AIWR

ASSIGNMENT - 02

Movie Recommender System

Team Members:

Name	SRN
Sanmat Sanjayakumar Payagoudar	PES1UG20CS385
Shamith V S	PES1UG20CS390
Monisha N	PES1UG21CS820
Sumukha S S	PES1UG21CS833

SECTION 1 : PROBLEM STATEMENT

The goal is to create a system for reliably recommending movies to users based on their viewing interests, viewing history, and other pertinent information like movie ratings and reviews. The system should be able to offer tailored movie suggestions that take into consideration each user's distinct preferences and interests as well as their demographics and behavioral tendencies. By providing pertinent and interesting movie suggestions that boost user engagement and happiness with the site, the aim is to enhance the user experience

SECTION 2 : INTRODUCTION

A personalized movie recommendation system proposes films to viewers based on their prior viewing habits, movie reviews, and other pertinent information. In order to generate personalized recommendations that are catered to each user's particular interests and preferences, the system employs algorithms to analyze a user's watching history, demographic traits, and behavioral patterns.

Systems for making movie recommendations are crucial for streaming video services like Netflix, Hulu, and Amazon Prime Video. It can be difficult for people to find films or TV episodes they want to watch on these platforms because they have such a large selection. A recommendation system makes it easier for users to access pertinent material fast, enhancing their platform engagement.

There are several applications for movie recommendation algorithms in the real world. For instance, in the entertainment sector, production companies can recommend movie scripts that are expected to perform well at the box office using recommendation systems. Additionally, movie theatres can make movie suggestions to patrons based on their location, movie preferences, and other pertinent variables using recommendation algorithms. By recommending instructive films or documentaries to students based on their interests, movie recommendation systems in the educational sector might enhance the learning process.

Generally speaking, movie recommendation systems are crucial for giving users tailored recommendations, enhancing the user experience, and raising user engagement and happiness with the platform.

SECTION 3 : DATASET DESCRIPTION

The data for this assignment is gathered from grouplens. It is a IMDB movie dataset.

Link to dataset:

https://grouplens.org/datasets/movielens/100k/

```
0s   1   ! unzip ml-100k

Archive:  ml-100k.zip
   creating: ml-100k/
  inflating: ml-100k/allbut.pl
  inflating: ml-100k/mku.sh
  inflating: ml-100k/README
  inflating: ml-100k/u.data
  inflating: ml-100k/u.genre
  inflating: ml-100k/u.info
  inflating: ml-100k/u.item
  inflating: ml-100k/u.occupation
  inflating: ml-100k/u.user
  inflating: ml-100k/u1.base
  inflating: ml-100k/u1.test
  inflating: ml-100k/u2.base
  inflating: ml-100k/u2.test
  inflating: ml-100k/u3.base
  inflating: ml-100k/u3.test
  inflating: ml-100k/u4.base
  inflating: ml-100k/u4.test
  inflating: ml-100k/u5.base
  inflating: ml-100k/u5.test
  inflating: ml-100k/ua.base
  inflating: ml-100k/ua.test
  inflating: ml-100k/ub.base
  inflating: ml-100k/ub.test
```

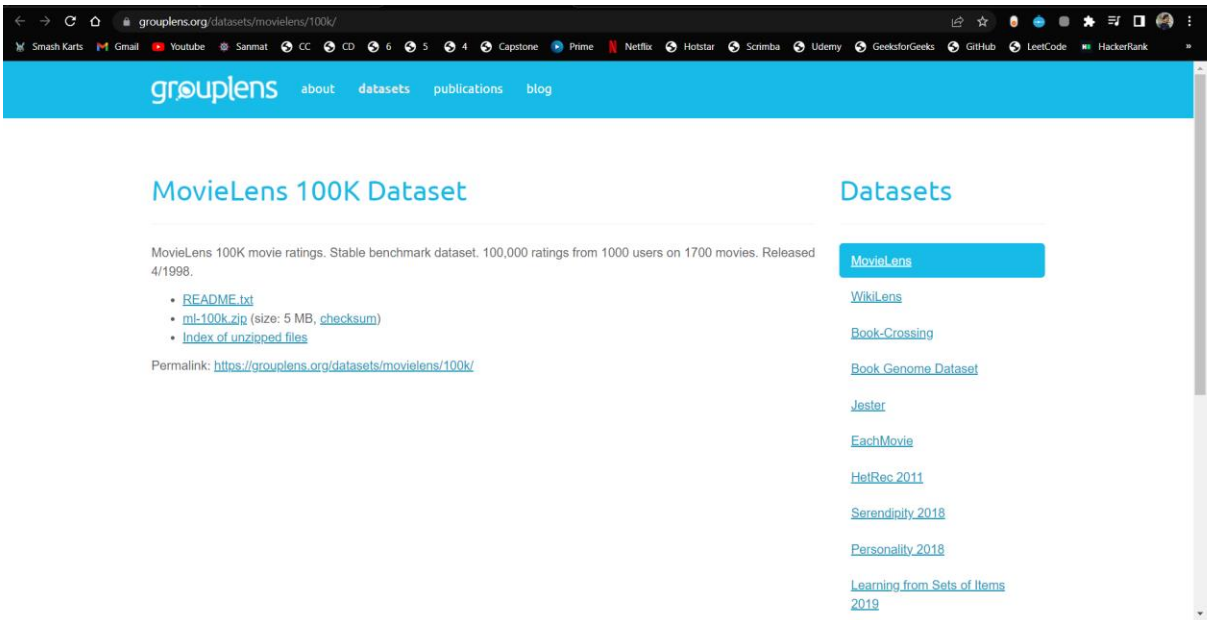
Summary of the Dataset:

A well-known benchmark dataset for recommender systems, the MovieLens 100K dataset is frequently used for developing and testing different machine learning algorithms.

The dataset consists of 100,000 ratings (from 1 to 5 stars) for 1,682 films from 943 individuals. Along with demographic data on users (such as age, gender, and occupation), it also contains details about the movie genres. The MovieLens users who volunteered to take part in the data collection process provided the evaluations between January 1995 and October 1998.

The dataset is made available in a number of different formats, including a CSV file that contains user ratings, a second file for user data, and a third file for movie data. The dataset also includes a README file that contains comprehensive information about the dataset, its application, and the data collection procedure.

Overall, the MovieLens 100K dataset is a useful tool for researchers and professionals working in the field of recommender systems. It can be applied to many different tasks, such as developing and assessing recommendation algorithms, analysing user behaviour, and examining the connection between demographics and movie preferences.



SECTION 4 : EDA

```
[4]
# Check the basic statistics of the dataset
print(data.describe())
```

```
            user_id        item_id         rating     timestamp
count  100000.00000  100000.000000  100000.000000  1.000000e+05
mean      462.48475     425.530130       3.529860  8.835289e+08
std       266.61442     330.798356       1.125674  5.343856e+06
min         1.00000       1.000000       1.000000  8.747247e+08
25%       254.00000     175.000000       3.000000  8.794487e+08
50%       447.00000     322.000000       4.000000  8.828269e+08
75%       682.00000     631.000000       4.000000  8.882600e+08
max       943.00000    1682.000000       5.000000  8.932866e+08
```

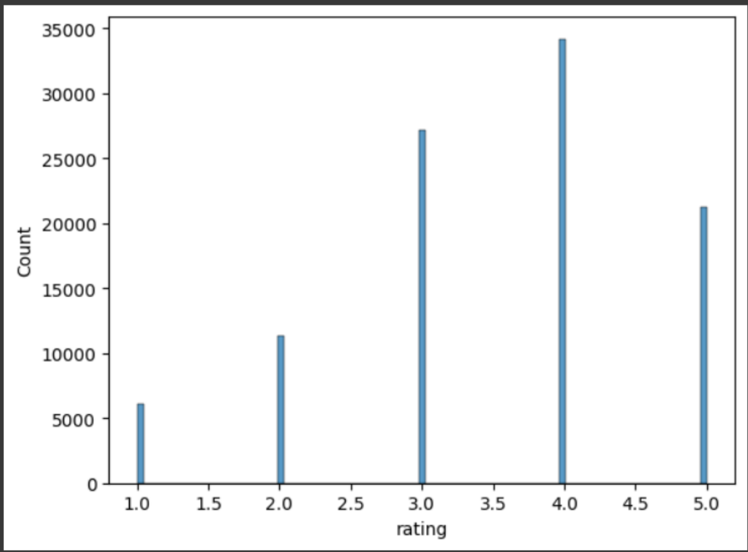
```
[5]
# Check the data types of the columns
print(data.dtypes)
```

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

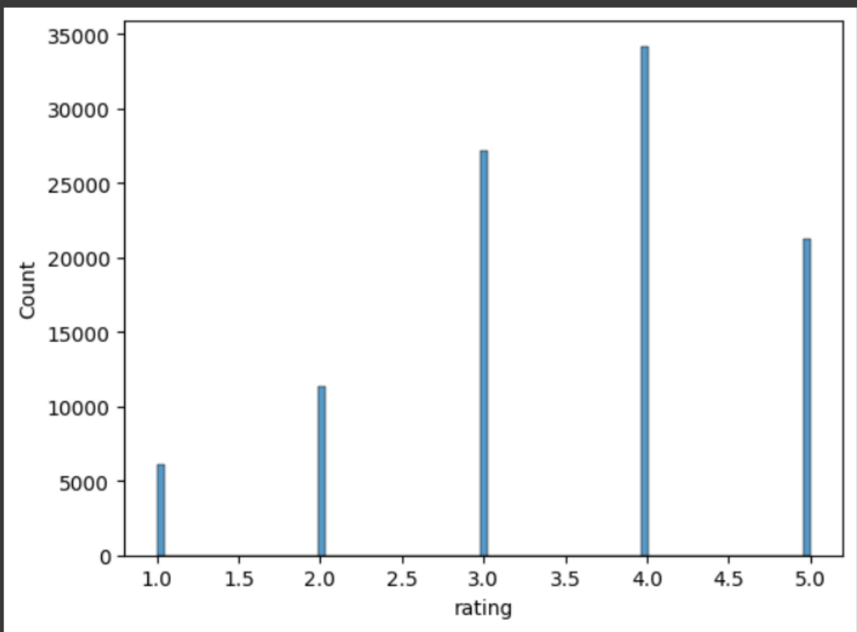
```
# Check the missing values in the dataset
print(data.isnull().sum())
```

```
user_id      0
item_id      0
rating       0
timestamp    0
dtype: int64
```

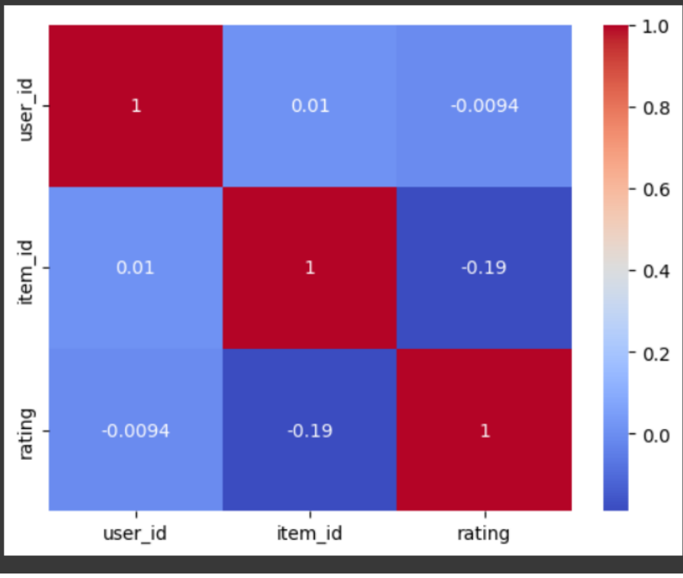
```python
# Check the distribution of the target variable
sns.histplot(data.rating)
plt.show()
```



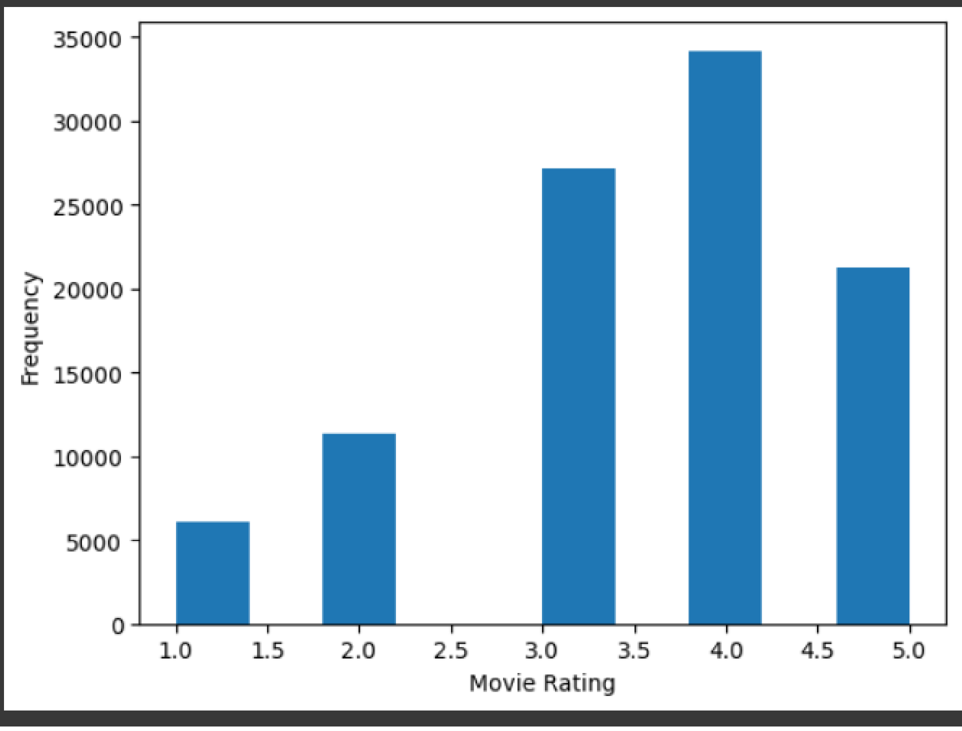

```python
# Check the distribution of the target variable
sns.histplot(data.rating)
plt.show()
```



```python
# calculate correlation matrix
corr_matrix = data.corr()

# create heatmap of correlation matrix
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.show()
```




```python
[12] import matplotlib.pyplot as plt

# create histogram of movie ratings
plt.hist(data['rating'], bins=10)
plt.xlabel('Movie Rating')
plt.ylabel('Frequency')
plt.show()
```



SECTION 5 : PRE PROCESSING OF DATA

```
user_data = pd.read_csv('ml-100k/u.user', sep='|', encoding='latin-1', header=None, names=['user id' ,'age' ,'gender', 'occupation', 'zip code'])
# movie_data = pd.read_csv('ml-100k/u.item')
data = pd.read_csv('ml-100k/u.data', sep='\t', names=['user_id', 'item_id', 'rating', 'timestamp'])
movie_data= pd.read_csv('ml-100k/u.genre', sep='|', names=['genres','sl_no'])
```

```
[77] data
```

	user_id	item_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
...	...	...	...	...
99995	880	476	3	880175444
99996	716	204	5	879795543
99997	276	1090	1	874795795
99998	13	225	2	882399156
99999	12	203	3	879959583

100000 rows × 4 columns

```
[69] user_data
```

	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
...	...	...	...	...	...
938	939	26	F	student	33319
939	940	32	M	administrator	02215
940	941	20	M	student	97229
941	942	48	F	librarian	78209
942	943	22	M	student	77841

943 rows × 5 columns

```
movie_data
```

	genres	sl_no
0	unknown	0
1	Action	1
2	Adventure	2
3	Animation	3
4	Children's	4
5	Comedy	5
6	Crime	6
7	Documentary	7
8	Drama	8
9	Fantasy	9
10	Film-Noir	10
11	Horror	11
12	Musical	12
13	Mystery	13
14	Romance	14
15	Sci-Fi	15
16	Thriller	16
17	War	17
18	Western	18

```
[35] movies
```

	item_id	title	unknown	Action	Adventure	Animation	Childrens	Comedy	Crime	Documentary	...	Fantasy	Film-Noir	Horror	Musical	Mystery	Romance	Sci-Fi	Thriller	War	Western
0	1	Toy Story (1995)	0	0	0	1	1	1	0	0	...	0	0	0	0	0	0	0	0	0	0
1	2	GoldenEye (1995)	0	1	1	0	0	0	0	0	...	0	0	0	0	0	0	0	1	0	0
2	3	Four Rooms (1995)	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	1	0	0
3	4	Get Shorty (1995)	0	1	0	0	0	1	0	0	...	0	0	0	0	0	0	0	0	0	0
4	5	Copycat (1995)	0	0	0	0	0	0	1	0	...	0	0	0	0	0	0	0	1	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1677	1678	Mat' i syn (1997)	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1678	1679	B. Monkey (1998)	0	0	0	0	0	0	0	0	...	0	0	0	0	0	1	0	1	0	0
1679	1680	Sliding Doors (1998)	0	0	0	0	0	0	0	0	...	0	0	0	0	0	1	0	0	0	0
1680	1681	You So Crazy (1994)	0	0	0	0	0	1	0	0	...	0	0	0	0	0	0	0	0	0	0
1681	1682	Scream of Stone (Schrei aus Stein) (1991)	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

1682 rows × 21 columns

SECTION 6 : USING COLLABARATIVE FILTERING

Collaborative filtering analysis :

A well-liked recommendation system called collaborative filtering uses user behaviour analysis to produce tailored recommendations. EDA approaches can be used to spot patterns in user behaviour, such as the films that people tend to watch together most often or the genres that appeal to different age groups.

COLLABRATIVE FILTERING METHOD

```python
# calculate user-item matrix
user_item = data.pivot_table(index='user_id', columns='item_id', values='rating')

# calculate item-item correlation matrix
item_corr = user_item.corr()

# get top movie recommendations for a specific user
user_id = 1
user_ratings = user_item.loc[user_id].dropna()
similar_items = pd.DataFrame()
for movie_id, rating in user_ratings.iteritems():
    similar_movies = item_corr[movie_id].dropna()
    similar_movies = similar_movies.map(lambda x: x * rating)
    similar_items = similar_items.append(similar_movies)
recommendations = similar_items.groupby(similar_items.index).sum()
```

```
<ipython-input-17-de68de1fcaa7>:14: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  similar_items = similar_items.append(similar_movies)
<ipython-input-17-de68de1fcaa7>:14: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  similar_items = similar_items.append(similar_movies)
<ipython-input-17-de68de1fcaa7>:14: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  similar_items = similar_items.append(similar_movies)
<ipython-input-17-de68de1fcaa7>:14: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  similar_items = similar_items.append(similar_movies)
<ipython-input-17-de68de1fcaa7>:14: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  similar_items = similar_items.append(similar_movies)
<ipython-input-17-de68de1fcaa7>:14: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  similar_items = similar_items.append(similar_movies)
<ipython-input-17-de68de1fcaa7>:14: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  similar_items = similar_items.append(similar_movies)
<ipython-input-17-de68de1fcaa7>:14: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  similar_items = similar_items.append(similar_movies)
<ipython-input-17-de68de1fcaa7>:14: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
```

OUTPUT :

```python
recommendations
```

item_id	1	2	3	4	5	6	7	8	9	10	...	1390	1430	1433	1612	1662	1294	1463	1602	1617	1656
1	5.000000	1.108921	0.878971	0.515676	1.932376	2.647005	0.796240	1.236836	0.450880	0.937508	...	0.0	0.000000	0.0	0.000000	0.0	0.000000e+00	0.000000	0.000000	0.0	0.0
2	0.665352	3.000000	0.691607	0.733667	0.652678	-0.474342	0.526334	1.021576	-0.682797	0.597248	...	0.0	0.000000	0.0	0.000000	0.0	0.000000e+00	0.000000	0.000000	0.0	0.0
3	0.703177	0.922142	4.000000	-0.807875	0.738449	3.224903	0.286034	-0.474350	0.066967	0.286251	...	0.0	0.000000	0.0	0.000000	0.0	0.000000e+00	0.000000	0.000000	0.0	0.0
4	0.309406	0.733667	-0.605906	3.000000	-0.712051	0.199876	0.458200	0.843810	0.625377	0.695792	...	0.0	0.000000	0.0	0.000000	0.0	0.000000e+00	0.000000	0.000000	0.0	0.0
5	1.159425	0.652678	0.553837	-0.712051	3.000000	3.000000	0.540634	0.615343	0.195508	-2.530984	...	0.0	0.000000	0.0	0.000000	0.0	0.000000e+00	0.000000	0.000000	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
268	0.939579	1.651774	1.023417	0.354126	-0.188942	3.411211	0.898446	-0.240446	1.085550	0.518785	...	5.0	0.000000	0.0	0.944911	0.0	2.428309e+00	-4.853627	0.000000	5.0	0.0
269	0.425014	1.136571	0.728447	0.159719	1.335971	3.071249	0.930584	-0.006626	0.809844	0.834010	...	5.0	-4.330127	-5.0	5.000000	0.0	-3.692745e+00	5.000000	4.724556	0.0	-5.0
270	1.570184	1.352171	1.030530	-0.549004	1.024631	-1.401121	0.644890	0.496382	-0.876519	0.674882	...	0.0	0.000000	0.0	5.000000	0.0	-3.585686e+00	0.000000	5.000000	0.0	5.0
271	0.222325	0.766828	0.818788	0.184635	0.251098	-0.216930	0.502504	-0.172970	0.262695	0.334077	...	0.0	0.000000	0.0	0.000000	0.0	-4.662524e-01	0.000000	2.000000	0.0	0.0
272	0.979869	1.527555	1.121110	0.129711	1.578236	-0.743151	1.206265	1.128532	0.891530	0.235918	...	0.0	-3.000000	0.0	3.000000	0.0	-8.599751e-16	3.000000	0.000000	0.0	0.0

272 rows × 1516 columns

```
[21] from sklearn.model_selection import train_test_split

     # Split the dataset into training and testing sets
     train_data, test_data = train_test_split(data, test_size=0.2, random_state=42)
```

```
[22] # Create a user-item matrix
     train_matrix = np.zeros((n_users, n_items))

     for row in train_data.itertuples():
         train_matrix[row[1]-1, row[2]-1] = row[3]
```

```
[23] from surprise import SVD
     from surprise import Dataset
     from surprise import Reader
     from surprise.model_selection import cross_validate

     # Load the dataset into the Surprise framework
     reader = Reader(rating_scale=(1, 5))
     data = Dataset.load_from_df(data[['user_id', 'item_id', 'rating']], reader)

     # Use the SVD algorithm
     algo = SVD()

     # Evaluate the algorithm using cross-validation
     cross_validate(algo, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9373  0.9343  0.9372  0.9420  0.9348  0.9371  0.0028
MAE (testset)     0.7392  0.7384  0.7385  0.7418  0.7372  0.7390  0.0015
Fit time          1.81    1.48    1.51    1.50    1.37    1.53    0.15
Test time         0.14    0.14    0.41    0.17    0.16    0.21    0.10
{'test_rmse': array([0.93729024, 0.93425409, 0.93719366, 0.94204859, 0.93479847]),
 'test_mae': array([0.73915899, 0.73835532, 0.73849565, 0.74177503, 0.73724993]),
```

COSINE SIMILARITY :

COSINE SIMILARITY

```
[26] from sklearn.metrics.pairwise import cosine_similarity

     # Compute the pairwise cosine similarity between movies
     movie_similarity_matrix = cosine_similarity(movie_genre_matrix)
```

```
[31] movie_similarity_matrix
```

```
array([[1.        , 0.        , 0.        , ..., 0.        , 0.70710678,
        0.        ],
       [0.        , 1.        , 1.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 1.        , 1.        , ..., 0.        , 0.        ,
        0.        ],
       ...,
       [0.        , 0.        , 0.        , ..., 1.        , 0.        ,
        0.70710678],
       [0.70710678, 0.        , 0.        , ..., 0.        , 1.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.70710678, 0.        ,
        1.        ]])
```

SECTION 7 : CONTENT BASED

```
# Get the index of the movie
movie_index = movies[movies['title'] == 'Toy Story (1995)'].index.values[0]

# Get the pairwise similarity scores for the given movie
movie_similarity_scores = list(enumerate(movie_similarity_matrix[movie_index]))

# Sort the movies based on the similarity scores
sorted_movie_similarity_scores = sorted(movie_similarity_scores, key=lambda x: x[1], reverse=True)

# Get the top 10 similar movies
top_10_similar_movies = sorted_movie_similarity_scores[1:11]

# Print the top 10 similar movies
for i, score in top_10_similar_movies:
    print(movies.iloc[i]['title'])
```

```
Santa Clause, The (1994)
Home Alone (1990)
D3: The Mighty Ducks (1996)
Love Bug, The (1969)
Willy Wonka and the Chocolate Factory (1971)
101 Dalmatians (1996)
Jungle2Jungle (1997)
George of the Jungle (1997)
Air Bud (1997)
Heavyweights (1994)
```

ANALYSIS OF MODEL :

```
from surprise import Dataset, Reader
from surprise import KNNBasic
from surprise import accuracy
from surprise.model_selection import train_test_split

# Load the dataset
reader = Reader(line_format='user item rating timestamp', sep='\t')
data = Dataset.load_from_file('ml-100k/u.data', reader=reader)

# Split the dataset into training and testing sets
trainset, testset = train_test_split(data, test_size=0.25)

# Train the model
model = KNNBasic()
model.fit(trainset)

# Make predictions on the testing set
predictions = model.test(testset)

# Compute the RMSE and MAE
rmse = accuracy.rmse(predictions)
mae = accuracy.mae(predictions)

print(f'RMSE: {rmse}, MAE: {mae}')
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE: 0.9785
MAE:  0.7714
RMSE: 0.9784938650064566, MAE: 0.7714496915381406
```

PREDICTIONS :

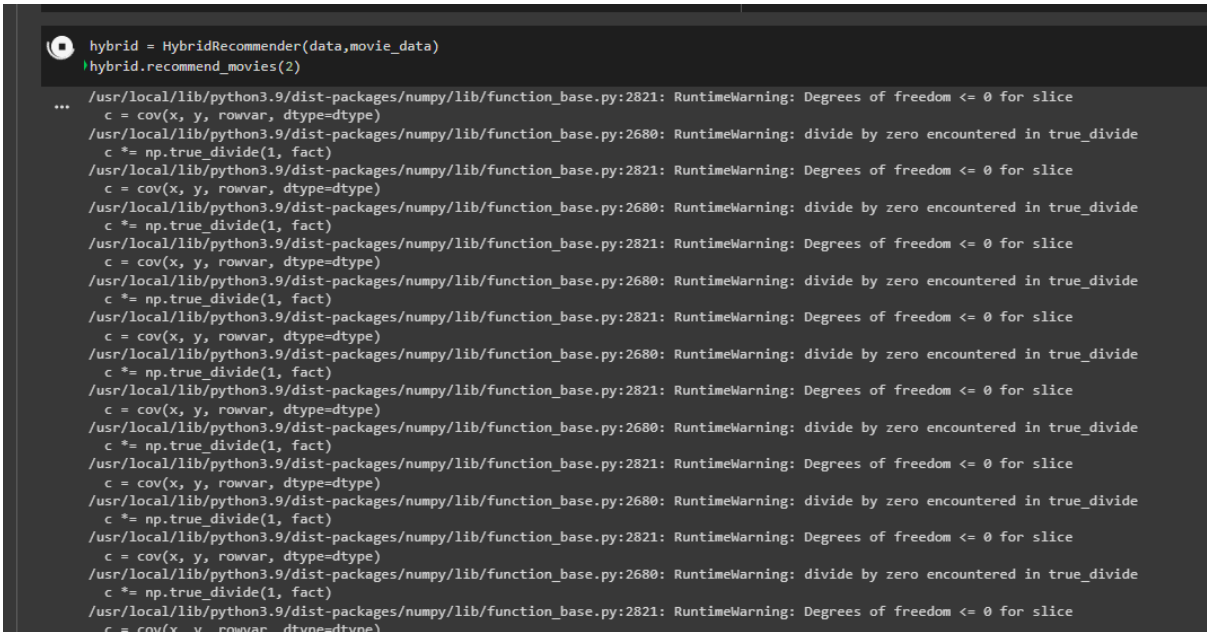
```
predictions
```

```
Prediction(uid='314', iid='196', r_ui=3.0, est=4.305108520325164, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='584', iid='114', r_ui=4.0, est=4.408402162319082, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='97', iid='174', r_ui=4.0, est=4.645338296898262, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='268', iid='117', r_ui=4.0, est=3.4841790677600204, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='920', iid='332', r_ui=3.0, est=3.068089872328938, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='178', iid='597', r_ui=4.0, est=3.101477133858893, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='5', iid='230', r_ui=3.0, est=3.3940271411158136, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='409', iid='300', r_ui=3.0, est=3.7472995565132976, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='496', iid='526', r_ui=3.0, est=3.549871011417222, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='421', iid='427', r_ui=4.0, est=4.319058793379312, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='923', iid='411', r_ui=4.0, est=3.283388466300335, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='311', iid='747', r_ui=3.0, est=3.3986471046014537, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='639', iid='14', r_ui=5.0, est=4.090612504321359, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='804', iid='415', r_ui=3.0, est=2.879291337848259, details={'actual_k': 21, 'was_impossible': False}),
Prediction(uid='747', iid='8', r_ui=5.0, est=4.188712399033378, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='385', iid='616', r_ui=4.0, est=3.4549134584237504, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='854', iid='220', r_ui=4.0, est=3.1330993262808136, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='9', iid='298', r_ui=5.0, est=3.791934240945285, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='748', iid='199', r_ui=4.0, est=4.023356628947637, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='854', iid='1013', r_ui=1.0, est=2.249350273285114, details={'actual_k': 32, 'was_impossible': False}),
Prediction(uid='835', iid='486', r_ui=4.0, est=3.960301130883048, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='64', iid='182', r_ui=4.0, est=3.9566211174782535, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='838', iid='153', r_ui=4.0, est=3.946796049534652, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='768', iid='25', r_ui=4.0, est=3.4090736980053498, details={'actual_k': 40, 'was_impossible': False}),
```

HYBRID RECOMMENDOR :

```
[98] from sklearn.feature_extraction.text import CountVectorizer
```

```python
class HybridRecommender:
    def __init__(self, user_data, movie_data):
        self.user_data = user_data
        self.movie_data = movie_data

        # calculate user-item matrix for collaborative filtering
        self.user_item = user_data.pivot_table(index='user_id', columns='item_id', values='rating')

        # create bag-of-words matrix for movie genres for content-based filtering
        self.vectorizer = CountVectorizer(tokenizer=lambda x: x, lowercase=False)
        self.genre_matrix = self.vectorizer.fit_transform(movie_data['genres'])

        # calculate pairwise cosine similarity between movies for content-based filtering
        self.movie_similarities = cosine_similarity(self.genre_matrix)

    def recommend_movies(self, user_id):
        # perform collaborative filtering
        user_ratings = self.user_item.loc[user_id].dropna()
        similar_users = self.user_item.drop(user_id).corrwith(user_ratings)
        similar_users = similar_users.dropna()
        user_recommendations = self.user_item.loc[similar_users.index].mean().drop(user_ratings.index)
        user_recommendations = user_recommendations.sort_values(ascending=False)[:10]

        # perform content-based filtering
        user_ratings = self.user_data[self.user_data['user_id'] == user_id]
        user_genres = user_ratings['genres'].apply(lambda x: '|'.join(x))
        user_genre_matrix = self.vectorizer.transform(user_genres)
        user_similarities = cosine_similarity(user_genre_matrix, self.genre_matrix).flatten()
        user_similarities = pd.Series(user_similarities, index=self.movie_data.index)
        content_recommendations = self.movie_data.loc[user_similarities.argsort()[::-1][:10]]

        # combine results from collaborative filtering and content-based filtering
        recommended_movies = pd.concat([user_recommendations, content_recommendations])
        recommended_movies = recommended_movies.drop_duplicates()
        recommended_movies = recommended_movies.sort_values('rating', ascending=False)

        # return top 10 recommended movies
        return recommended_movies.head(10)
```



```
hybrid = HybridRecommender(data,movie_data)
hybrid.recommend_movies(2)
```

```
/usr/local/lib/python3.9/dist-packages/numpy/lib/function_base.py:2821: RuntimeWarning: Degrees of freedom <= 0 for slice
  c = cov(x, y, rowvar, dtype=dtype)
/usr/local/lib/python3.9/dist-packages/numpy/lib/function_base.py:2680: RuntimeWarning: divide by zero encountered in true_divide
  c *= np.true_divide(1, fact)
/usr/local/lib/python3.9/dist-packages/numpy/lib/function_base.py:2821: RuntimeWarning: Degrees of freedom <= 0 for slice
  c = cov(x, y, rowvar, dtype=dtype)
/usr/local/lib/python3.9/dist-packages/numpy/lib/function_base.py:2680: RuntimeWarning: divide by zero encountered in true_divide
  c *= np.true_divide(1, fact)
/usr/local/lib/python3.9/dist-packages/numpy/lib/function_base.py:2821: RuntimeWarning: Degrees of freedom <= 0 for slice
  c = cov(x, y, rowvar, dtype=dtype)
/usr/local/lib/python3.9/dist-packages/numpy/lib/function_base.py:2680: RuntimeWarning: divide by zero encountered in true_divide
  c *= np.true_divide(1, fact)
/usr/local/lib/python3.9/dist-packages/numpy/lib/function_base.py:2821: RuntimeWarning: Degrees of freedom <= 0 for slice
  c = cov(x, y, rowvar, dtype=dtype)
/usr/local/lib/python3.9/dist-packages/numpy/lib/function_base.py:2680: RuntimeWarning: divide by zero encountered in true_divide
  c *= np.true_divide(1, fact)
/usr/local/lib/python3.9/dist-packages/numpy/lib/function_base.py:2821: RuntimeWarning: Degrees of freedom <= 0 for slice
  c = cov(x, y, rowvar, dtype=dtype)
/usr/local/lib/python3.9/dist-packages/numpy/lib/function_base.py:2680: RuntimeWarning: divide by zero encountered in true_divide
  c *= np.true_divide(1, fact)
/usr/local/lib/python3.9/dist-packages/numpy/lib/function_base.py:2821: RuntimeWarning: Degrees of freedom <= 0 for slice
  c = cov(x, y, rowvar, dtype=dtype)
/usr/local/lib/python3.9/dist-packages/numpy/lib/function_base.py:2680: RuntimeWarning: divide by zero encountered in true_divide
  c *= np.true_divide(1, fact)
/usr/local/lib/python3.9/dist-packages/numpy/lib/function_base.py:2821: RuntimeWarning: Degrees of freedom <= 0 for slice
  c = cov(x, y, rowvar, dtype=dtype)
/usr/local/lib/python3.9/dist-packages/numpy/lib/function_base.py:2680: RuntimeWarning: divide by zero encountered in true_divide
  c *= np.true_divide(1, fact)
/usr/local/lib/python3.9/dist-packages/numpy/lib/function_base.py:2821: RuntimeWarning: Degrees of freedom <= 0 for slice
```

CONCLUSION:

In conclusion, a movie recommendation system is a crucial tool for giving customers tailored movie suggestions based on their viewing interests and history. Collaborative filtering, content-based filtering, and hybrid recommender systems—which integrate both methods—are a few examples of different recommender system kinds. Each of these strategies has benefits and drawbacks.

A common strategy for recommending films is collaborative filtering, which makes use of user behaviour data. It operates by identifying user similarities and recommending films that those users have seen and highly rated. On the other side, content-based filtering suggests films based on their qualities and characteristics, such as their genre, actors, and directors.

For more precise and varied recommendations, hybrid recommender systems integrate collaborative and content-based filtering. They can get beyond each method's drawbacks and offer a better overall suggestion experience.

To create a movie recommendation system, a variety of tools and packages are available, including Surprise, LightFM, and TensorFlow. These tools offer a quick and scalable solution to install various recommender system types and assess their effectiveness.

Movie streaming services like Netflix, Amazon Prime Video, and Hulu frequently use movie recommender systems to offer their subscribers individualised movie recommendations. They can boost user experience overall, raise revenue, and improve user engagement and retention.

The movie recommendation system is an effective tool that may offer users personalised and pertinent movie choices. We can create accurate and efficient recommendation systems that give viewers a better movie-watching experience by utilising the advantages of various methodologies.