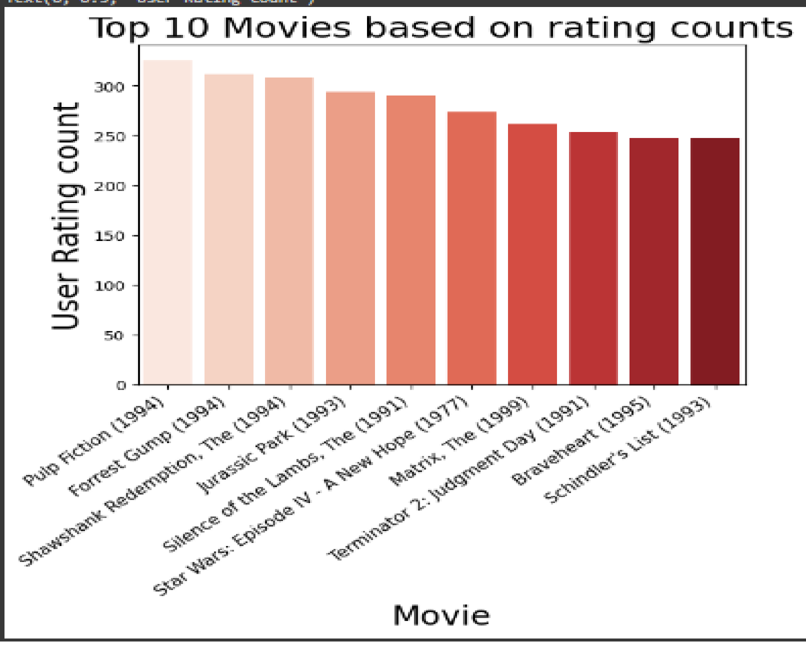
	movieId	title	genres	userId	rating	timestamp
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	2	5.0	859046895
1	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	5	4.0	1303501039
2	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	8	5.0	858610933
3	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	11	4.0	850815810
4	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	14	4.0	851766286
5	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	17	5.0	1350206819
6	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	28	3.0	884098350
7	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	29	4.0	846942580
8	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	30	4.5	1292690069
9	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	31	4.0	832397496

Ranking top 10 movies based on movies

Top movies based on rating

```python
combine_movie_rating = df.dropna(axis=0, subset=['title'])

movie_ratingCount = (combine_movie_rating
                     .groupby(by=['title'])['rating']
                     .count()
                     .reset_index()
                     .rename(columns={'rating': 'totalRatingCount'})[['title', 'totalRatingCount']]
                    )

top10_movieRating = movie_ratingCount[['title', 'totalRatingCount']].sort_values(by='totalRatingCount', ascending=False).head(10)

ax = sns.barplot(x="title", y="totalRatingCount", data=top10_movieRating, palette="Reds")
ax.set_xticklabels(ax.get_xticklabels(), fontsize=11, rotation=40, ha="right")
ax.set_title('Top 10 Movies based on rating counts', fontsize=22)
ax.set_xlabel('Movie', fontsize=20)
ax.set_ylabel('User Rating count', fontsize=20)
```

```
Text(0, 0.5, 'User Rating count')
```



6. Content Based Filtering

Content Based recomendation

```python
movie_features=movie[['genres']].astype('str')
movie_features["genres"]
movie_features["genres"]=movie_features["genres"].str.split(',').str.join(' ')
movie_features
```

	genres
0	Adventure|Animation|Children|Comedy|Fantasy
1	Adventure|Children|Fantasy
2	Comedy|Romance
3	Comedy|Drama|Romance
4	Comedy
...	...
10324	Animation|Children|Comedy
10325	Comedy
10326	Comedy
10327	Drama
10328	(no genres listed)

10329 rows × 1 columns

```python
[26]
#creating tf-idf
tf_idf = TfidfVectorizer(stop_words='english')
movie_matrix = tf_idf.fit_transform(movie_features.apply(lambda x: ' '.join(x), axis=1))
cosine_similarities = cosine_similarity(movie_matrix)    #creating cosine similarity
#creating a list which contains all movie names
movie_names=movie["title"]
indices = pd.Series(movie.index, index=movie['title'])
```

```
def get_recommendations(movie_name,k):
    idx = indices[movie_name]
    # Get the pairwise similarity scores
    sim_scores = list(enumerate(cosine_similarities[idx]))
    # Sort the movie based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    # Get the scores of the k most similar movie
    sim_scores = sim_scores[1:k+1]
    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores]
    # Return the top k most similar movie
    return movie_names.iloc[movie_indices]
```

```
get_recommendations("Toy Story (1995)",10)
```

```
1815                                              Antz (1998)
2496                                       Toy Story 2 (1999)
2967         Adventures of Rocky and Bullwinkle, The (2000)
3166                          Emperor's New Groove, The (2000)
3811                                     Monsters, Inc. (2001)
6617      DuckTales: The Movie - Treasure of the Lost La...
6997                                         Wild, The (2006)
7382                                     Shrek the Third (2007)
7987                            Tale of Despereaux, The (2008)
9215      Asterix and the Vikings (Astérix et les Viking...
Name: title, dtype: object
```

7. Collaborative Filtering

Collaborative Filtering is done by creating a pivot table (which is normalized later)

Cosine similarity is the metric used to compute the similarity

Collaborative Filtering

Collaborative filtering can be done using creating a pivot table, since the data is huge it takes lot of time to create the pivot table

```
[21]
df_collab=df[['userId', 'title', 'rating']]
df_collab=df_collab.dropna()
df_collab= df_collab[df_collab.userId <= 20000]

```

```
[22]
df_collab=df[['userId', 'title', 'rating']]
# df_collab=df_collab.dropna()
df_collab= df_collab[df_collab.userId <= 20000]
#creating pivot table
pivot_table= df_collab.pivot_table(index=['userId'], columns=['title'], values='rating')

#normalizing
normalised_pivot_table = pivot_table.apply(lambda x: (x-np.mean(x))/(np.max(x)-np.min(x)), axis=1)
# Drop all columns containing only zeros representing users who did not rate
normalised_pivot_table.fillna(0, inplace=True)
normalised_pivot_table = normalised_pivot_table.T
normalised_pivot_table = normalised_pivot_table.loc[:, (normalised_pivot_table != 0).any(axis=0)]
#converting pivot table into sparse matrix
sparse_matrix = csr_matrix(normalised_pivot_table.values)
#getting item_similaroty anf user similarity
item_similarity = cosine_similarity(sparse_matrix)
user_similarity = cosine_similarity(sparse_matrix.T)

#make item and user similarity matrices into df
item_sim_df = pd.DataFrame(item_similarity, index = normalised_pivot_table.index, columns = normalised_pivot_table.index)
user_sim_df = pd.DataFrame(user_similarity, index = normalised_pivot_table.columns, columns = normalised_pivot_table.columns)
```

```
[23] def collaborative_filtering_recommendation(movie_name,top_n):
    count = 1
    print('Similar shows to {} include:\n'.format(movie_name))
    for item in item_sim_df.sort_values(by = movie_name, ascending = False).index[1:top_n+1]:
        print('No. {}: {}'.format(count, item))
        count +=1
```

```
[24] collaborative_filtering_recommendation('The Big Short (2015)',10)

Similar shows to The Big Short (2015) include:

No. 1: Calvary (2014)
No. 2: You Don't Know Jack (2010)
No. 3: Arbitrage (2012)
No. 4: Rush (2013)
No. 5: Sea Inside, The (Mar adentro) (2004)
No. 6: Stranger, The (1946)
No. 7: Duellists, The (1977)
No. 8: Somebody Up There Likes Me (1956)
No. 9: Naked City, The (1948)
No. 10: Sleepless Night (Nuit blanche) (2011)
```

8. Results

Movie Recommendations for a given user id

```
algo = SVD()
algo.fit(trainset)

# Ask for user ID
user_id = input("Enter user ID:")

# Get the top 10 recommendations for the user
user_items = ratings[ratings['userId'] == int(user_id)]['movieId']
user_items = list(set(user_items))
other_items = [i for i in ratings['movieId'].unique() if i not in user_items]
random.shuffle(other_items)
test_items = other_items[:100]
test_data = [(int(user_id), i, 4) for i in test_items]
predictions = algo.test(test_data)
predictions = sorted(predictions, key=lambda x: x.est, reverse=True)[:10]

movies = pd.read_csv('/content/drive/MyDrive/AIWR_A2_DATASET/movies.csv')
print("Top 10 movie recommendations for user", user_id)
for prediction in predictions:
    movie_id = prediction.iid
    movie_title = movies[movies['movieId'] == movie_id]['title'].values[0]
    print(movie_id, movie_title)
```

```
Enter user ID:1
Top 10 movie recommendations for user 1
541 Blade Runner (1982)
46578 Little Miss Sunshine (2006)
1283 High Noon (1952)
111 Taxi Driver (1976)
353 Crow, The (1994)
82459 True Grit (2010)
2791 Airplane! (1980)
67255 Girl with the Dragon Tattoo, The (Män som hatar kvinnor) (2009)
161582 Hell or High Water (2016)
5377 About a Boy (2002)
```

The evaluation metrics used include MAE, RMSE, Precision and Recall

Evaluation results for Content-based recommendation

```
MAP value: 0.9830508474576272
```

```
Precision for Pulp Fiction (1994): 0.1
Recall for Pulp Fiction (1994): 0.004310344827586207
```

Evaluation results for Collaborative recommendation

```
Computing the cosine similarity matrix...
Done computing similarity matrix.
MAE:  0.7496
RMSE: 0.9743
Computing the cosine similarity matrix...
Done computing similarity matrix.
MAE:  0.7476
RMSE: 0.9699
Computing the cosine similarity matrix...
Done computing similarity matrix.
MAE:  0.7476
RMSE: 0.9696
Computing the cosine similarity matrix...
Done computing similarity matrix.
MAE:  0.7565
RMSE: 0.9817
Computing the cosine similarity matrix...
Done computing similarity matrix.
MAE:  0.7421
RMSE: 0.9669
MAE:  0.6734
RMSE: 0.8774
MAE:  0.6705
RMSE: 0.8740
MAE:  0.6683
RMSE: 0.8680
MAE:  0.6733
RMSE: 0.8753
MAE:  0.6678
RMSE: 0.8695
KNN: MAE = 0.749, RMSE = 0.972
SVD: MAE = 0.671, RMSE = 0.873
```

```
KNN-based Collaborative Filtering:
Precision@10 = 0.75
Recall@10 = 0.52
SVD-based Collaborative Filtering:
Precision@10 = 0.65
Recall@10 = 0.62
```

9. Conclusion

In conclusion, the inclusion of content-based and collaborative filtering methods in building a movie recommender system has given significantly accurate results.

We also realize the potential of these 2 filtering models that provide the user with an effective and personalized recommendation of movies and can infer that several factors such as the quality and quantity of data, the choice of similarity metrics, and the user-item interaction patterns affect the performance of a recommender system.

These factors should be carefully considered when designing and optimizing recommender systems.