In [1]:
```
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
from collections import defaultdict
from scipy import sparse
from scipy.stats import pearsonr
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.neighbors import NearestNeighbors
import warnings
# from cmfrec import CMF
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.metrics import mean_squared_error
from surprise import Reader, Dataset, SVD
from surprise.model_selection import cross_validate

```

DESCRIPTIONS

RATINGS FILE DESCRIPTION

==

All ratings are contained in the file "ratings.dat" and are in the following format:

UserID::MovieID::Rating::Timestamp

- UserIDs range between 1 and 6040
- MovieIDs range between 1 and 3952
- Ratings are made on a 5-star scale (whole-star ratings only)
- Timestamp is represented in seconds
- Each user has at least 20 ratings

USERS FILE DESCRIPTION

==

User information is in the file "users.dat" and is in the following format:

UserID::Gender::Age::Occupation::Zip-code

All demographic information is provided voluntarily by the users and is not checked for accuracy.
Only users who have provided some demographic information are included in this data set.

- Gender is denoted by a "M" for male and "F" for female
- Age is chosen from the following ranges:
  - 1: "Under 18"
  - 18: "18-24"
  - 25: "25-34"
  - 35: "35-44"
  - 45: "45-49"
  - 50: "50-55"
  - 56: "56+"

- 0: "other" or not specified
- 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"

MOVIES FILE DESCRIPTION

==

Movie information is in the file "movies.dat" and is in the following format:

MovieID::Title::Genres

- Titles are identical to titles provided by the IMDB (including year of release)
- Genres are pipe-separated and are selected from the following genres:
  - Action
  - Adventure
  - Animation
  - Children's
  - Comedy
  - Crime
  - Documentary
  - Drama
  - Fantasy
  - Film-Noir
  - Horror
  - Musical
  - Mystery
  - Romance
  - Sci-Fi
  - Thriller
  - War
  - Western

HOW TO THINK AND WHAT TO LOOK FOR

1. Users of which age group have watched and rated the most number of movies?
2. Users belonging to which profession have watched and rated the most movies?
3. Most of the users in our dataset who've rated the movies are Male. (T/F)
4. Most of the movies present in our dataset were released in which decade?
    1. 70s b. 90s c. 50s d.80s
5. The movie with maximum no. of ratings is ___.
6. Name the top 3 movies similar to 'Liar Liar' on the item-based approach.
7. On the basis of approach, Collaborative Filtering methods can be classified into ___-based and ___-based.
8. Pearson Correlation ranges between ___ to ___ whereas, Cosine Similarity belongs to the interval between ___ to ___.
9. Mention the RMSE and MAPE that you got while evaluating the Matrix Factorization model.
10. Give the sparse 'row' matrix representation for the following dense matrix -

    [[1 0]
    [3 7]]

1. DATA

1.1 Movies

In [2]:
```
movies = pd.read_fwf("zee-movies.dat",encoding="ISO-8859-1")

movies.drop(["Unnamed: 1","Unnamed: 2"],axis = 1,inplace=True)

delimiter ="::"

movies = movies["Movie ID::Title::Genres"].str.split(delimiter,expand = True)
movies.columns = ["MovieID","Title","Genres"]

```

In [3]:
```
movies.head()
```
Out[3]:

	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

1.2 Rating

In [4]:
```
ratings =pd.read_fwf("zee-ratings.dat",encoding="ISO-8859-1")

delimiter ="::"

ratings = ratings["UserID::MovieID::Rating::Timestamp"].str.split(delimiter,expand = True)
ratings.columns = ["UserID","MovieID","Rating","Timestamp"]

```
In [5]:
```
ratings.head()
```
Out[5]:

	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 [6]:
```
rating1 = ratings.copy()
```

1.3 Users

In [7]:
```
users = pd.read_fwf("zee-users.dat",encoding="ISO-8859-1")
delimiter ="::"

users = users["UserID::Gender::Age::Occupation::Zip-code"].str.split(delimiter,expand = True)
users.columns = ["UserID","Gender","Age","Occupation","Zipcode"]

```

In [8]:
```
users
```

Out[8]:

	UserID	Gender	Age	Occupation	Zipcode
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
...	...	...	...	...	...
6035	6036	F	25	15	32603
6036	6037	F	45	1	76006
6037	6038	F	56	1	14706
6038	6039	F	45	0	01060
6039	6040	M	25	6	11106

6040 rows × 5 columns

In [9]:
```
users1 = users.copy()
```

As given in the user description we have to change the age group and occupation as well

In [10]:
```
users.replace({'Age' : {'1': "Under 18",
                        '18': "18-24",
                        '25': "25-34",
                        '35': "35-44",
                        '45': "45-49",
                        '50': "50-55",
                        '56': "56+" }} , inplace = True)
```

In [11]:
```python
users.replace({'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 [12]:
```python
users
```

Out[12]:

	UserID	Gender	Age	Occupation	Zipcode
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
...	...	...	...	...	...
6035	6036	F	25-34	scientist	32603
6036	6037	F	45-49	academic/educator	76006
6037	6038	F	56+	academic/educator	14706
6038	6039	F	45-49	other	01060
6039	6040	M	25-34	doctor/health care	11106

6040 rows × 5 columns

2. ANALYSIS

In [13]:
```
movies
```

Out[13]:

	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
...	...	...	...
3878	3948	Meet the Parents (2000)	Comedy
3879	3949	Requiem for a Dream (2000)	Drama
3880	3950	Tigerland (2000)	Drama
3881	3951	Two Family House (2000)	Drama
3882	3952	Contender, The (2000)	Drama|Thriller

3883 rows × 3 columns

In [14]:
```
movies.head()
```

Out[14]:

	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 [15]:
```
movies1 = movies.copy()
```

2.1 Since we have to find out most movie released in which year , we need to extract movie years from Title.

- Using regular expressions to find a year stored between parentheses
- We specify the parantheses so we don't conflict with movies that have years in their titles

In [16]:
```
movies['Year'] = movies.Title.str.extract('(\(\d\d\d\d\))',expand=False)
#Removing the parentheses
movies['Year'] = movies.Year.str.extract('(\d\d\d\d)',expand=False)

#Removing the years from the 'Title' column
movies['Title'] = movies.Title.str.replace('(\(\d\d\d\d\))', '')

#Applying the strip function to get rid of any ending whitespace characters that may have appeared
movies['Title'] = movies['Title'].apply(lambda x: x.strip())
movies.head()
```

```
C:\Users\Acer\AppData\Local\Temp/ipykernel_21484/2592777869.py:6: FutureWarning: The default value of regex will change from True to False in a future version.
  movies['Title'] = movies.Title.str.replace('(\(\d\d\d\d\))', '')
```

Out[16]:

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

In [17]:
```
movies.shape
```

Out[17]: (3883, 4)

In [18]:
```
dfmov = movies.copy()
dfmov.dropna(inplace=True)
dfmov.Genres = dfmov.Genres.str.split('|')
dfmov['Genres'] = dfmov['Genres'].apply(lambda x: [i for i in x if i!='A' and i!='D' and i!= 'F' and i!='C' and i!='M' and i!= 'W' and i!= ' '])
for i in dfmov['Genres']:
    for j in range(len(i)):
        if i[j] == 'Ro' or i[j] == 'Rom' or i[j] == 'Roman' or i[j] == 'R' or i[j] == 'Roma':
            i[j] = 'Romance'
        elif i[j] == 'Chil' or i[j] == 'Childre' or i[j] == 'Childr' or i[j] == "Children'" or i[j] =='Children' or i[j] =='Chi':
            i[j] = "Children's"
        elif i[j] == 'Fantas' or i[j] == 'Fant':
            i[j] = 'Fantasy'
        elif i[j] == 'Dr' or i[j] == 'Dram':
            i[j] = 'Drama'
        elif i[j] == 'Documenta'or i[j] == 'Docu' or i[j] == 'Document' or i[j] == 'Documen':
            i[j] = 'Documentary'
        elif i[j] == 'Wester'or i[j] == 'We':
            i[j] = 'Western'
        elif i[j] == 'Animati':
            i[j] = 'Animation'
        elif i[j] == 'Come'or i[j] == 'Comed' or i[j] == 'Com':
            i[j] = 'Comedy'
        elif i[j] == 'Sci-F'or i[j] == 'S' or i[j] == 'Sci-' or i[j] == 'Sci':
            i[j] = 'Sci-Fi'
        elif i[j] == 'Adv'or i[j] == 'Adventu' or i[j] == 'Adventur' or i[j] == 'Advent':
            i[j] = 'Adventure'
        elif i[j] == 'Horro'or i[j] == 'Horr':
            i[j] = 'Horror'
        elif i[j] == 'Th'or i[j] == 'Thri' or i[j] == 'Thrille':
            i[j] = 'Thriller'
        elif i[j] == 'Acti':
            i[j] = 'Action'
        elif i[j] == 'Wa':
            i[j] = 'War'
        elif i[j] == 'Music':
            i[j] = 'Musical'
dfmov.head()
```

Out[18]:

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

In [19]:
```
movies.dropna(inplace = True)
```

2.2 Merging all the datasets to create the final dataset

In [20]:
```
df = pd.merge(movies, ratings, on = 'MovieID' , how = 'inner' )
df.head()
```

Out[20]:

	MovieID	Title	Genres	Year	UserID	Rating	Timestamp
0	1	Toy Story	Animation|Children's|Comedy	1995	1	5	978824268
1	1	Toy Story	Animation|Children's|Comedy	1995	6	4	978237008
2	1	Toy Story	Animation|Children's|Comedy	1995	8	4	978233496
3	1	Toy Story	Animation|Children's|Comedy	1995	9	5	978225952
4	1	Toy Story	Animation|Children's|Comedy	1995	10	5	978226474

In [21]:
```
data = pd.merge(df, users, on = 'UserID' , how = 'inner')
data.head()
```

Out[21]:

	MovieID	Title	Genres	Year	UserID	Rating	Timestamp	Gender	Age	Occupation	Zipcode
0	1	Toy Story	Animation|Children's|Comedy	1995	1	5	978824268	F	Under 18	K-12 student	48067
1	48	Pocahontas	Animation|Children's|Musical|Romance	1995	1	5	978824351	F	Under 18	K-12 student	48067
2	150	Apollo 13	Drama	1995	1	5	978301777	F	Under 18	K-12 student	48067
3	260	Star Wars: Episode IV - A New Hope	Action|Adventure|Fantas	1977	1	4	978300760	F	Under 18	K-12 student	48067
4	527	Schindler's List	Drama|War	1993	1	5	978824195	F	Under 18	K-12 student	48067

In [22]:
```
data.shape
```

Out[22]:
```
(996144, 11)
```

In [23]:
```
# conda install -c conda-forge scikit-surprise
```

2.3 EDA BASED ON Questionnaire

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

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

FEATURE ENGINEERING

In [24]:
```
data['Year']=data['Year'].astype('int32') #Change the datatype from object to Integer
```

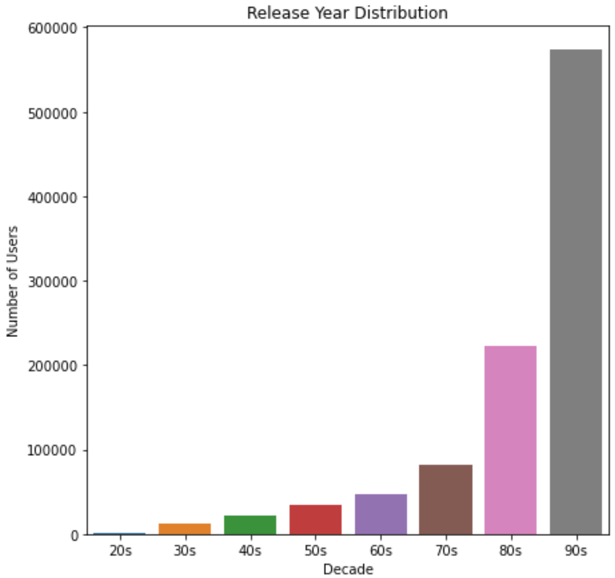
In [25]:
```
bins = [1919, 1929, 1939, 1949, 1959, 1969, 1979, 1989, 2000]
labels = ['20s', '30s', '40s', '50s', '60s', '70s', '80s', '90s']
data['releasedERA'] = pd.cut( data['Year'] , bins = bins , labels= labels)
```

In [26]:
```
data.head()
```

Out[26]:

	MovieID	Title	Genres	Year	UserID	Rating	Timestamp	Gender	Age	Occupation	Zipcode	releasedERA
0	1	Toy Story	Animation|Children's|Comedy	1995	1	5	978824268	F	Under 18	K-12 student	48067	90s
1	48	Pocahontas	Animation|Children's|Musical|Romance	1995	1	5	978824351	F	Under 18	K-12 student	48067	90s
2	150	Apollo 13	Drama	1995	1	5	978301777	F	Under 18	K-12 student	48067	90s
3	260	Star Wars: Episode IV - A New Hope	Action|Adventure|Fantas	1977	1	4	978300760	F	Under 18	K-12 student	48067	70s
4	527	Schindler's List	Drama|War	1993	1	5	978824195	F	Under 18	K-12 student	48067	90s

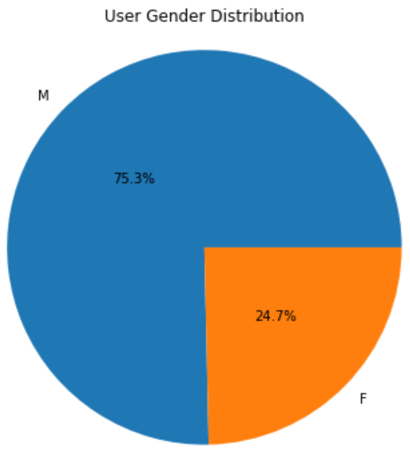
In [27]:
```
plt.figure(figsize=(7, 7))
sns.countplot(x='releasedERA', data=data)
plt.title('Release Year Distribution')
plt.xlabel('Decade')
plt.ylabel('Number of Users')
plt.show()
```



- From the above plot we can infer most of the movies present in the dataset were released in the year 90s.

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

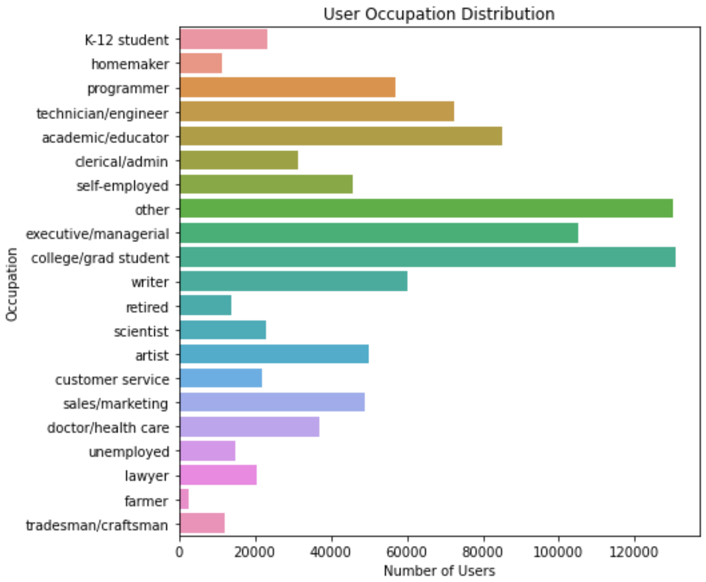
In [28]:
```
x = data['Gender'].value_counts().values
plt.figure(figsize=(7, 6))
plt.pie(x, center=(0, 0), radius=1.5, labels=['M','F'], autopct='%1.1f%%', pctdistance=0.5)
plt.title('User Gender Distribution')
plt.axis('equal')
plt.show()
```



- From the above plot most of the users in our dataset who've rated the movies are Male.

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

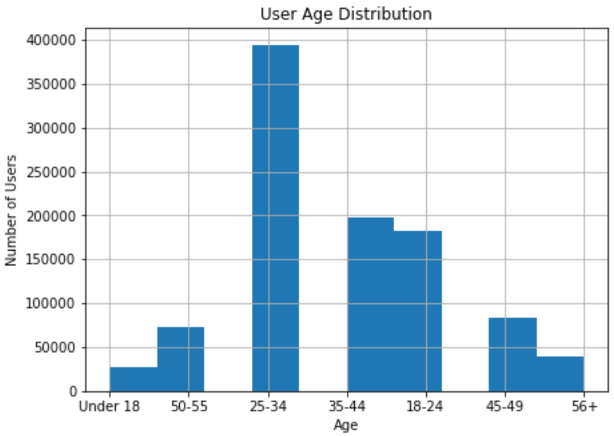
In [29]:
```
plt.figure(figsize=(7, 7))
sns.countplot(y='Occupation', data=data)
plt.title('User Occupation Distribution')
plt.xlabel('Number of Users')
plt.ylabel('Occupation')
plt.show()
```



- From the above plot users belonging to college/grad student profession have watched and rated the most movies.

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

In [30]:
```
data['Age'].hist(figsize=(7, 5))
plt.title('User Age Distribution')
plt.xlabel('Age')
plt.ylabel('Number of Users')
plt.show()
```



- From the above plot we can infer that 25-34 age group have watched and rated the most number of movies

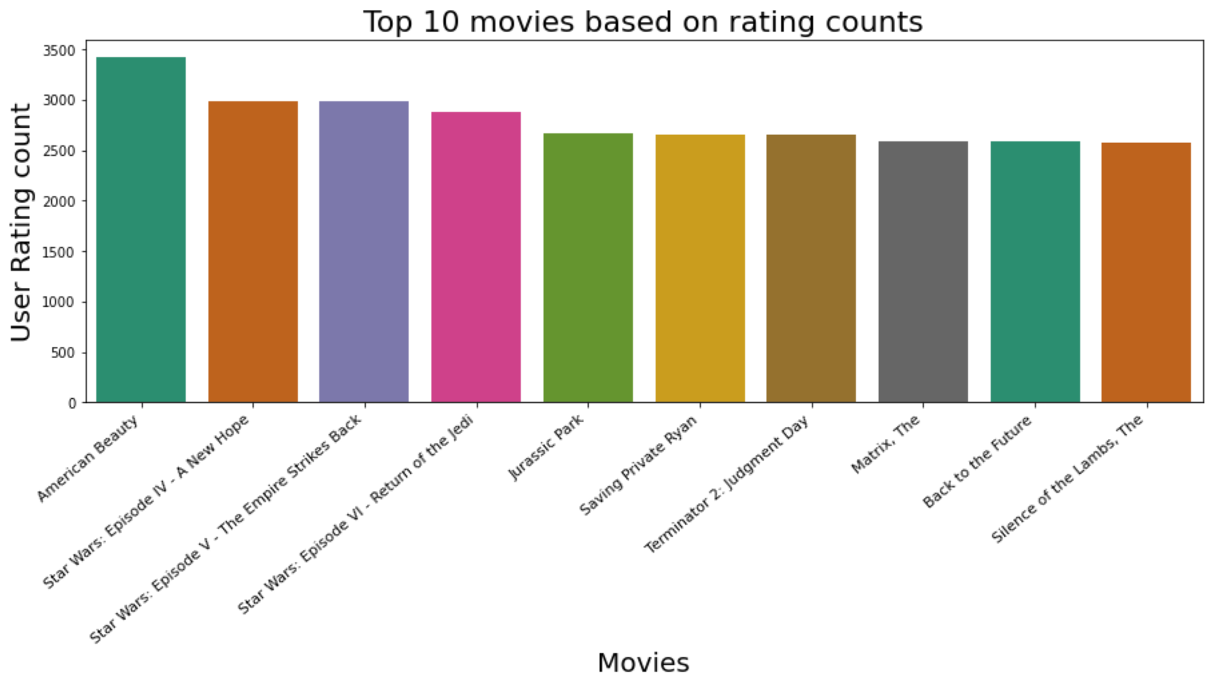
2.3.5 The movie with maximum no. of ratings is

In [31]:
```
## Counting the ratings based on movies
movies_rating_count = data.groupby(by = ['Title'])['Rating'].count().reset_index()

top10_movies=movies_rating_count[['Title', 'Rating']].sort_values(by = 'Rating',ascending = False).head(10)

plt.figure(figsize=(15,5))
ax=sns.barplot(x="Title", y="Rating", data=top10_movies, palette="Dark2")
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('Movies',fontsize = 20)
ax.set_ylabel('User Rating count', fontsize = 20)
```

Out[31]:
```
Text(0, 0.5, 'User Rating count')
```



- From the above plot, the movie with maximum number of ratings is American Beauty.

3. Build a Recommender System based on Pearson Correlation

- 3.1 Creating a pivot table of movie titles & user id and imputing the NaN values
- 3.2 Use the Item-based approach to create a simple recommender system that uses Pearson Correlation

3.1 Creating a pivot table of movie titles & user id and imputing the NaN values

In [32]:
```
data
```

Out[32]:

	MovieID	Title	Genres	Year	UserID	Rating	Timestamp	Gender	Age	Occupation	Zipcode	releasedERA
0	1	Toy Story	Animation|Children's|Comedy	1995	1	5	978824268	F	Under 18	K-12 student	48067	90s
1	48	Pocahontas	Animation|Children's|Musical|Romance	1995	1	5	978824351	F	Under 18	K-12 student	48067	90s
2	150	Apollo 13	Drama	1995	1	5	978301777	F	Under 18	K-12 student	48067	90s
3	260	Star Wars: Episode IV - A New Hope	Action|Adventure|Fantas	1977	1	4	978300760	F	Under 18	K-12 student	48067	70s
4	527	Schindler's List	Drama|War	1993	1	5	978824195	F	Under 18	K-12 student	48067	90s
...	...	...	...	...	...	...	...	...	...	...	...	...
996139	3513	Rules of Engagement	Drama|Thriller	2000	5727	4	958489970	M	25-34	college/grad student	92843	90s
996140	3535	American Psycho	Comedy|Horror|Thriller	2000	5727	2	958489970	M	25-34	college/grad student	92843	90s
996141	3536	Keeping the Faith	Comedy|Romance	2000	5727	5	958489902	M	25-34	college/grad student	92843	90s
996142	3555	U-571	Action|Thriller	2000	5727	3	958490699	M	25-34	college/grad student	92843	90s
996143	3578	Gladiator	Action|Drama	2000	5727	5	958490171	M	25-34	college/grad student	92843	90s

996144 rows × 12 columns

In [33]:
```
matrix = pd.pivot_table(data, index = 'UserID' , columns = 'Title', values = 'Rating' , aggfunc= 'mean')
```

In [34]:
```
matrix.fillna(0, inplace = True)
matrix.head(10)
```

Out[34]:

Title	$1,000,000 Duck	'Night Mother	'Til There Was You	'burbs, The	...And Justice for All	1-900	10 Things I Hate About You	101 Dalmatians	12 Angry Men	13th Warrior, The	...	Young Poisoner's Handbook, The	Young Sherlock Holmes	Young and Innocent	Your Friends and Neighbors	Zachariah	Zed & Two Noughts, A	Zero Effect	Zero Kelvin (Kjærlighetens kjøtere)	Zeus and Roxanne	eXistenZ
UserID																					
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	4.0	0.0	0.0	0.0	0.0	3.0	4.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
100	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1001	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3.0	0.0	0.0	...	0.0	0.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0	5.0
1002	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1003	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1004	0.0	0.0	0.0	0.0	0.0	0.0	0.0	22.0	0.0	0.0	...	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1005	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1006	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3.0

10 rows × 3640 columns

In [35]:
```
matrix.shape
```

Out[35]: (6040, 3640)

3.2 Pearson Correlation

Take a movie name as input from the user

Recommend 5 similar movies based on Pearson Correlation

EXPLANATION :

Correlation is a measure that tells how closely two variables move in the same or opposite direction. A positive value indicates that they move in the same direction (i.e. if one increases other increases), where as a negative value indicates the opposite.

The most popular correlation measure for numerical data is Pearson’s Correlation. This measures the degree of linear relationship between two numeric variables and lies between -1 to +1. It is represented by ‘r’.

r=1 means perfect positive correlation r=-1 means perfect negative correlation r=0 means no linear correlation (note, it does not mean no correlation)```

3.2.1 Item - Based approach

We will take a movie name as an input from the user and see which other 5 (five) movies have maximum correlation with it.

In [36]:

```
data[data['Title']=='Toy Story']
```

Out[36]:

	MovieID	Title	Genres	Year	UserID	Rating	Timestamp	Gender	Age	Occupation	Zipcode	releasedERA
0	1	Toy Story	Animation|Children's|Comedy	1995	1	5	978824268	F	Under 18	K-12 student	48067	90s
53	1	Toy Story	Animation|Children's|Comedy	1995	6	4	978237008	F	50-55	homemaker	55117	90s
123	1	Toy Story	Animation|Children's|Comedy	1995	8	4	978233496	M	25-34	programmer	11413	90s
262	1	Toy Story	Animation|Children's|Comedy	1995	9	5	978225952	M	25-34	technician/engineer	61614	90s
368	1	Toy Story	Animation|Children's|Comedy	1995	10	5	978226474	F	35-44	academic/educator	95370	90s
...	...	...	...	...	...	...	...	...	...	...	...	...
573061	1	Toy Story	Animation|Children's|Comedy	1995	6022	5	956755763	M	25-34	technician/engineer	57006	90s
573109	1	Toy Story	Animation|Children's|Comedy	1995	6025	5	956812867	F	25-34	academic/educator	32607	90s
573379	1	Toy Story	Animation|Children's|Comedy	1995	6032	4	956718127	M	45-49	executive/managerial	55108	90s
573483	1	Toy Story	Animation|Children's|Comedy	1995	6035	4	956712849	F	25-34	academic/educator	78734	90s
573763	1	Toy Story	Animation|Children's|Comedy	1995	6040	3	957717358	M	25-34	doctor/health care	11106	90s

2077 rows × 12 columns

In [37]:

```
movie_name='Toy Story'
movie_rating = matrix[movie_name] # Taking the ratings of that movie
print(movie_rating)
```

```
UserID
1       5.0
10      5.0
100     0.0
1000    5.0
1001    4.0
       ...
995     0.0
996     4.0
997     4.0
998     0.0
999     0.0
Name: Toy Story, Length: 6040, dtype: float64
```

In [38]:
```
similar_movies = matrix.corrwith(movie_rating) #Finding similar movies

sim_df = pd.DataFrame(similar_movies, columns=['Correlation'])
sim_df.sort_values('Correlation', ascending=False, inplace=True) # Sorting the values based on correlation

sim_df.iloc[1: , :].head() #Top 5 correlated movies.
```

Out[38]:

	Correlation
Title	
Toy Story 2	0.487370
Aladdin	0.470753
Lion King, The	0.411131
Groundhog Day	0.407547
Bug's Life, A	0.402679

4. Build a Recommender System based on Cosine Similarity.

- Print the user similarity matrix and item similarity matrix
- Use the Item-based approach to create a recommender system that uses Nearest Neighbors algorithm and Cosine Similarity

Cosine similarity is a measure of similarity between two sequences of numbers. Those sequences are viewed as vectors in a higher dimensional space, and the cosine similarity is defined as the cosine of the angle between them, i.e. the dot product of the vectors divided by the product of their lengths.

The cosine similarity always belongs to the interval [-1,1]. For example, two proportional vectors have a cosine similarity of 1, two orthogonal vectors have a similarity of 0, and two opposite vectors have a similarity of -1.

In [39]:
```
item_sim = cosine_similarity(matrix.T) #Finding the similarity values between item-item using cosine_similarity
item_sim
```

Out[39]:
```
array([[1.        , 0.07235746, 0.03701053, ..., 0.        , 0.12024178,
        0.02700277],
       [0.07235746, 1.        , 0.11528952, ..., 0.        , 0.        ,
        0.07780705],
       [0.03701053, 0.11528952, 1.        , ..., 0.        , 0.04752635,
        0.0632837 ],
       ...,
       [0.        , 0.        , 0.        , ..., 1.        , 0.        ,
        0.04564448],
       [0.12024178, 0.        , 0.04752635, ..., 0.        , 1.        ,
        0.04433508],
       [0.02700277, 0.07780705, 0.0632837 , ..., 0.04564448, 0.04433508,
        1.        ]])
```

4.1 Item-Based Similarity

In [40]:
```
item_sim_matrix = pd.DataFrame(item_sim, index=matrix.columns, columns=matrix.columns)
item_sim_matrix.head() #Item-similarity Matrix
```

Out[40]:

Title	$1,000,000 Duck	'Night Mother	'Til There Was You	'burbs, The	...And Justice for All	1-900	10 Things I Hate About You	101 Dalmatians	12 Angry Men	13th Warrior, The	...	Young Poisoner's Handbook, The	Young Sherlock Holmes	Young and Innocent	Your Friends and Neighbors	Zachariah	Zed & Two Noughts, A	Zero Effect	Zero Kelvin (Kjærlighetens kjøtere)	Zeus and Roxanne	eXistenZ
Title																					
$1,000,000 Duck	1.000000	0.072357	0.037011	0.079291	0.060838	0.00000	0.058619	0.217550	0.094785	0.058418	...	0.038725	0.076474	0.000000	0.044074	0.0	0.045280	0.039395	0.000000	0.120242	0.027003
'Night Mother	0.072357	1.000000	0.115290	0.115545	0.159526	0.00000	0.076798	0.138239	0.111413	0.046135	...	0.053010	0.087828	0.063758	0.135962	0.0	0.091150	0.074787	0.000000	0.000000	0.077807
'Til There Was You	0.037011	0.115290	1.000000	0.098756	0.066301	0.08025	0.127895	0.135076	0.079115	0.066598	...	0.029200	0.062893	0.000000	0.079187	0.0	0.022594	0.079261	0.000000	0.047526	0.063284
'burbs, The	0.079291	0.115545	0.098756	1.000000	0.143620	0.00000	0.192191	0.225182	0.170719	0.197808	...	0.113386	0.207897	0.019962	0.138064	0.0	0.055704	0.161174	0.000000	0.033567	0.110525
...And Justice for All	0.060838	0.159526	0.066301	0.143620	1.000000	0.00000	0.075093	0.178003	0.205486	0.122431	...	0.089998	0.153006	0.067009	0.109029	0.0	0.086080	0.110867	0.074317	0.000000	0.111040

5 rows × 3640 columns

4.2 User-Based Similarity

In [41]:
```
user_sim = cosine_similarity(matrix) #Finding the similarity values between user-user using cosine_similarity
user_sim
```

Out[41]:
```
array([[1.        , 0.25449626, 0.12396703, ..., 0.15926709, 0.11935626,
        0.15362375],
       [0.25449626, 1.        , 0.23920712, ..., 0.15265679, 0.12283209,
        0.23094243],
       [0.12396703, 0.23920712, 1.        , ..., 0.20430203, 0.11352239,
        0.28505089],
       ...,
       [0.15926709, 0.15265679, 0.20430203, ..., 1.        , 0.18657496,
        0.2286199 ],
       [0.11935626, 0.12283209, 0.11352239, ..., 0.18657496, 1.        ,
        0.10055099],
       [0.15362375, 0.23094243, 0.28505089, ..., 0.2286199 , 0.10055099,
        1.        ]])
```

In [42]:

```
user_sim_matrix = pd.DataFrame(user_sim, index=matrix.index, columns=matrix.index)
user_sim_matrix.head()
```

Out[42]:

UserID UserID	1	10	100	1000	1001	1002	1003	1004	1005	1006	...	990	991	992	993	994	995	996	997	998	999
1	1.000000	0.254496	0.123967	0.207800	0.137839	0.110320	0.121384	0.159694	0.103896	0.052816	...	0.079367	0.038048	0.032136	0.047557	0.070052	0.035731	0.170184	0.159267	0.119356	0.153624
10	0.254496	1.000000	0.239207	0.258402	0.155321	0.104029	0.130809	0.403120	0.190428	0.094420	...	0.142258	0.204891	0.077148	0.081559	0.109226	0.135016	0.280814	0.152657	0.122832	0.230942
100	0.123967	0.239207	1.000000	0.306067	0.074933	0.110450	0.358686	0.210437	0.172872	0.099147	...	0.098235	0.097953	0.065152	0.125634	0.271311	0.033754	0.344290	0.204302	0.113522	0.285051
1000	0.207800	0.258402	0.306067	1.000000	0.098066	0.047677	0.201722	0.338103	0.325966	0.130702	...	0.170100	0.076779	0.000000	0.140878	0.380741	0.044404	0.330748	0.172803	0.098456	0.232698
1001	0.137839	0.155321	0.074933	0.098066	1.000000	0.163105	0.053315	0.140485	0.137132	0.133281	...	0.144718	0.026606	0.095982	0.083215	0.091256	0.108536	0.219762	0.102160	0.267088	0.180497

5 rows × 6040 columns

4.2.1 Nearest Neighbors

In [43]:

```
model_knn = NearestNeighbors(metric='cosine')
model_knn.fit(matrix.T)
```

Out[43]:

```
NearestNeighbors
NearestNeighbors(metric='cosine')
```

In [44]:

```
##The distances and indices are being calculated with neighbors being 6
distances, indices = model_knn.kneighbors(matrix.T, n_neighbors= 6)
```

In [45]:

```
result = pd.DataFrame(indices, columns=['Title1', 'Title2', 'Title3', 'Title4', 'Title5','Title6'])
result.head()
#The result dataframe consits of the different indices of movies based on the distance
```

Out[45]:

	Title1	Title2	Title3	Title4	Title5	Title6
0	0	735	416	286	3247	584
1	1	807	72	2167	3036	3369
2	2	1627	2529	3320	2588	1999
3	3	1457	2169	1308	1047	3511
4	4	26	726	894	495	944

In [46]:
```
##With this for loop replacing the indices in the result dataframe with movie titles of that corresponding ones
result2 = result.copy()
for i in range(1, 7):
    mov = pd.DataFrame(matrix.T.index).reset_index()
    mov = mov.rename(columns={'index':f'Title{i}'})
    result2 = pd.merge(result2, mov, on=[f'Title{i}'], how='left')
    result2 = result2.drop(f'Title{i}', axis=1)
    result2 = result2.rename(columns={'Title':f'Title{i}'})
result2.head()
```

Out[46]:

	Title1	Title2	Title3	Title4	Title5	Title6
0	$1,000,000 Duck	Computer Wore Tennis Shoes, The	Blackbeard's Ghost	Barefoot Executive, The	That Darn Cat!	Candleshoe
1	'Night Mother	Cry in the Dark, A	Agnes of God	Mommie Dearest	Sophie's Choice	Trip to Bountiful, The
2	'Til There Was You	If Lucy Fell	Picture Perfect	To Gillian on Her 37th Birthday	Practical Magic	Mad Love
3	'burbs, The	Harry and the Hendersons	Money Pit, The	Ghostbusters II	European Vacation	Weekend at Bernie's
4	...And Justice for All	52 Pick-Up	Coma	Deliverance	Boys from Brazil, The	Dog Day Afternoon

In [47]:
```
#movie_name = input("Enter a movie name: ")
movie_name = 'Liar Liar'
result2.loc[result2['Title1']==movie_name] #5 nearest movies for the movie present in Title1.
```

Out[47]:

	Title1	Title2	Title3	Title4	Title5	Title6
1887	Liar Liar	Mrs. Doubtfire	Ace Ventura: Pet Detective	Dumb & Dumber	Home Alone	Wayne's World

5. Build a Recommender System based on Matrix Factorization.

- Create a Recommender System using the Matrix Factorization method
- Evaluate the model in terms of the Root Mean Squared Error and Mean Absolute Percentage Error
- Use embeddings for visualization and similarity-based models.

5.1 Matrix Factorization

Creating a pivot table of movie titles and userid and ratings are taken as values.

In [48]:
```
rm = data.pivot(index = 'UserID', columns ='MovieID', values = 'Rating').fillna(0)
rm.head()
```

Out[48]:

MovieID	1	10	100	1000	1002	1003	1004	1005	1006	1007	...	99	990	991	992	993	994	996	997	998	999
UserID																					
1	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
10	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
100	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1000	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1001	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 3682 columns

5.2 Using Surprise Library

In [49]:
```
from surprise import Reader, SVD, Dataset
from surprise.model_selection import cross_validate
```

In [50]:
```
data.Rating.value_counts()
```

Out[50]:
```
4    347758
3    260473
5    224639
2    107261
1     56013
Name: Rating, dtype: int64
```

In [51]:
```
user_itm = data[['UserID', 'Title', 'Rating']].copy()
reader = Reader(rating_scale=(1,5))
data1 = Dataset.load_from_df(user_itm[['UserID', 'Title', 'Rating']], reader)
```

In [52]:
```
print(user_itm.shape)
print("No.of Users:",len(user_itm['UserID'].unique()))
print("No.of Items:",len(user_itm['Title'].unique()))
```

```
(996144, 3)
No.of Users: 6040
No.of Items: 3640
```

The dataset is divided into train and test and with 3 folds the rmse has been calculated

In [53]:
```
svd = SVD()
cross_validate(svd, data1, measures=['rmse'], cv=3, return_train_measures=True)
```

Out[53]:
```
{'test_rmse': array([0.88789339, 0.88443197, 0.8867463 ]),
 'train_rmse': array([0.6727083 , 0.67013802, 0.67063559]),
 'fit_time': (13.795299053192139, 13.035279512405396, 12.504131555557251),
 'test_time': (5.711869716644287, 5.795715808868408, 5.359710693359375)}
```

In [54]:
```
trainset = data1.build_full_trainset()
svd.fit(trainset)
```

Out[54]:
```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x20db72954c0>
```

In [55]:
```
#Storing all the movie titles in items
items = movies['Title'].unique()
##Considering the user '662'
test = [[662, iid, 4] for iid in items]
##Finding the user predictions(ratings) for all the movies
predictions = svd.test(test)
pred = pd.DataFrame(predictions)
```

In [56]:
```
a = pred.sort_values(by='est', ascending=False) ##Sorting the values based on the estimated predictions
```

In [57]:
```
a[0:10] ##TOP 10
```

Out[57]:

	uid	iid	r_ui	est	details
2789	662	Sanjuro	4	4.693435	{'was_impossible': False}
313	662	Shawshank Redemption, The	4	4.522334	{'was_impossible': False}
49	662	Usual Suspects, The	4	4.509663	{'was_impossible': False}
1122	662	Wrong Trousers, The	4	4.493728	{'was_impossible': False}
884	662	Rear Window	4	4.482147	{'was_impossible': False}
730	662	Close Shave, A	4	4.466262	{'was_impossible': False}
1177	662	To Kill a Mockingbird	4	4.466140	{'was_impossible': False}
519	662	Schindler's List	4	4.464739	{'was_impossible': False}
3216	662	For All Mankind	4	4.456773	{'was_impossible': False}
839	662	Godfather, The	4	4.450662	{'was_impossible': False}

In [58]:
```
testset = trainset.build_anti_testset()

predictions_svd = svd.test(testset)
```

5.2 Evaluate the model in terms of the Root Mean Squared Error and Mean Absolute Percentage Error

In [59]:
```
from surprise import accuracy
print('SVD - RMSE:', accuracy.rmse(predictions_svd, verbose=False))
print('SVD - MAE:', accuracy.mae(predictions_svd, verbose=False))
```

```
SVD - RMSE: 0.6994191159969535
SVD - MAE: 0.5419253734716936
```

5.3 Use embeddings for visualization and similarity-based models.

Embeddings for user-user similarity using surprise library.

In [60]:
```
user=cosine_similarity(svd.pu)

user_sim_matrix = pd.DataFrame(user, index=matrix.index, columns=matrix.index)
user_sim_matrix.head()  #User similarity matrix using the embeddings from matrix factorization
```

Out[60]:

UserID	1	10	100	1000	1001	1002	1003	1004	1005	1006	...	990	991	992	993	994	995	996	997	998	999
UserID																					
1	1.000000	0.290507	-0.099379	-0.019971	0.077568	-0.029885	-0.084126	-0.360547	-0.025882	0.141122	...	-0.134906	0.134005	-0.024641	-0.074470	-0.061368	-0.053806	-0.001545	-0.096991	-0.087138	-0.015941
10	0.290507	1.000000	-0.283279	-0.328441	0.192936	-0.112861	-0.026382	0.055132	-0.005282	0.093622	...	0.052235	0.313318	-0.128025	-0.097947	-0.127320	-0.146098	0.051454	-0.148813	0.189078	0.018334
100	-0.099379	-0.283279	1.000000	0.250130	-0.099875	0.190514	0.070234	-0.092592	0.061800	-0.125390	...	-0.040224	-0.051556	0.087697	0.105919	-0.033403	0.108971	-0.011668	-0.071514	0.132569	0.047270
1000	-0.019971	-0.328441	0.250130	1.000000	-0.036093	0.189465	0.173088	0.051528	-0.222573	-0.061808	...	-0.051078	0.008570	0.112663	-0.134072	0.015789	-0.165443	-0.111114	-0.011247	0.015153	0.085729
1001	0.077568	0.192936	-0.099875	-0.036093	1.000000	0.050728	0.073371	-0.181408	0.203388	0.054850	...	0.158906	0.147977	-0.062121	-0.106678	0.153794	-0.043252	0.314728	-0.227274	-0.011133	0.015996

5 rows × 6040 columns

Embeddings for item-item similarity using surprise library.

In [61]:
```
itm=cosine_similarity(svd.qi)

itm_sim_matrix = pd.DataFrame(itm, index=user_itm['Title'].unique(), columns=user_itm['Title'].unique())
itm_sim_matrix.head()#Item similarity matrix using the embeddings from matrix factorization
```

Out[61]:

	Toy Story	Pocahontas	Apollo 13	Star Wars: Episode IV - A New Hope	Schindler's List	Secret Garden, The	Aladdin	Snow White and the Seven Dwarfs	Beauty and the Beast	Fargo	...	Aiqing wansui	Dry Cleaning (Nettoyage à sec)	Lured	Ulysses (Ulisse)	Schlafes Bruder (Brother of Sleep)	Baby, The	Roula	Voyage to the Beginning of the World	Project Moon Base	Heaven's Burning
Toy Story	1.000000	0.214417	0.325640	0.280488	0.225234	0.229668	0.541580	0.288923	0.517204	-0.020903	...	-0.088741	-0.051816	0.141133	-0.088073	0.002842	-0.094793	0.050225	-0.066913	-0.079111	0.267166
Pocahontas	0.214417	1.000000	0.247989	-0.030755	0.013895	0.066583	0.191044	0.180144	0.205039	-0.095515	...	0.179810	0.011872	-0.156984	-0.041906	-0.146199	-0.142120	-0.003985	-0.081682	-0.077422	0.040982
Apollo 13	0.325640	0.247989	1.000000	0.154322	0.324950	0.140670	0.211839	0.185880	0.213917	-0.039480	...	0.043635	0.066468	0.048814	-0.079112	-0.005858	-0.175392	-0.001218	-0.017238	-0.042729	-0.027983
Star Wars: Episode IV - A New Hope	0.280488	-0.030755	0.154322	1.000000	0.185041	0.009078	0.304450	0.122005	0.040982	0.071204	...	0.140080	-0.116623	0.143318	-0.223099	0.090629	-0.088763	-0.028485	0.059960	-0.123078	0.175458
Schindler's List	0.225234	0.013895	0.324950	0.185041	1.000000	0.044077	0.161584	0.106203	0.214886	0.192483	...	0.001414	-0.024837	0.052976	-0.085995	0.018597	0.021956	-0.028822	-0.090104	0.067254	0.090025

5 rows × 3640 columns

In [62]:
```
movie_name='Home Alone'
movie_rating = itm_sim_matrix[movie_name] # Taking the ratings of that movie
print(movie_rating)
```

```
Toy Story                              0.163836
Pocahontas                             0.247086
Apollo 13                              0.280101
Star Wars: Episode IV - A New Hope     0.157616
Schindler's List                       0.072954
                                         ...
Baby, The                              0.016923
Roula                                  0.010945
Voyage to the Beginning of the World  -0.081100
Project Moon Base                     -0.225241
Heaven's Burning                       0.050224
Name: Home Alone, Length: 3640, dtype: float64
```

In [63]:
```python
similar_movies = itm_sim_matrix.corrwith(movie_rating) #Finding similar movies

sim_df = pd.DataFrame(similar_movies, columns=['Correlation'])
sim_df.sort_values('Correlation', ascending=False, inplace=True) # Sorting the values based on correlation

sim_df.iloc[1: , :].head() #Top 5 correlated movies.
```

Out[63]:

	Correlation
Home Alone 2: Lost in New York	0.708116
Mrs. Doubtfire	0.674725
Father of the Bride Part II	0.672703
Santa Clause, The	0.662581
Crocodile Dundee	0.647412

In [ ]:
```python

```

6. Build a Recommender System based Pearson Correlation. (Optional)

- Use the User-based approach to create a recommender system that uses Pearson Correlation

In [64]:
```python
from sklearn.preprocessing import StandardScaler
```

In [65]:
```python
movies.head()
```

Out[65]:

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

In [66]:
```python
users1.head()
```

Out[66]:

	UserID	Gender	Age	Occupation	Zipcode
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 [67]:
```
rating1.head()
```

Out[67]:

	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 [ ]:
```

```

In [68]:
```
genres_df = pd.get_dummies(dfmov['Genres'].apply(pd.Series).stack()).sum(level=0)
genres_df.head()
```

```
C:\Users\Acer\AppData\Local\Temp/ipykernel_21484/3063366026.py:1: FutureWarning: Using the level keyword in DataFrame and Series aggregations is deprecated and will be removed in a future version. Use groupby instead. df.sum(level=1) should use df.groupby(level=1).sum().
  genres_df = pd.get_dummies(dfmov['Genres'].apply(pd.Series).stack()).sum(level=0)
```

Out[68]:

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

In [69]:
```
m = pd.concat([movies1['MovieID'],genres_df.iloc[:,1:]],axis=1)
m.head()
```

Out[69]:

	MovieID	Action	Adventure	Animation	Children's	Comedy	Crime	Documentary	Drama	Fantasy	Film-Noir	Horror	Musical	Mystery	Romance	Sci-Fi	Thriller	War	Western
0	1	0.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	2	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	3	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
3	4	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	5	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

In [70]:
```
from datetime import datetime
r = rating1.copy()
r['Timestamp']=r['Timestamp'].astype('int32')
r['Rating']=r['Rating'].astype('int32')
r['hour'] = r['Timestamp'].apply(lambda x: datetime.fromtimestamp(x).hour)
r.head()
```

Out[70]:

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

In [71]:
```
users2 = users1.merge(r.groupby('UserID').Rating.mean().reset_index(), on='UserID')
users2 = users2.merge(r.groupby('UserID').hour.mean().reset_index(), on='UserID')
users2.head(2)
```

Out[71]:

	UserID	Gender	Age	Occupation	Zipcode	Rating	hour
0	1	F	1	10	48067	4.188679	3.792453
1	2	M	56	16	70072	3.713178	2.968992

In [72]:
```
u = users2[['UserID','Age', 'Rating', 'hour']].copy()
u = u.set_index('UserID')
u .columns = ['Age', 'User_avg_rating', 'hour']

scaler = StandardScaler()
u = pd.DataFrame(scaler.fit_transform(u), columns=u.columns, index=u.index)
u.head(2)
```

Out[72]:

	Age	User_avg_rating	hour
UserID			
1	-2.298525	1.131261	-0.909947
2	1.966729	0.024380	-1.037952

In [73]:
```
df_cat = users2[['Gender','Occupation']]
df_cat['Gender']=pd.get_dummies(df_cat['Gender'], columns=['Gender'],drop_first=True)
df_cat = pd.concat([users['UserID'],df_cat],axis=1)
df_cat.head()
```

```
C:\Users\Acer\AppData\Local\Temp/ipykernel_21484/2292319686.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df_cat['Gender']=pd.get_dummies(df_cat['Gender'], columns=['Gender'],drop_first=True)
```

Out[73]:

	UserID	Gender	Occupation
0	1	0	10
1	2	1	16
2	3	1	15
3	4	1	7
4	5	1	20

In [74]:
```
X = ratings[['MovieID', 'UserID', 'Rating']].copy()
X = X.merge(u.reset_index(), on='UserID', how='right')
X = X.merge(m.reset_index(), on='MovieID', how='right')
X = X.merge(df_cat, on='UserID', how='right')
X.drop(columns=['index'], axis=1, inplace=True)
X.dropna(inplace=True)
X.reset_index(inplace=True,drop=True)
X1=X.copy()
X.head()
```

Out[74]:

	MovieID	UserID	Rating	Age	User_avg_rating	hour	Action	Adventure	Animation	Children's	...	Horror	Musical	Mystery	Romance	Sci-Fi	Thriller	War	Western	Gender	Occupation
0	1	1	5	-2.298525	1.131261	-0.909947	0.0	0.0	1.0	1.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	10
1	48	1	5	-2.298525	1.131261	-0.909947	0.0	0.0	1.0	1.0	...	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0	10
2	150	1	5	-2.298525	1.131261	-0.909947	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	10
3	260	1	4	-2.298525	1.131261	-0.909947	1.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	10
4	527	1	5	-2.298525	1.131261	-0.909947	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0	10

5 rows × 26 columns

In [75]:
```
X = X.drop(columns = ['MovieID', 'UserID'])
y = X.pop('Rating')
```

In [76]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=10)
```

In [77]:
```
from sklearn.ensemble import GradientBoostingRegressor

model = GradientBoostingRegressor()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

In [78]:
```
rmse = mean_squared_error(y_test, y_pred, squared=False) # calculating rmse value
print('Root Mean Squared Error: {:.3f}'.format(rmse))
```

```
Root Mean Squared Error: 1.008
```

In [79]:
```
mape =  mean_absolute_percentage_error(y_test, y_pred) #calculating mape value
print('Mean Absolute Percentage Error: {:.3f}'.format(mape))
```

```
Mean Absolute Percentage Error: 0.325
```

Questionnaire

1. Users of which age group have watched and rated the most number of movies? :- **25-34 age group**
2. Users belonging to which profession have watched and rated the most movies? :- **college/grad student**
3. Most of the users in our dataset who've rated the movies are Male. (T/F):- **True**
4. Most of the movies present on our dataset were released in which decade? :- **b.90s** a.70s b. 90s c. 50s d.80s
5. The movie with maximum no. of ratings is ___ :- **American Beauty**
6. Name the top 3 movies similar to 'Liar Liar' on the item-based approach. :- **Mrs. Doubtfire, Ace Ventura: Pet, Detective Dumb & Dumber**
7. On the basis of approach, Collaborative Filtering methods can be classified into **Memory-based** and **Model-based**.
8. Pearson Correlation ranges between **-1 to 1** whereas, Cosine Similarity belongs to the interval between **-1 to 1**
9. Mention the RMSE and MAPE that you got while evaluating the Matrix Factorization model.:- **RMSE:0.701 and MAPE: 0.54**

10 Give the sparse 'row' matrix representation for the following dense matrix - [[1 0],[3 7]]

In [81]:
```
from scipy.sparse import csr_matrix
# create dense matrix
A = np.array([[1,0],[3,7]])
# convert to sparse matrix (CSR method)
S = csr_matrix(A)
print(S)
```

```
  (0, 0)	1
  (1, 0)	3
  (1, 1)	7
```

In [ ]:
```

```