NAME : ANBUSELVAN MAHALINGAM

SUBJECT : DSC 630 PREDICTIVE ANALYSIS - WEEK 10 ASSIGNMENT

In [46]:
```
# To supress the warning.
import warnings
warnings.filterwarnings('ignore')
```

In [47]:
```
# To Load the File into dataframe
import pandas as pd
def readfile(fileName):
    try:
        df = pd.read_csv(fileName)
        return df
    except:
        print(r'Unable to read the file. Validate the file d try again.!')
```

In [48]:
```
#To read the Movies dataset
movies_df = readfile("ml-latest-small/movies.csv")

#To read the Links dataset
links_df = readfile("ml-latest-small/links.csv")

#To read the Ratings dataset
ratings_df = readfile("ml-latest-small/ratings.csv")

#To read the Tags dataset
tags_df = readfile("ml-latest-small/tags.csv")
```

In [49]:
```
# EDA - To understand more about the dataset
print(movies_df.info())
print(movies_df.describe())
print(movies_df.head(n=2))
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9742 entries, 0 to 9741
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   movieId  9742 non-null   int64 
 1   title    9742 non-null   object
 2   genres   9742 non-null   object
dtypes: int64(1), object(2)
memory usage: 228.5+ KB
None
             movieId
count    9742.000000
mean    42200.353623
std     52160.494854
min         1.000000
25%      3248.250000
50%      7300.000000
75%     76232.000000
max    193609.000000
   movieId             title                                       genres
0        1  Toy Story (1995)  Adventure|Animation|Children|Comedy|Fantasy
1        2    Jumanji (1995)                   Adventure|Children|Fantasy
```

In [50]:

```
# EDA - To understand more about the dataset
print(links_df.info())
print(links_df.describe())
print(links_df.head(n=2))
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9742 entries, 0 to 9741
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   movieId  9742 non-null   int64
 1   imdbId   9742 non-null   int64
 2   tmdbId   9734 non-null   float64
dtypes: float64(1), int64(2)
memory usage: 228.5 KB
None
             movieId        imdbId         tmdbId
count    9742.000000  9.742000e+03    9734.000000
mean    42200.353623  6.771839e+05   55162.123793
std     52160.494854  1.107228e+06   93653.481487
min         1.000000  4.170000e+02       2.000000
25%      3248.250000  9.518075e+04    9665.500000
50%      7300.000000  1.672605e+05   16529.000000
75%     76232.000000  8.055685e+05   44205.750000
max    193609.000000  8.391976e+06  525662.000000
   movieId  imdbId  tmdbId
0        1  114709   862.0
1        2  113497  8844.0
```

In [51]:

```
# EDA - To understand more about the dataset
print(ratings_df.info())
print(ratings_df.describe())
print(ratings_df.head(n=2))
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100836 entries, 0 to 100835
Data columns (total 4 columns):
 #   Column     Non-Null Count   Dtype
---  ------     --------------   -----
 0   userId     100836 non-null  int64
 1   movieId    100836 non-null  int64
 2   rating     100836 non-null  float64
 3   timestamp  100836 non-null  int64
dtypes: float64(1), int64(3)
memory usage: 3.1 MB
None
              userId        movieId         rating     timestamp
count  100836.000000  100836.000000  100836.000000  1.008360e+05
mean      326.127564   19435.295718       3.501557  1.205946e+09
std       182.618491   35530.987199       1.042529  2.162610e+08
min         1.000000       1.000000       0.500000  8.281246e+08
25%       177.000000    1199.000000       3.000000  1.019124e+09
50%       325.000000    2991.000000       3.500000  1.186087e+09
75%       477.000000    8122.000000       4.000000  1.435994e+09
max       610.000000  193609.000000       5.000000  1.537799e+09
   userId  movieId  rating  timestamp
0       1        1     4.0  964982703
1       1        3     4.0  964981247
```

In [52]:

```
# EDA - To understand more about the dataset
print(tags_df.info())
print(tags_df.describe())
print(tags_df.head(n=2))
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3683 entries, 0 to 3682
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   userId     3683 non-null   int64 
 1   movieId    3683 non-null   int64 
 2   tag        3683 non-null   object
 3   timestamp  3683 non-null   int64 
dtypes: int64(3), object(1)
memory usage: 115.2+ KB
None
            userId        movieId     timestamp
count  3683.000000    3683.000000  3.683000e+03
mean    431.149335   27252.013576  1.320032e+09
std     158.472553   43490.558803  1.721025e+08
min       2.000000       1.000000  1.137179e+09
25%     424.000000    1262.500000  1.137521e+09
50%     474.000000    4454.000000  1.269833e+09
75%     477.000000   39263.000000  1.498457e+09
max     610.000000  193565.000000  1.537099e+09
   userId  movieId              tag   timestamp
0       2    60756            funny  1445714994
1       2    60756  Highly quotable  1445714996
```

In [53]:

```
# Merge the 4 dataframes into one dataframe

# To merge movies_df with links_df on movieId
movies_links_df = pd.merge(movies_df, links_df,
                           on='movieId')

# to merge the resulting DataFrame with ratings_df on movieId
movies_links_ratings_df = pd.merge(movies_links_df,
                                   ratings_df, on='movieId')

# to merge the resulting DataFrame with tags_df on movieId and userId
full_movie_rating_df = pd.merge(movies_links_ratings_df, tags_df,
                   on=['movieId', 'userId'], how='left')
```

In [54]:

```
full_movie_rating_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 102677 entries, 0 to 102676
Data columns (total 10 columns):
 #   Column       Non-Null Count   Dtype  
---  ------       --------------   -----  
 0   movieId      102677 non-null  int64  
 1   title        102677 non-null  object 
 2   genres       102677 non-null  object 
 3   imdbId       102677 non-null  int64  
 4   tmdbId       102664 non-null  float64
 5   userId       102677 non-null  int64  
 6   rating       102677 non-null  float64
 7   timestamp_x  102677 non-null  int64  
 8   tag          3476 non-null    object 
 9   timestamp_y  3476 non-null    float64
dtypes: float64(3), int64(4), object(3)
memory usage: 7.8+ MB
```

In [55]:
```
# To drop unnecessary columns
movie_data_cleaned = full_movie_rating_df.drop(columns=['timestamp_x', 'timestamp_y', 'imdbId', 'tmdbId', 'tag'])
```

The columns 'timestamp_x', 'timestamp_y', 'imdbId', 'tmdbId', 'tag' are not needed for our analytical purpose and not required for the recommendation system design.

In [56]:
```
# To calculate average ratings and total count of ratings for each movie
average_ratings = pd.DataFrame(data_cleaned.groupby('title')['rating'].mean())
average_ratings['Total Ratings'] = data_cleaned.groupby('title')['rating'].count()
```

In [86]:
```
# display the first few rows of the merged DataFrame
movie_data_cleaned.head(n=3)
```

Out[86]:

	movieId	title	genres	userId	rating
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	1	4.0
1	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	5	4.0
2	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	7	4.5

In [87]:
```python
# To create User-Item Matrix
movie_user = data_cleaned.pivot_table(index='userId',columns='title',values='rating')
```

In [88]:
```python
# To find the correlation value for the movie with all other movies
correlations = movie_user.corrwith(movie_user['Toy Story (1995)'])
correlations.head(n=10)
```

Out[88]:
```
title
'71 (2014)                                    NaN
'Hellboy': The Seeds of Creation (2004)       NaN
'Round Midnight (1986)                        NaN
'Salem's Lot (2004)                           NaN
'Til There Was You (1997)                     NaN
'Tis the Season for Love (2015)               NaN
'burbs, The (1989)                       0.240563
'night Mother (1986)                          NaN
(500) Days of Summer (2009)              0.353833
*batteries not included (1987)          -0.427425
dtype: float64
```

In [95]:
```python
# Reusable - user defined function

def get_movie_recommendations(movie_title, user_item_matrix, average_ratings, min_ratings=100):

    # To calculates correlations between the target movie and all other movies
    correlations = user_item_matrix.corrwith(user_item_matrix[movie_title])

    # To create new Data frame with recommendation based on correlation
    recommendation = pd.DataFrame(correlations, columns=['Correlation']).dropna()

    # Joins the dataFrame with total number of ratings for each movie
    recommendation = recommendation.join(average_ratings['Total Ratings'])

    # To Filter movies with good ratings
    recommendation = recommendation[recommendation['Total Ratings'] > min_ratings]

    # To sort movies by their correlation and return to 10 movies.
    return recommendation.sort_values('Correlation', ascending=False).head(10)
```

In [100…
```python
# Reusable - user defined function, To get User input
def recommend_movies(movie_title):
```

```
    try:
        recommendations = get_movie_recommendations(movie_title,
                                                    user_item_matrix,
                                                    average_ratings)
        print(f"Top 10 recommendations for '{movie_title}':")
        for idx, row in recommendations.iterrows():
             print(f"Movie: {idx}, Correlation: {row['Correlation']:.3f},
             Total Ratings: {row['Total Ratings']}")
    except KeyError:
        print(f"Sorry, the movie '{movie_title}' is not found in the dataset.")
```

In [101…

```
# To the Recommended movies - Validation
user_movie = "Toy Story (1995)"
recommend_movies(user_movie)
```

```
Top 10 recommendations for 'Toy Story (1995)':
Movie: Toy Story (1995), Correlation: 1.000, Total Ratings: 215.0
Movie: Toy Story 2 (1999), Correlation: 0.699, Total Ratings: 103.0
Movie: Incredibles, The (2004), Correlation: 0.643, Total Ratings: 127.0
Movie: Finding Nemo (2003), Correlation: 0.619, Total Ratings: 142.0
Movie: Aladdin (1992), Correlation: 0.612, Total Ratings: 183.0
Movie: Monsters, Inc. (2001), Correlation: 0.490, Total Ratings: 132.0
Movie: Mrs. Doubtfire (1993), Correlation: 0.446, Total Ratings: 146.0
Movie: Amelie (Fabuleux destin d'Amélie Poulain, Le) (2001), Correlation: 0.438, Total Ratings: 120.0
Movie: American Pie (1999), Correlation: 0.420, Total Ratings: 103.0
Movie: Die Hard: With a Vengeance (1995), Correlation: 0.411, Total Ratings: 144.0
```

In [98]:

```
# To the Recommended movies, validation
user_movie = "Star wars"
recommend_movies(user_movie)
```

```
Sorry, the movie 'Star wars' is not found in the dataset.
```

Summary of the Approach

```
Load datasets - Read each datasets and merge them into a single dataFrame, remove unwanted
elements from the dataset
Feature Engineering - Compute average ratings and the total number of ratings for each movie.
Collaborative Filtering User-Item Matrix - Pivot the data to form a matrix of user ratings for
each movie.
Correlations calculation: Calculate similarities between the target movie and all other movies
using correlation.
```

Generate Recommendations: Filter and sort movies based on correlation and display the top 10 recommendations.

Process: Here we have created a collaborative filtering-based recommender system, focusing on calculating movie similarities using correlation and recommending movies based on the corelations. The user-item matrix and the calculation of correlations helps us to generate movie recommendations.

Reference : Recommender System Using Python & MovieLens - https://analyticsindiamag.com/how-to-build-your-first-recommender-system-using-python-movielens-dataset/