1. Import Libraries

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
from collections import defaultdict
from scipy import sparse
from scipy.stats import pearsonr
from sklearn.metrics.pairwise import cosine_similarity
from scipy.sparse import csr_matrix
from sklearn.neighbors import NearestNeighbors
import warnings
from cmfrec import CMF
from sklearn.metrics import mean_squared_error, mean_absolute_percentage_error
from surprise import Reader, SVD, KNNWithMeans, Dataset, accuracy
from surprise.model_selection import GridSearchCV, train_test_split, cross_validate
```

2. Set Options

In [2]:

```
#warnings.filterwarnings('ignore')
warnings.simplefilter('ignore')
pd.set_option("display.max_columns",None)
pd.options.display.float_format='{:.2f}'.format
sns.set_style('white')
```

3. Problem Statement

- Perform Analysis and provide Basic Recommendations based on followings:
  - Similar Movies
  - Similar watch by Users
  - Similar Genres
  - Highest rated movies
  - Movies That has received most Ratings

4. Read Data & Data Formatting

4.1 Movies

In [3]:

```
movies = pd.read_fwf('movies.dat',encoding='ISO-8859-1')
print(movies.shape)
movies.head()
```

```
(3883, 3)
```

Out[3]:

	Movie ID::Title::Genres	Unnamed: 1	Unnamed: 2
0	1::Toy Story (1995)::Animation|Children's|Comedy	NaN	NaN
1	2::Jumanji (1995)::Adventure|Children's|Fantasy	NaN	NaN
2	3::Grumpier Old Men (1995)::Comedy|Romance	NaN	NaN
3	4::Waiting to Exhale (1995)::Comedy|Drama	NaN	NaN
4	5::Father of the Bride Part II (1995)::Comedy	NaN	NaN

In [4]:

```
movies = movies["Movie ID::Title::Genres"].str.split("::",expand=True)
movies.rename(columns={0:"MovieID",1:"Title",2:'Genres'},inplace=True)
movies.head()
```

Out[4]:

	MovieID	Title	Genres
0	1	Toy Story (1995)	Animation|Children's|Comedy
1	2	Jumanji (1995)	Adventure|Children's|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama
4	5	Father of the Bride Part II (1995)	Comedy

In [5]:

```
duplicate_movies = movies[movies.duplicated()]
print("Duplicate Movies :")
duplicate_movies
```

```
Duplicate Movies :
```

Out[5]:

MovieID	Title	Genres

4.2 Users

In [6]:

```
users = pd.read_fwf('users.dat',encoding='ISO-8859-1')
print(users.shape)
users.head()
```

```
(6040, 1)
```

Out[6]:

	UserID::Gender::Age::Occupation::Zip-code
0	1::F::1::10::48067
1	2::M::56::16::70072
2	3::M::25::15::55117
3	4::M::45::7::02460
4	5::M::25::20::55455

In [7]:

```
users = users["UserID::Gender::Age::Occupation::Zip-code"].str.split("::",expand=True)
users.rename(columns={0:"UserID",1:"Gender",2:'Age',3:'Occupation',4:'Zip-code'},inplace
users.head()
```

Out[7]:

	UserID	Gender	Age	Occupation	Zip-code
0	1	F	1	10	48067
1	2	M	56	16	70072
2	3	M	25	15	55117
3	4	M	45	7	02460
4	5	M	25	20	55455

In [8]:

```
duplicate_users = users[users.duplicated()]
print("Duplicate Users :")
duplicate_users
```

```
Duplicate Users :
```

Out[8]:

UserID	Gender	Age	Occupation	Zip-code

4.3 Ratings

In [9]:

```
ratings = pd.read_fwf('ratings.dat',encoding='ISO-8859-1')
print(ratings.shape)
ratings.head()
```

```
(1000209, 1)
```

Out[9]:

	UserID::MovieID::Rating::Timestamp
0	1::1193::5::978300760
1	1::661::3::978302109
2	1::914::3::978301968
3	1::3408::4::978300275
4	1::2355::5::978824291

In [10]:

```
ratings= ratings['UserID::MovieID::Rating::Timestamp'].str.split("::",expand=True)
ratings.rename(columns={0:"UserID",1:"MovieID",2:'Rating',3:'Timestamp'},inplace=True)
ratings.head()
```

Out[10]:

	UserID	MovieID	Rating	Timestamp
0	1	1193	5	978300760
1	1	661	3	978302109
2	1	914	3	978301968
3	1	3408	4	978300275
4	1	2355	5	978824291

In [11]:

```
duplicate_ratings = ratings[ratings.duplicated()]
print("Duplicate ratings :")
duplicate_ratings
```

```
Duplicate ratings :
```

Out[11]:

UserID	MovieID	Rating	Timestamp

5. Data Pre-processing - Tranformation & Cleanup

5.1 Movies - Genres

5.1.1 Cleaning Genres

In [12]:

```
movies_genres = movies["Genres"].str.split("|",expand=True)
movies_genres.replace({
                        'Adv': "Adventure",
                        'Advent': "Adventure",
                        'Adventu': "Adventure",
                        'Adventur': "Adventure",
                        'Animati': "Animation",
                        'Acti': "Action",
                        'Chi': "Children",
                        'Chil': "Children",
                        'Childr': "Children",
                        'Childre': "Children",
                        'Children': "Children",
                        "Children's": "Children",
                        "Children'": "Children",
                        'Com': "Comedy",
                        'Come': "Comedy",
                        'Comed': "Comedy",
                        'D': "Documentary",
                        'Docu': "Documentary",
                        'Documen': "Documentary",
                        'Document': "Documentary",
                        'Documenta': "Documentary",
                        'Dr': "Drama",
                        'Dram': "Drama",
                        'F': "Fantasy",
                        'Fant': "Fantasy",
                        'Fantas': "Fantasy",
                        'Horr': "Horror",
                        'Horro': "Horror",
                        'Music': "Musical",
                        'R': "Romance",
                        'Ro': "Romance",
                        'Rom': "Romance",
                        'Roma': "Romance",
                        'Roman': "Romance",
                        'S': "Sci-Fiction",
                        'Sci': "Sci-Fiction",
                        'Sci-': "Sci-Fiction",
                        'Sci-F': "Sci-Fiction",
                        'Sci-Fi': "Sci-Fiction",
                        'Th': "Thriller",
                        'Thri': "Thriller",
                        'Thrille': "Thriller",
                        'Wa': "War",
                        'We': "Western",
                        'Wester': "Western",
                        'nan': "Unknown",
                        '': "Unknown",
                        'A': "Unknown"
                },inplace=True)
```

In [13]:
```
# Concatinating expanded columns post data cleaning
def concat_column_values(row):
    col0,col1,col2,col3,col4 = (row[0],row[1],row[2],row[3],row[4])
    str_list =[]

    if col0 != None:
        str_list.append(col0)
    if col1 != None:
        str_list.append(col1)
    if col2 != None :
        str_list.append(col2)
    if col3 != None:
        str_list.append(col3)
    if col4 != None:
        str_list.append(col4)

    return '|'.join(str_list)
```

In [14]:
```
# Cleaning typo Genres
movies['Genres']=movies_genres.apply(concat_column_values, axis=1)
# Tagging whitespace Genres to "Unknown"
movies.replace(r'^\s*$', "Unknown", regex=True,inplace=True)
movies.head()
```

Out[14]:

	MovieID	Title	Genres
0	1	Toy Story (1995)	Animation|Children|Comedy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama
4	5	Father of the Bride Part II (1995)	Comedy

5.1.2 Extracting Release Year from Title

In [15]:
```
movies['release_year'] = movies.Title.str.extract('\((\d{4})\)', expand=False)
movies[['Title','release_year']].head()
```

Out[15]:

	Title	release_year
0	Toy Story (1995)	1995
1	Jumanji (1995)	1995
2	Grumpier Old Men (1995)	1995
3	Waiting to Exhale (1995)	1995
4	Father of the Bride Part II (1995)	1995

Checking Genres post data cleanup

In [16]:
```
m = movies.copy()
m['Genres'] = m['Genres'].str.split('|')
m = m.explode('Genres')
m = m.pivot(index='MovieID', columns='Genres', values='Title')
m = ~m.isna()
m = m.astype(int)
```

In [17]:
```
m.columns
```

Out[17]:
```
Index(['Action', 'Adventure', 'Animation', 'Children', 'Comedy', 'Crime',
       'Documentary', 'Drama', 'Fantasy', 'Film-Noir', 'Horror', 'Musical',
       'Mystery', 'Romance', 'Sci-Fiction', 'Thriller', 'Unknown', 'War',
       'Western'],
      dtype='object', name='Genres')
```

5.2 Users - Age and Occupation

In [18]:
```
users.replace({'Age':{
                        '1': "Under 18",
                        '18': "18-24",
                        '25': "25-34",
                        '35': "35-44",
                        '45': "45-49",
                        '50': "50-55",
                        '56': "56+"
                    },
               'Occupation':{
                        '0': "other",
                        '1': "academic/educator",
                        '2': "artist",
                        '3': "clerical/admin",
                        '4': "college/grad student",
                        '5': "customer service",
                        '6': "doctor/health care",
                        '7': "executive/managerial",
                        '8': "farmer",
                        '9': "homemaker",
                        '10': "K-12 student",
                        '11': "lawyer",
                        '12': "programmer",
                        '13': "retired",
                        '14': "sales/marketing",
                        '15': "scientist",
                        '16': "self-employed",
                        '17': "technician/engineer",
                        '18': "tradesman/craftsman",
                        '19': "unemployed",
                        '20': "writer"
               }
},inplace=True)
```

In [19]:
```
print("Age categories -> ", users["Age"].unique())
print("Occupation categories ->",users["Occupation"].unique())
```

```
Age categories ->  ['Under 18' '56+' '25-34' '45-49' '50-55' '35-44' '18-24']
Occupation categories -> ['K-12 student' 'self-employed' 'scientist' 'executive/manageri
al'
 'writer' 'homemaker' 'academic/educator' 'programmer'
 'technician/engineer' 'other' 'clerical/admin' 'sales/marketing'
 'college/grad student' 'lawyer' 'farmer' 'unemployed' 'artist'
 'tradesman/craftsman' 'customer service' 'retired' 'doctor/health care']
```

In [20]:
```
users.head()
```

Out[20]:

	UserID	Gender	Age	Occupation	Zip-code
0	1	F	Under 18	K-12 student	48067
1	2	M	56+	self-employed	70072
2	3	M	25-34	scientist	55117
3	4	M	45-49	executive/managerial	02460
4	5	M	25-34	writer	55455

6. Feature Engineering

6.1 Ratings - Type Converstion (Timestamp to datetime and Rating to integer)

In [21]:

```
# unit='s' to convert it into epoch time
ratings['Timestamp'] = pd.to_datetime(ratings['Timestamp'],unit='s')
ratings["Rating"] = ratings["Rating"].astype(int)
ratings
```

Out[21]:

	UserID	MovieID	Rating	Timestamp
0	1	1193	5	2000-12-31 22:12:40
1	1	661	3	2000-12-31 22:35:09
2	1	914	3	2000-12-31 22:32:48
3	1	3408	4	2000-12-31 22:04:35
4	1	2355	5	2001-01-06 23:38:11
...	...	...	...	...
1000204	6040	1091	1	2000-04-26 02:35:41
1000205	6040	1094	5	2000-04-25 23:21:27
1000206	6040	562	5	2000-04-25 23:19:06
1000207	6040	1096	4	2000-04-26 02:20:48
1000208	6040	1097	4	2000-04-26 02:19:29

1000209 rows × 4 columns

6.2 Ratings - Deriving new features ‘Release_Year’,‘Release_Hour’,‘Release_Month’

In [22]:

```
ratings['Watch_Hour'] = ratings.Timestamp.dt.hour
ratings['Watch_Month'] = ratings.Timestamp.dt.month
ratings['Watch_Hour']=ratings['Watch_Hour'].astype(np.int64)
ratings['Watch_Month']=ratings['Watch_Month'].astype(np.int64)
```

In [23]:

```
ratings.head()
```

Out[23]:

	UserID	MovieID	Rating	Timestamp	Watch_Hour	Watch_Month
0	1	1193	5	2000-12-31 22:12:40	22	12
1	1	661	3	2000-12-31 22:35:09	22	12
2	1	914	3	2000-12-31 22:32:48	22	12
3	1	3408	4	2000-12-31 22:04:35	22	12
4	1	2355	5	2001-01-06 23:38:11	23	1

6.3 Ratings - Deriving new features ‘Rating_category’

In [24]:

```
rating_category_map={5:"Excellent",4:"Good",3:"Average",2:"Below Average",1:"Below Avera
ratings["Rating_Category"]= ratings["Rating"].map(rating_category_map)
```

In [25]:

```
ratings["Rating_Category"].value_counts()
```

Out[25]:

```
Good             348971
Average          261197
Excellent        226310
Below Average    163731
Name: Rating_Category, dtype: int64
```

6.4 Users - ‘Average_Rating_By_User’ and ‘Average_Hours_Spend_By_User’

In [26]:

```
users = users.merge(ratings.groupby("UserID")["Rating"].mean().reset_index(),on="UserID"
users = users.merge(ratings.groupby("UserID")["Rating"].count().reset_index(),on="UserII
users = users.merge(ratings.groupby("UserID")["Watch_Hour"].mean().reset_index(),on="Use
users.head()
```

Out[26]:

	UserID	Gender	Age	Occupation	Zip-code	Rating_x	Rating_y	Watch_Hour
0	1	F	Under 18	K-12 student	48067	4.19	53	22.25
1	2	M	56+	self-employed	70072	3.71	129	21.16
2	3	M	25-34	scientist	55117	3.90	51	21.00
3	4	M	45-49	executive/managerial	02460	4.19	21	20.00
4	5	M	25-34	writer	55455	3.15	198	6.02

In [27]:

```
# Re-Naming columns with appropriate names
users.rename(columns = {'Rating_x':'Average_Rating_By_User','Rating_y':'Number_Of_Rating
users=users[['UserID', 'Gender', 'Age', 'Occupation', 'Zip-code','Average_Rating_By_User
users.head()
```

Out[27]:

	UserID	Gender	Age	Occupation	Zip-code	Average_Rating_By_User	Number_Of_Ratings_Given_By_User
0	1	F	Under 18	K-12 student	48067	4.19	53
1	2	M	56+	self-employed	70072	3.71	129
2	3	M	25-34	scientist	55117	3.90	51
3	4	M	45-49	executive/managerial	02460	4.19	21
4	5	M	25-34	writer	55455	3.15	198

7. Merging the data and creating a single consolidated dataframe

In [28]:

```
df = ratings[['UserID', 'MovieID', 'Rating','Watch_Hour','Watch_Month', 'Rating_Category
#df = ratings[['UserID', 'MovieID', 'Rating']].copy()
df = df.merge(users,how="right",on="UserID")
df = df.merge(m.reset_index(),how="right",on="MovieID")
X = df.drop(columns = ['UserID', 'MovieID'])
y = df.pop('Rating')
```

In [29]:

```
X.columns
```

Out[29]:

```
Index(['Rating', 'Watch_Hour', 'Watch_Month', 'Rating_Category', 'Gender',
       'Age', 'Occupation', 'Zip-code', 'Average_Rating_By_User',
       'Number_Of_Ratings_Given_By_User', 'Average_Hours_Spend_By_User',
       'Action', 'Adventure', 'Animation', 'Children', 'Comedy', 'Crime',
       'Documentary', 'Drama', 'Fantasy', 'Film-Noir', 'Horror', 'Musical',
       'Mystery', 'Romance', 'Sci-Fiction', 'Thriller', 'Unknown', 'War',
       'Western'],
      dtype='object')
```

In [30]:

```
print(X.shape)
print(y.shape)
```

```
(1000386, 30)
(1000386,)
```

In [31]:

```
X.head()
```

Out[31]:

	Rating	Watch_Hour	Watch_Month	Rating_Category	Gender	Age	Occupation	Zip-code	Average_Ratin
0	5.00	23.00	1.00	Excellent	F	Under 18	K-12 student	48067	
1	4.00	4.00	12.00	Good	F	50-55	homemaker	55117	
2	4.00	3.00	12.00	Good	M	25-34	programmer	11413	
3	5.00	1.00	12.00	Excellent	M	25-34	technician/engineer	61614	
4	5.00	1.00	12.00	Excellent	F	35-44	academic/educator	95370	

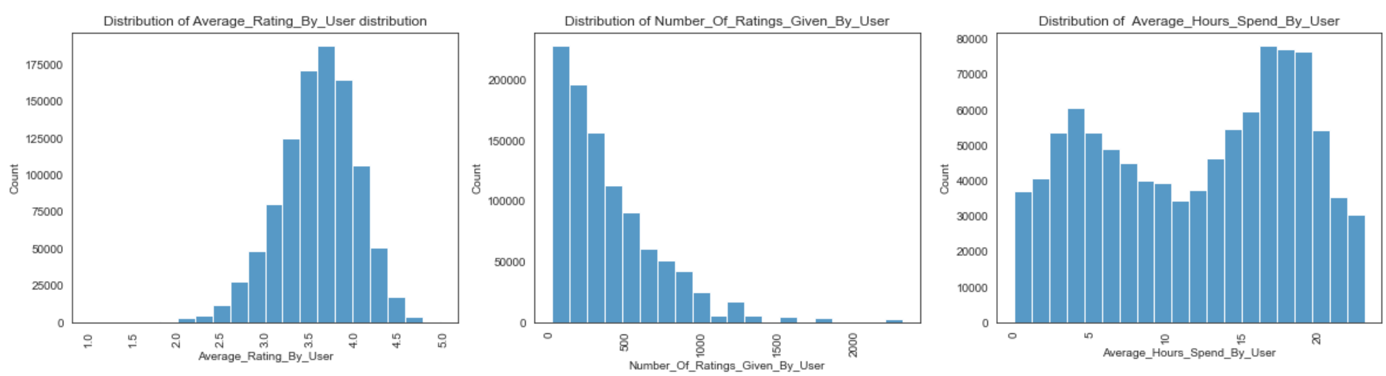
8. Exploratory Data Analysis

- Reviewing the shape and structure of the dataset
- Investigating the data for any inconsistency
- Group the data according to the average rating and no. of ratings

8.1 Distribution of Average Rating ,Number of Rating and Avg Hour Spend

In [32]:

```
plt.figure(figsize=(20, 10))
plt.subplot(2, 3, 1)    # Define 2 rows, 3 column, Activate subplot 1.
sp = sns.histplot(X["Average_Rating_By_User"], bins=20)
sp.set(title='Distribution of Average_Rating_By_User distribution')
plt.xticks(rotation=90)
plt.subplot(2, 3, 2)    # Define 2 rows, 3 column, Activate subplot 2.
sp = sns.histplot(X["Number_Of_Ratings_Given_By_User"], bins=20)
sp.set(title='Distribution of Number_Of_Ratings_Given_By_User')
plt.xticks(rotation=90)
plt.subplot(2, 3, 3)    # Define 2 rows, 3 column, Activate subplot 3.
sp = sns.histplot(X["Average_Hours_Spend_By_User"], bins=20)
sp.set(title='Distribution of  Average_Hours_Spend_By_User')
plt.xticks(rotation=90)
plt.show()
```

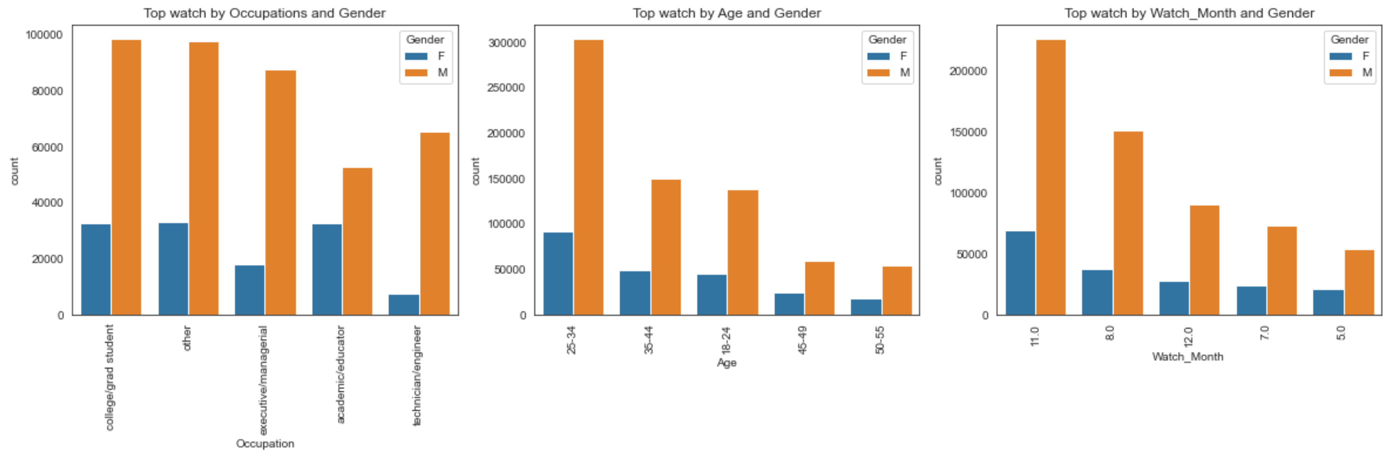


- **Insights**
  - Movies beyond Average ratings (between 3.0 to 4.0) has been watched mostly
  - Most watch are during afreenoon , evening hours

8.2 Distribution by Occupation, Age , Watch Month and Gender

In [33]:

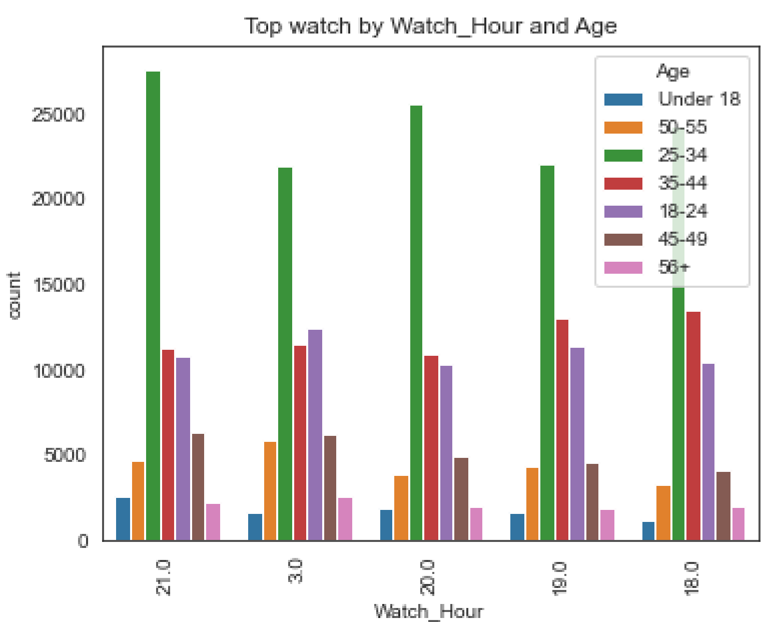
```
plt.figure(figsize=(20, 10))
plt.subplot(2, 3, 1)    # Define 2 rows, 3 column, Activate subplot 1.
sp = sns.countplot(x="Occupation",data=X,hue="Gender",order=X.Occupation.value_counts().
sp.set(title='Top watch by Occupations and Gender')
plt.xticks(rotation=90)
plt.subplot(2, 3, 2)    # Define 2 rows, 3 column, Activate subplot 2.
sp = sns.countplot(x="Age",data=X,hue="Gender",order=X.Age.value_counts().iloc[:5].index
sp.set(title='Top watch by Age and Gender')
plt.xticks(rotation=90)
plt.subplot(2, 3, 3)    # Define 2 rows, 3 column, Activate subplot 3.
sp = sns.countplot(x="Watch_Month",data=X,hue="Gender",order=X.Watch_Month.value_counts
sp.set(title='Top watch by Watch_Month and Gender')
plt.xticks(rotation=90)
plt.show()
```



- **Insights**
  - **College / Grad Students and Executive/Managerial professionals watches most** movies
  - **Age group 25-34 watches most movies**
  - **Top watchs are during November,August and December** month

8.3 Distribution by Watch_Hour, Age

In [34]:
```
plt.figure(figsize=(20, 10))
plt.subplot(2, 3, 1)    # Define 2 rows, 3 column, Activate subplot 1.
sp = sns.countplot(x="Watch_Hour",data=X,hue="Age",order=X.Watch_Hour.value_counts().ilo
sp.set(title='Top watch by Watch_Hour and Age')
plt.xticks(rotation=90)
plt.show()
```



- **Insights**
  - **Top watchs are during 7 to 9 PM**
  - Under 18 generally watches during 9 PM

9. Recommender System based on Pearson Correlation

9.1 Data Preparation - Title and Ratings

In [35]:

```
rating_selected = ratings[["UserID","MovieID","Rating"]].copy()
movies_selected = movies[["MovieID","Title"]].copy()
movies_rating_merged=rating_selected.merge(movies_selected,on="MovieID")
movies_rating_merged.head()
```

Out[35]:

	UserID	MovieID	Rating	Title
0	1	1193	5	One Flew Over the Cuckoo's Nest (1975)
1	2	1193	5	One Flew Over the Cuckoo's Nest (1975)
2	12	1193	4	One Flew Over the Cuckoo's Nest (1975)
3	15	1193	4	One Flew Over the Cuckoo's Nest (1975)
4	17	1193	5	One Flew Over the Cuckoo's Nest (1975)

9.2 Item Based Pearson Correlation

In [647…

```
def item_based_recommender_pearson_correlation_similarity(input_df,input_movie_name,top_
    # Creating pivot based on item movie Title
    pivot_item_based = pd.pivot_table(input_df,
                                      index='Title',
                                      columns=['UserID'], values='Rating')
    # Clipped data due to limited resource in laptop
    data_selected= movies_rating_pt.iloc[:, : 1000]
    # Calculating correlation matrix
    item_recommender_correlation_matrix = data_selected.corr().fillna(0)
    recommender_df = pd.DataFrame(item_recommender_correlation_matrix,
                                  columns=pivot_item_based.index,
                                  index=pivot_item_based.index)
    ## Creating recommendation based on filter movie title
    recommended_correlation_df = pd.DataFrame(recommender_df[input_movie_name].sort_valu
    recommended_correlation_df.reset_index(level=0, inplace=True)
    recommended_correlation_df.columns = ['Title','title_pearson_correlation_similarity

    return recommended_correlation_df
```

9.2.1 List of 5 other Movies to recommend who watched '20 Dates (1998)' Movie

In [648…

```
item_based_recommender_pearson_correlation_similarity(movies_rating_merged, "20 Dates (1
```

Out[648]:

	Title	title_pearson_correlation_similarity
0	American Pie (1999)	0.19
1	Cruel Intentions (1999)	0.19
2	Big Daddy (1999)	0.17
3	Arlington Road (1999)	0.16
4	Austin Powers: The Spy Who Shagged Me (1999)	0.16

9.2 User Based Pearson Correlation

In [649…
```
def user_based_recommender_pearson_correlation_similarity(input_df,input_user_name,top_r
    # Creating pivot based on user
    pivot_user_based = pd.pivot_table(input_df,
                                      index='UserID',
                                      columns=['Title'], values='Rating')
    # Calculating correlation matrix
    user_correlation_matrix = pivot_user_based.corr().fillna(0)
    user_correlation_matrix_df = pd.DataFrame(user_correlation_matrix,
                                  columns=pivot_user_based.index,
                                  index=pivot_user_based.index)
    ## Creating recommendation based on filter user
    user_recommended_correlation_df = pd.DataFrame(user_correlation_matrix_df[input_use
    user_recommended_correlation_df.reset_index(level=0, inplace=True)
    user_recommended_correlation_df.columns = ['UserID','User_pearson_correlation_simila

    return user_recommended_correlation_df
```

In [650…
```
user_based_recommender_pearson_correlation_similarity(movies_rating_merged,"15",top_n=5)
```

Out[650]:

	UserID	User_pearson_correlation_similarity
0	10	NaN
1	100	NaN
2	1000	NaN
3	1001	NaN
4	1002	NaN

10. Recommender System based on Cosine Similarity

Use the Item-based approach to create a recommender system that uses Nearest Neighbors algorithm and Cosine Similarity

In [37]:
```
rating_selected = ratings[["UserID","MovieID","Rating"]].copy()
movies_selected = movies[["MovieID","Title","Genres"]].copy()
movie_ratings_df=rating_selected.merge(movies_selected,on="MovieID")
```

10.1. Movie Title Based Cosine Similarity

In [652…
```
def item_based_recommender_consine_similarity(input_df,input_movie_name):
    pivot_item_based = pd.pivot_table(input_df,
                                      index='Title',
                                      columns=['UserID'], values='Rating')
    sparse_pivot_ib = csr_matrix(pivot_item_based.fillna(0))
    item_recommender_cosine_matrix = cosine_similarity(sparse_pivot_ib)
    recommender_df = pd.DataFrame(item_recommender_cosine_matrix,
                                  columns=pivot_item_based.index,
                                  index=pivot_item_based.index)
    ## Item Rating Based Cosine Similarity
    cosine_df = pd.DataFrame(recommender_df[input_movie_name].sort_values(ascending=Fals
    cosine_df.reset_index(level=0, inplace=True)
    cosine_df.columns = ['Title','title_cosine_similarity']
    return cosine_df
```

10.2. User Based Cosine Similarity

In [653…

```
def user_based_recommender_consine_similarity(input_df,input_user_id):
    pivot_user_based = pd.pivot_table(input_df, index='UserID', columns=['Title'], value
    sparse_pivot_ub = csr_matrix(pivot_user_based.fillna(0))
    user_recomm_cosine_matrix = cosine_similarity(sparse_pivot_ub)
    user_recomm_df = pd.DataFrame(user_recomm_cosine_matrix,columns=pivot_user_based.inc
                 index=pivot_user_based.index.values)
    ## User Rating Based Cosine Similarity
    usr_cosine_df = pd.DataFrame(user_recomm_df[input_user_id].sort_values(ascending=Fal
    usr_cosine_df.reset_index(level=0, inplace=True)
    usr_cosine_df.columns = ['UserID','user_cosine_similarity']
    return usr_cosine_df
```

In [654…

```
user_based_cosine_similarity_recommendations = user_based_recommender_consine_similarity
user_based_cosine_similarity_recommendations[:5]
```

Out[654]:

	UserID	user_cosine_similarity
0	4	1.00
1	4143	0.51
2	1575	0.46
3	5876	0.45
4	562	0.45

10.4. Final Cosine Similarity Recommender - Item and User combined

In [655…

```
def show_recomendations(input_movies_rated_df,input_movie_name,input_user_id,top_n=5):
    print("Recomendation similar to Movie -> ", input_movie_name)
    ## Item Rating Based Cosine Similarity
    cos_sim_df = item_based_recommender_consine_similarity(input_movies_rated_df,input_r
    display(cos_sim_df[1:top_n+1])

    ## User Based Cosine Similarity
    print("Movies reccomended for User -> ",input_user_id)
    display(user_based_recommender_consine_similarity(input_movies_rated_df,input_user_

    return None
```

In [656…

```
show_recomendations(movie_ratings_df,"Jumanji (1995)","4")
```

```
Recomendation similar to Movie ->  Jumanji (1995)
```

	Title	title_cosine_similarity
1	Hook (1991)	0.57
2	Dragonheart (1996)	0.50
3	Indian in the Cupboard, The (1995)	0.48
4	Honey, I Shrunk the Kids (1989)	0.48
5	NeverEnding Story, The (1984)	0.48

```
Movies reccomended for User ->  4
```

	UserID	user_cosine_similarity
1	4143	0.51
2	1575	0.46
3	5876	0.45
4	562	0.45
5	87	0.45

10.5. Recommender - KNN and Consine

In [38]:
```
movie_ratings_df.head()
```

Out[38]:

	UserID	MovieID	Rating	Title	Genres
0	1	1193	5	One Flew Over the Cuckoo's Nest (1975)	Drama
1	2	1193	5	One Flew Over the Cuckoo's Nest (1975)	Drama
2	12	1193	4	One Flew Over the Cuckoo's Nest (1975)	Drama
3	15	1193	4	One Flew Over the Cuckoo's Nest (1975)	Drama
4	17	1193	5	One Flew Over the Cuckoo's Nest (1975)	Drama

10.5.1 Data preperation collaborative filtering

In [39]:
```
## The Reader class is used to parse a file containing ratings.It orders the data in fo
## and even by considering the rating scale
reader = Reader(rating_scale=(0.5 , 5))
# The columns must correspond to user id, item id and ratings (in that order).
data = Dataset.load_from_df(movie_ratings_df[['UserID','MovieID','Rating']], reader) # .
```

In [40]:
```
anti_set = data.build_full_trainset().build_anti_testset()
```

- An antiset is a set of those user and item pairs for which a rating doesn't exist in original dataset. This is the set for which we are trying to predict ratings.
- Surprise creates a set of such combinations by providing a default average rating. We'll be calculating an estimated rating for this set using our model.

In [41]:
```
trainset, testset = train_test_split(data, test_size=.15) # Splitting the data
```

10.5.2 User based collaborative filtering

In [42]:
```
algo = KNNWithMeans(k = 50, sim_options={'name': 'cosine', 'user_based': True})

# K value represents the (max) number of neighbors to take into account for aggregation
# There are many similarity options to calculate the similarity between the neighbors. I
# when user_based = True then it performs user based collaborative filtering

algo.fit(trainset) #fitting the train dataset
```

```
Computing the cosine similarity matrix...
Done computing similarity matrix.
```

Out[42]:

```
<surprise.prediction_algorithms.knns.KNNWithMeans at 0x21f80d4d210>
```

In [43]:

```python
# run the trained model against the testset
test_pred = algo.test(testset)
```

In [44]:

```python
test_pred[0]
```

Out[44]:

```
Prediction(uid='2526', iid='1097', r_ui=3.0, est=3.9993750355742055, details={'actual_
k': 50, 'was_impossible': False})
```

- uid – The (raw) user id.
- iid – The (raw) item id.
- r_ui (float) – The true rating .
- est (float) – The estimated rating. This is calculated by taking mean ratings of each item for item-based collab filtering.
- details (dict) – Stores additional details about the prediction.
- In this details was_impossible defines status of the true rating
  - if was_impossible: False - Then there is some true rating.
  - else if was_impossible: True - Then there is no information on true rating for that particular record.

In [45]:

```python
# get RMSE and MAE on test set
print("User-based Model : Test Set")
accuracy.rmse(test_pred, verbose=True)
accuracy.mae(test_pred, verbose=True)
```

```
User-based Model : Test Set
RMSE: 0.9379
MAE:  0.7470
```

Out[45]:

```
0.7469935573830148
```

In [46]:

```python
movie_ratings_df.head()
```

Out[46]:

	UserID	MovieID	Rating	Title	Genres
0	1	1193	5	One Flew Over the Cuckoo's Nest (1975)	Drama
1	2	1193	5	One Flew Over the Cuckoo's Nest (1975)	Drama
2	12	1193	4	One Flew Over the Cuckoo's Nest (1975)	Drama
3	15	1193	4	One Flew Over the Cuckoo's Nest (1975)	Drama
4	17	1193	5	One Flew Over the Cuckoo's Nest (1975)	Drama

In [47]:

```python
# we can query for specific predicions
uid = str(15)   # raw user id
iid = str(1193)   # raw item id

# get a prediction for specific users and items.
pred = algo.predict(uid, iid, verbose=True)
```

```
user: 15         item: 1193       r_ui = None   est = 3.91   {'actual_k': 50, 'was_impos
sible': False}
```

- For this user `15` for movie `1193` the true rating is None where as the estimated rating is `3.91`

In [ ]:
```
anti_pre = algo.test(anti_set)
pred_df = pd.DataFrame(anti_pre).merge(movies , left_on = ['iid'], right_on = ['MovieID
pred_df = pd.DataFrame(pred_df).merge(users , left_on = ['uid'], right_on = ['UserID'])
pred_df.head()
```

In [ ]:
```
pred_df[(pred_df['est']== 5.0)&(pred_df['UserID']== 200)]
```

10.5.3 Item based collaborative filtering

In [ ]:
```
# K value represents the (max) number of neighbors to take into account for aggregation
# There are many similarity options to calculate the similarity between the neighbors .
# when user_based = False then it performs item based collaborative filtering

algo_i = KNNWithMeans(k=50, sim_options={'name': 'cosine', 'user_based': False})
algo_i.fit(trainset)
```

In [ ]:
```
# run the trained model against the testset
test_pred = algo_i.test(testset)
test_pred[0]
```

In [ ]:
```
# get RMSE on test set
print("Item-based Model : Test Set")
accuracy.rmse(test_pred, verbose=True)
accuracy.mae(test_pred, verbose=True)
```

In [ ]:
```
# we can query for specific predicions
uid = str(196)  # raw user id
iid = str(303)  # raw item id
```

In [ ]:
```
# get a prediction for specific users and items.
pred = algo_i.predict(uid, iid, verbose=True)
```

In [ ]:
```
tsr_inner_id = algo_i.trainset.to_inner_iid(1) # Considering the movieId 1
tsr_neighbors = algo_i.get_neighbors(tsr_inner_id, k=5) #Getting the 5 nearest neighbors
movies[movies.movieId.isin([algo.trainset.to_raw_iid(inner_id)
                       for inner_id in tsr_neighbors])] #Displaying the 5 nearest neighb
```

11. Recommender System based on Matrix Factorization

11.1 Using CMF

In [657…
```
movie_ratings_df = movies_rating_merged[['UserID', 'MovieID', 'Rating']].copy()
movie_ratings_df.columns = ['UserId', 'ItemId', 'Rating']  # Lib requires specific colur
movie_ratings_df.head(2)
```

Out[657]:

	UserId	ItemId	Rating
0	1	1193	5
1	2	1193	5

In [658…

```
model = CMF(k=4, lambda_=0.1, user_bias=False, item_bias=False, verbose=False)
model.fit(movie_ratings_df)
```

Out[658]:

```
Collective matrix factorization model
(explicit-feedback variant)
```

In [659…

```
top_items = model.topN(user=4, n=10)
movies.loc[movies.MovieID.isin(top_items)]
```

Out[659]:

	MovieID	Title	Genres	release_year
36	37	Across the Sea of Time (1995)	Documentary	1995
108	110	Braveheart (1995)	Action|Drama|War	1995
352	356	Forrest Gump (1994)	Comedy|Romance|War	1994
724	733	Rock, The (1996)	Action|Adventure|Thriller	1996
770	780	Independence Day (ID4) (1996)	Action|Sci-Fiction|War	1996
801	811	Bewegte Mann, Der (1994)	Comedy	1994
1023	1036	Die Hard (1988)	Action|Thriller	1988
3078	3147	Green Mile, The (1999)	Drama|Thriller	1999
3509	3578	Gladiator (2000)	Action|Drama	2000
3684	3753	Patriot, The (2000)	Action|Drama|War	2000

11.2 Using surprise

11.2.1 Read and Load Data

In [660…

```
movie_ratings_df = movies_rating_merged[['UserID', 'MovieID', 'Rating']].copy()
reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(movie_ratings_df[['UserID', 'MovieID', 'Rating']], reader)
```

11.2.2 Train Test Data Split

In [661…

```
trainset, testset = train_test_split(data, test_size=.25)
```

11.2.3 Modelling

In [662…

```
svd_model = SVD() # Suprise library uses the SVD algorithm to perform the matrix factor
svd_model.fit(trainset) ## Fitting the trainset with the help of svd
```

Out[662]:

```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x140198ed3f0>
```

In [663…

```
svd_model.pu.shape , svd_model.qi.shape # pu gives the embeddings of Users and qi gives
```

Out[663]:
```
((6040, 100), (3661, 100))
```

11.2.4 Model Prediction

In [664…

```
predictions = svd_model.test(testset)
predictions_df = pd.DataFrame(predictions)
predictions_df.sort_values(by='est', ascending=False)[0:10] ## Sorting the values based
```

Out[664]:

	uid	iid	r_ui	est	details
4013	5483	1204	5.00	5.00	{'was_impossible': False}
113596	412	1198	5.00	5.00	{'was_impossible': False}
229292	2761	1204	5.00	5.00	{'was_impossible': False}
120165	4040	296	5.00	5.00	{'was_impossible': False}
139131	5056	913	5.00	5.00	{'was_impossible': False}
229333	2726	1207	5.00	5.00	{'was_impossible': False}
191387	692	750	5.00	5.00	{'was_impossible': False}
103622	4708	1208	5.00	5.00	{'was_impossible': False}
103619	4406	1196	5.00	5.00	{'was_impossible': False}
242154	3226	858	5.00	5.00	{'was_impossible': False}

11.2.5 Evaluate Model

11.2.5.1 RMSE

In [665…

```
accuracy.rmse(predictions)
```

```
RMSE: 0.8775
```

Out[665]:
```
0.8774553840519983
```

11.2.5.2 MAE

In [666…

```
accuracy.mae(predictions)
```

```
MAE:  0.6893
```

Out[666]:
```
0.689298950024674
```

11.2.5.3 MAPE

In [667…

```
test_ratings = list(map(lambda x: x[2], testset))
predictions_ratings = list(map(lambda x: x[2], predictions))
mean_absolute_percentage_error(test_ratings,predictions_ratings)
```

Out[667]:
```
0.0
```

11.2.5.4 Cross validation

In [668…

```
cross_validate(svd_model, data, measures=['RMSE', 'MAE'], cv=5, return_train_measures=Tr
##The dataset is divided into train and test and with 5 folds the rmse has been calcula
```

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

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.8718  0.8754  0.8728  0.8763  0.8739  0.8740  0.0016
MAE (testset)     0.6843  0.6872  0.6858  0.6874  0.6863  0.6862  0.0011
RMSE (trainset)   0.6702  0.6694  0.6695  0.6702  0.6699  0.6699  0.0003
MAE (trainset)    0.5300  0.5293  0.5295  0.5301  0.5297  0.5297  0.0003
Fit time          11.62   10.78   11.54   11.81   13.33   11.81   0.84
Test time         2.08    2.13    1.49    1.77    2.30    1.95    0.29
```

Out[668]:

```
{'test_rmse': array([0.8718129 , 0.87543005, 0.87280114, 0.87628823, 0.8738596 ]),
 'train_rmse': array([0.67023198, 0.66943849, 0.66949858, 0.67018701, 0.66990383]),
 'test_mae': array([0.68430187, 0.68715287, 0.68584865, 0.68743157, 0.68630198]),
 'train_mae': array([0.5299862 , 0.52928535, 0.5295014 , 0.53010523, 0.52969141]),
 'fit_time': (11.615907192230225,
  10.778180837631226,
  11.540077447891235,
  11.809574127197266,
  13.33030891418457),
 'test_time': (2.082003116607666,
  2.1287100315093994,
  1.48716402053833,
  1.7748198509216309,
  2.2995760440826416)}
```

- The above data gives the RMSE and MAE values for each fold as well as average value and standard deviation value.
  - `test_rmse` represents the rmse values of testsets.
  - `train_rmse` represents the rmse values of trainsets.
  - similarly, `test_mae` and `train_mae` represents MAE values of train and testsets.
  - `fit_time` represents time taken to fit the trainsets.
  - `test_time` represents time taken to fit the testsets.

11.2.6 Tune Model

In [669…

```
param_grid = {'n_epochs': [5, 10], 'lr_all': [0.002, 0.005]}

gs = GridSearchCV(SVD,
                  param_grid,
                  measures=['rmse', 'mae'],
                  cv=3,
                  n_jobs=-1,
                  joblib_verbose=True)

gs.fit(data)
gs.best_score['rmse']
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  10 out of  12 | elapsed:   35.2s remaining:    7.0s
[Parallel(n_jobs=-1)]: Done  12 out of  12 | elapsed:   38.1s finished
```

Out[669]:

```
0.9010816110736591
```

11.2.7 Embeddings for item-item and user-user similarity

11.2.7.1 User embeddings

In [670…

```
svd_model.pu
```

Out[670]:
```
array([[-0.02472445, -0.07541024, -0.16676115, ...,  0.06821624,
        -0.12928861,  0.25744438],
       [ 0.0330547 ,  0.10515517, -0.08219863, ..., -0.07466421,
         0.09247859, -0.12374525],
       [ 0.31981572,  0.10381752,  0.17992462, ..., -0.11041203,
        -0.14705581, -0.07336445],
       ...,
       [-0.11315691,  0.03622798, -0.0773759 , ...,  0.13683511,
        -0.05202012, -0.0392672 ],
       [ 0.00684156,  0.01411961, -0.18696687, ..., -0.15472687,
        -0.02600178,  0.30740056],
       [ 0.15370876,  0.1228381 , -0.059839  , ..., -0.07533753,
         0.08947219,  0.07665322]])
```

11.2.7.2 Item embeddings

In [671…
```
svd_model.qi
```

Out[671]:
```
array([[-6.46000703e-02,  1.59097597e-01, -2.00439498e-01, ...,
        -1.20867763e-01, -3.69273485e-01, -9.40357941e-02],
       [ 1.92578679e-01, -9.51120308e-02, -4.99378514e-01, ...,
        -4.72740983e-02,  1.53323090e-04, -1.71659150e-01],
       [ 1.91724983e-01,  6.39249634e-02, -2.01731758e-01, ...,
        -3.15579591e-02, -2.23347436e-01,  3.07462898e-01],
       ...,
       [ 3.88207983e-02,  9.89556283e-03,  8.82846611e-02, ...,
         1.97888417e-02, -1.11037381e-01,  7.21326750e-02],
       [-3.86896901e-02,  8.08610946e-02,  1.08635141e-01, ...,
        -6.95538479e-02, -2.21432764e-02,  4.72948868e-03],
       [ 1.21705985e-01, -9.94692270e-02, -1.62222956e-01, ...,
        -5.80745150e-02, -1.07569286e-01,  2.10988970e-02]])
```

13. Questionnaire

13.1 Users of which age group have watched and rated the most number of movies?

In [672…
```
users.groupby(['Age'])['Number_Of_Ratings_Given_By_User'].count().sort_values(ascending=
```

Out[672]:
```
Age
25-34       2096
35-44       1193
18-24       1103
45-49        550
50-55        496
56+          380
Under 18     222
Name: Number_Of_Ratings_Given_By_User, dtype: int64
```

- Users of **age group "25-34"** have **watched and rated the most number of movies**

13.2 Users belonging to which profession have watched and rated the most movies?

In [673…
```
users.groupby(['Occupation'])['Number_Of_Ratings_Given_By_User'].count().sort_values(asc
```

Out[673]:
```
Occupation
college/grad student    759
other                   711
executive/managerial    679
academic/educator       528
technician/engineer     502
programmer              388
sales/marketing         302
writer                  281
artist                  267
self-employed           241
doctor/health care      236
K-12 student            195
clerical/admin          173
scientist               144
retired                 142
lawyer                  129
customer service        112
homemaker                92
unemployed               72
tradesman/craftsman      70
farmer                   17
Name: Number_Of_Ratings_Given_By_User, dtype: int64
```

- Users of **profession "college/grad student"** have **watched and rated the most number of movies**

13.3 Most of the users in our dataset who've rated the movies are Male. (T/F)

In [674…
```
users.groupby(['Gender'])['Number_Of_Ratings_Given_By_User'].count().sort_values(ascend:
```

Out[674]:
```
Gender
M    4331
F    1709
Name: Number_Of_Ratings_Given_By_User, dtype: int64
```

- Yes , Most of the users in our dataset who've rated the movies are Male

13.4 Most of the movies present in our dataset were released in which decade?

```
70s b. 90s c. 50s d.80s
```

In [675…
```
movie_release_df = movies[["MovieID","release_year"]]
movie_release_df["release_decade"]= movie_release_df.release_year.str[2].fillna(0).astyp
movie_release_df.head()
```

Out[675]:

	MovieID	release_year	release_decade
0	1	1995	90
1	2	1995	90
2	3	1995	90
3	4	1995	90
4	5	1995	90

In [676…
```
movie_release_df["release_decade"].value_counts().head(4)
```

Out[676]:
```
90    2274
80     595
70     244
60     189
Name: release_decade, dtype: int64
```

- Most of the movies present in our dataset were released in 90s decade

13.5 The movie with maximum no. of ratings is ___.

In [677…
```
max_high_rated_movie_id = ratings[ratings["Rating"] == "5"]["MovieID"].value_counts().he
max_high_rated_movie_id
```

Out[677]:
```
Series([], Name: MovieID, dtype: int64)
```

In [678…
```
movies[movies["MovieID"] =="2858"]
```

Out[678]:

	MovieID	Title	Genres	release_year
2789	2858	American Beauty (1999)	Comedy|Drama	1999

- "American Beauty (1999)" has maximum number of ratings

13.6 Name the top 3 movies similar to 'Liar Liar' on the item-based approach.

In [679…
```
movies[movies["Title"].str.contains("Liar Liar")]
```

Out[679]:

	MovieID	Title	Genres	release_year
1455	1485	Liar Liar (1997)	Comedy	1997

In [680…
```
item_based_recommender_consine_similarity(movies_rating_merged,"Liar Liar (1997)")[1:4]
```

Out[680]:

	Title	title_cosine_similarity
1	Mrs. Doubtfire (1993)	0.56
2	Ace Ventura: Pet Detective (1994)	0.52
3	Dumb & Dumber (1994)	0.51

13.7 On the basis of approach, Collaborative Filtering methods can be classified into ***-based and*** -based.

- Collaborative Filtering methods can be classified into **user**-based and **item**-based

13.8 Pearson Correlation ranges between ***to*** whereas, Cosine Similarity belongs to the interval between ***to*** .

Pearson Correlation ranges between **-1 to 1** whereas, Cosine Similarity belongs to the interval between **-1 to 1**.

13.9 Mention the RMSE and MAPE that you got while evaluating the Matrix Factorization model.

In [681…

```
print("RMSE --> ",accuracy.rmse(predictions))
print("MAPE --> ", mean_absolute_percentage_error(test_ratings,predictions_ratings))
```

```
RMSE: 0.8775
RMSE -->  0.8774553840519983
MAPE -->  0.0
```

13.10 Give the sparse ‘row’ matrix representation for the following dense matrix -

```
[[1 0]
[3 7]]
```

In [682…

```
dense_mat= []
dense_mat = [[0 for _ in range(2)] for _ in range(2)]
dense_mat[0][0], dense_mat[0][1] = 1,0
dense_mat[1][0], dense_mat[1][1] = 3,7
sparse_mat = csr_matrix(dense_mat)
sparse_mat
```

Out[682]:

```
<2x2 sparse matrix of type '<class 'numpy.intc'>'
	with 3 stored elements in Compressed Sparse Row format>
```