🎥 🎬 Case Study: ZEE5 - Recommendation System (Unsupervised Learning) 🎬 🎥

Analysed by : **KASI**

Zee Recommender Systems: Personalized Movie Recommendations

Dataset Explanation:

The dataset for this project is composed of three primary files, each contributing essential information for building the recommender system:

1. Ratings File (ratings.dat):

- Format: UserID::MovieID::Rating::Timestamp
- Contains user ratings for movies on a 5-star scale.
- Includes a timestamp representing when the rating was given.
- Each user has rated at least 20 movies.

2. Users File (users.dat):

- Format: UserID::Gender::Age::Occupation::Zip-code
- Provides demographic information about the users, including gender, age group, occupation, and zip code.
- Demographic data is voluntarily provided by users and varies in accuracy and completeness.

3. Movies File (movies.dat):

- Format: MovieID::Title::Genres
- Lists movie titles alongside their respective genres.
- Genres are categorized into multiple types like Action, Comedy, Drama, etc., and are pipe-separated.

Key Points to Note:

1. UserID and MovieID serve as unique identifiers for users and movies, respectively.
2. Ratings reflect user preferences and are crucial for understanding individual and collective tastes.
3. User Demographics (gender, age, occupation) can provide insights into user preferences and behavior patterns.
4. Movie Details (title, genres) are essential for categorizing movies and understanding their appeal to different user segments.

What is Expected?

Assuming you're a data scientist at Zee, your responsibility involves creating a personalized movie recommender system.

Our primary goals are:

- To analyze user ratings, demographic data, and movie characteristics to understand viewing preferences.
- To apply collaborative filtering, Pearson Correlation, Cosine Similarity, and Matrix Factorization techniques to build an effective recommender system.
- To evaluate the system's performance and refine it for accuracy and user relevance.

```
#Importing libraries

import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

from scipy import sparse
from scipy.stats import pearsonr

from sklearn.metrics.pairwise import cosine_similarity
from sklearn.neighbors import NearestNeighbors

import warnings
warnings.filterwarnings('ignore')

from sklearn.metrics import mean_absolute_percentage_error
from sklearn.metrics import mean_squared_error
```

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

```
movies = pd.read_fwf('zee-movies.dat',encoding='ISO-8859-1')
ratings = pd.read_fwf('zee-ratings.dat',encoding='ISO-8859-1')
users = pd.read_fwf('zee-users.dat',encoding='ISO-8859-1')
```

```
print("Shape of the Movies dataset: ", movies.shape)
print("Shape of the Ratings dataset: ", ratings.shape)
print("Shape of the Users dataset: ", users.shape)
```

```
Shape of the Movies dataset:  (3883, 3)
Shape of the Ratings dataset:  (1000209, 1)
Shape of the Users dataset:  (6040, 1)
```

```
movies.head()
```

	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

```
ratings.head()
```

	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

```
users.head()
```

	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

```
#Read Data and Data Formatting:
#MOVIES:

movies.drop(columns=['Unnamed: 1', 'Unnamed: 2'], inplace=True)

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

movies.rename(columns={'Movie ID':'MovieID'}, inplace=True)
movies1=movies.copy()
movies.head()
```

	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

```
#Read Data and Data Formatting:
#RATINGS:

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

ratings1=ratings.copy()
ratings.head()
```

	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

```
#Read Data and Data Formatting:
#USERS:

users = users['UserID::Gender::Age::Occupation::Zip-code'].str.split(delimiter, expand=True)
users.columns = ['UserID', 'Gender', 'Age', 'Occupation', 'ZipCode']
users1=users.copy()
users.head()
```

	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

```
users.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6040 entries, 0 to 6039
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   UserID      6040 non-null   object
 1   Gender      6040 non-null   object
 2   Age         6040 non-null   object
 3   Occupation  6040 non-null   object
 4   ZipCode     6040 non-null   object
dtypes: object(5)
memory usage: 236.1+ KB
```

```
users.describe()
```

	UserID	Gender	Age	Occupation	ZipCode
count	6040	6040	6040	6040	6040
unique	6040	2	7	21	3439
top	6040	M	25	4	48104
freq	1	4331	2096	759	19

```
users['Age'].value_counts()
```

Age	count
25	2096
35	1193
18	1103
45	550
50	496
56	380
1	222

dtype: int64

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

```
users['Occupation'].value_counts()
```

Occupation	count
4	759
0	711
7	679
1	528
17	502
12	388
14	302
20	281
2	267
16	241
6	236
10	195
3	173
15	144
13	142
11	129
5	112
9	92
19	72
18	70
8	17

dtype: int64

```
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)
```

```
users.head()
```

	UserID	Gender	Age	Occupation	ZipCode
0	1	F	Under 18	k-12 student	48067
1	2	M	56 and above	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

Exploratory Data Analysis

```
# Extracting the year between parentheses
movies['Year'] = movies['Title'].str.extract(r'\((\d{4})\)', expand=False)

# Removing the year from the 'Title' column, accounting for potential spaces before or after the parenthese
movies['Title'] = movies['Title'].str.replace(r'\s?\(\d{4}\)', '', regex=True)

# Applying strip to remove any unwanted leading/trailing spaces that may appear after removal
movies['Title'] = movies['Title'].str.strip()

# Display the updated DataFrame
movies.head()
```

	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

```
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'
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] =='Chil
            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()
```

	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

```
df_1 = pd.merge(dfmov, ratings, how='inner', on='MovieID')
df_1.head()
```

	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

```
data = pd.merge(df_1, users, how='inner', on='UserID')
data.head()
```

	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	1	Toy Story	[Animation, Children's, Comedy]	1995	6	4	978237008	F	50-55	homemaker	55117
2	1	Toy Story	[Animation, Children's, Comedy]	1995	8	4	978233496	M	25-34	programmer	11413

```
#Shape of final dataset:
print("Number of rows: ", data.shape[0])
print("Number of columns: ", data.shape[1])
```

```
Number of rows:  996144
Number of columns:  11
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 996144 entries, 0 to 996143
Data columns (total 11 columns):
 #   Column      Non-Null Count   Dtype 
---  ------      --------------   ----- 
 0   MovieID     996144 non-null  object
 1   Title       996144 non-null  object
 2   Genres      996144 non-null  object
 3   Year        996144 non-null  object
 4   UserID      996144 non-null  object
 5   Rating      996144 non-null  object
 6   Timestamp   996144 non-null  object
 7   Gender      996144 non-null  object
 8   Age         996144 non-null  object
 9   Occupation  996144 non-null  object
 10  ZipCode     996144 non-null  object
dtypes: object(11)
memory usage: 83.6+ MB
```

```
data.describe()
```

	MovieID	Title	Genres	Year	UserID	Rating	Timestamp	Gender	Age	Occupation	ZipCode
count	996144	996144	996144	996144	996144	996144	996144	996144	996144	996144	996144
unique	3682	3640	300	81	6040	5	457632	2	7	21	3439
top	2858	American Beauty	[Comedy]	1999	4169	4	975528402	M	25-34	college/grad student	94110
freq	3428	3428	117248	86833	2303	347758	30	750590	394105	130626	3782

```
missing_values = pd.DataFrame({
    'Missing Value': data.isnull().sum(),
    'Percentage': (data.isnull().sum() / len(data))*100
})
```

```
missing_values.sort_values(by='Percentage', ascending=False)
```

	Missing Value	Percentage
MovieID	0	0.00
Title	0	0.00
Genres	0	0.00
Year	0	0.00
UserID	0	0.00
Rating	0	0.00
Timestamp	0	0.00
Gender	0	0.00
Age	0	0.00
Occupation	0	0.00
ZipCode	0	0.00

```
#Feature Engineering

data['Datetime'] = pd.to_datetime(data['Timestamp'], unit='s') #Change the datatype from object to date_tim
data['Year']=data['Year'].astype('int32') #Change the datatype from object to Integer
data['Rating']=data['Rating'].astype('int32') #Change the datatype from object to Integer
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 996144 entries, 0 to 996143
Data columns (total 12 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   MovieID     996144 non-null  object
 1   Title       996144 non-null  object
 2   Genres      996144 non-null  object
 3   Year        996144 non-null  int32
 4   UserID      996144 non-null  object
 5   Rating      996144 non-null  int32
 6   Timestamp   996144 non-null  object
 7   Gender      996144 non-null  object
 8   Age         996144 non-null  object
 9   Occupation  996144 non-null  object
 10  ZipCode     996144 non-null  object
 11  Datetime    996144 non-null  datetime64[ns]
dtypes: datetime64[ns](1), int32(2), object(9)
memory usage: 83.6+ MB
```

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

```
data.head()
```

	MovieID	Title	Genres	Year	UserID	Rating	Timestamp	Gender	Age	Occupation	ZipCode	Datetime	ReleaseDec
0	1	Toy Story	[Animation, Children's, Comedy]	1995	1	5	978824268	F	Under 18	k-12 student	48067	2001-01-06 23:37:48	90s
1	1	Toy Story	[Animation, Children's, Comedy]	1995	6	4	978237008	F	50-55	homemaker	55117	2000-12-31 04:30:08	90s
		Toy	[Animation,									2000-12-	

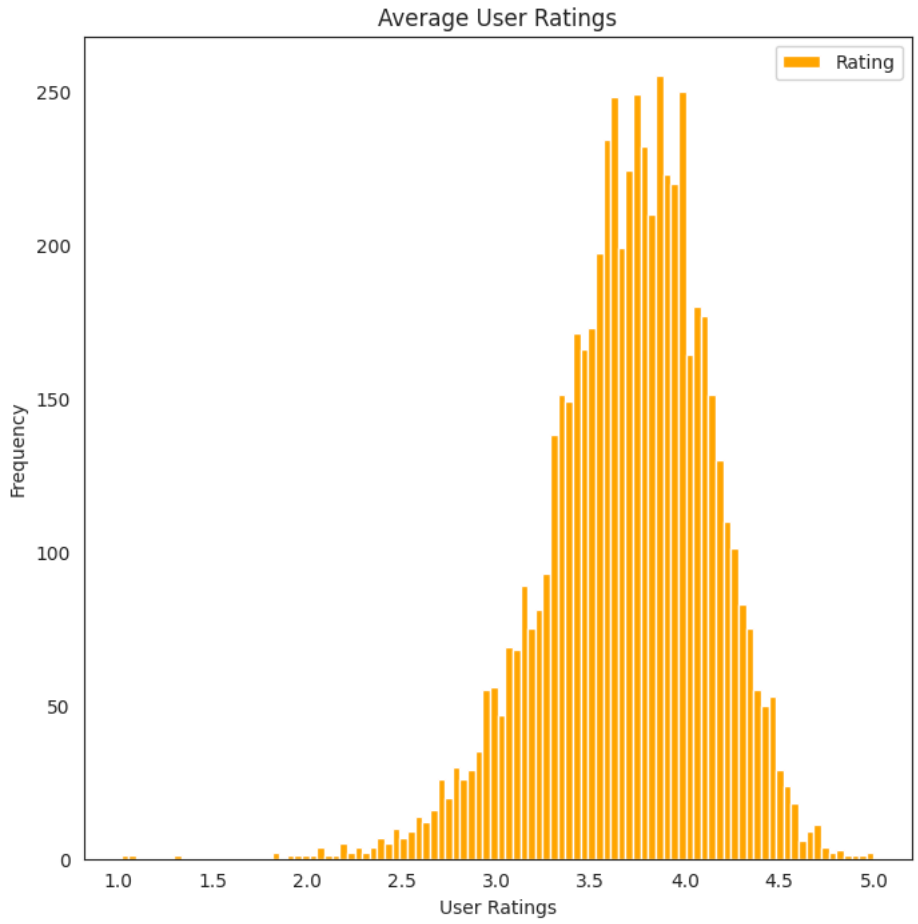
Univariate, Bi-variate and Multivariate Analysis

```
#Average Users Ratings:

user_ratings =data[['UserID','Rating']].groupby('UserID').mean()
```

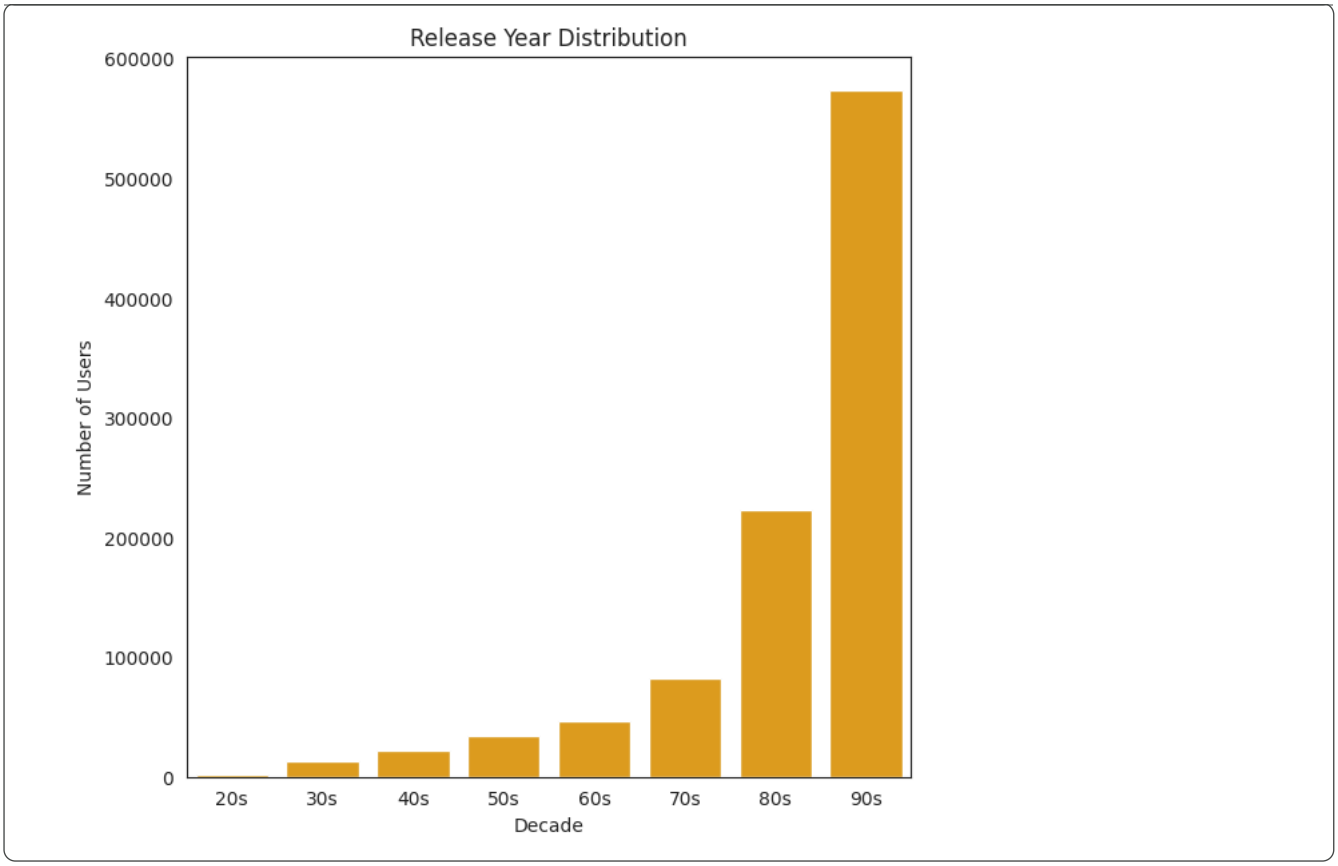
```
fig = plt.figure(figsize = (8,8))
user_ratings.plot(kind = 'hist', bins = 100, figsize = (8,8), color = 'orange')
plt.plot()
plt.xlabel('User Ratings')
```

```
plt.title('Average User Ratings')
plt.ylabel('Frequency')
```

```
Text(0, 0.5, 'Frequency')
<Figure size 800x800 with 0 Axes>
```




```
#Number of movies by Release Year:

plt.figure(figsize=(7, 7))
sns.countplot(x='ReleaseDec', data=data, color='orange')
plt.title('Release Year Distribution')
plt.xlabel('Decade')
plt.ylabel('Number of Users')
plt.show()
```



Insights - Most movies were released in the 90's

```
#### Top Genres based on movies count

genres_df = pd.get_dummies(dfmov['Genres'].apply(pd.Series).stack()).sum(axis=0)
genres_df
```

	0
	8
Action	501
Adventure	282
Animation	104
Children's	249
Comedy	1189
Crime	210
Documentary	124
Drama	1582
Fantasy	62
Film-Noir	44
Horror	340
Musical	113
Mystery	105
Romance	462
Sci-Fi	265
Thriller	488
War	139
Western	68

dtype: int64

```
### considering only the genre columns for the test
test = genres_df.iloc[1:]
```

```
test
```

	0
Action	501
Adventure	282
Animation	104
Children's	249
Comedy	1189
Crime	210
Documentary	124
Drama	1582
Fantasy	62
Film-Noir	44
Horror	340
Musical	113
Mystery	105
Romance	462
Sci-Fi	265
Thriller	488
War	139
Western	68

dtype: int64

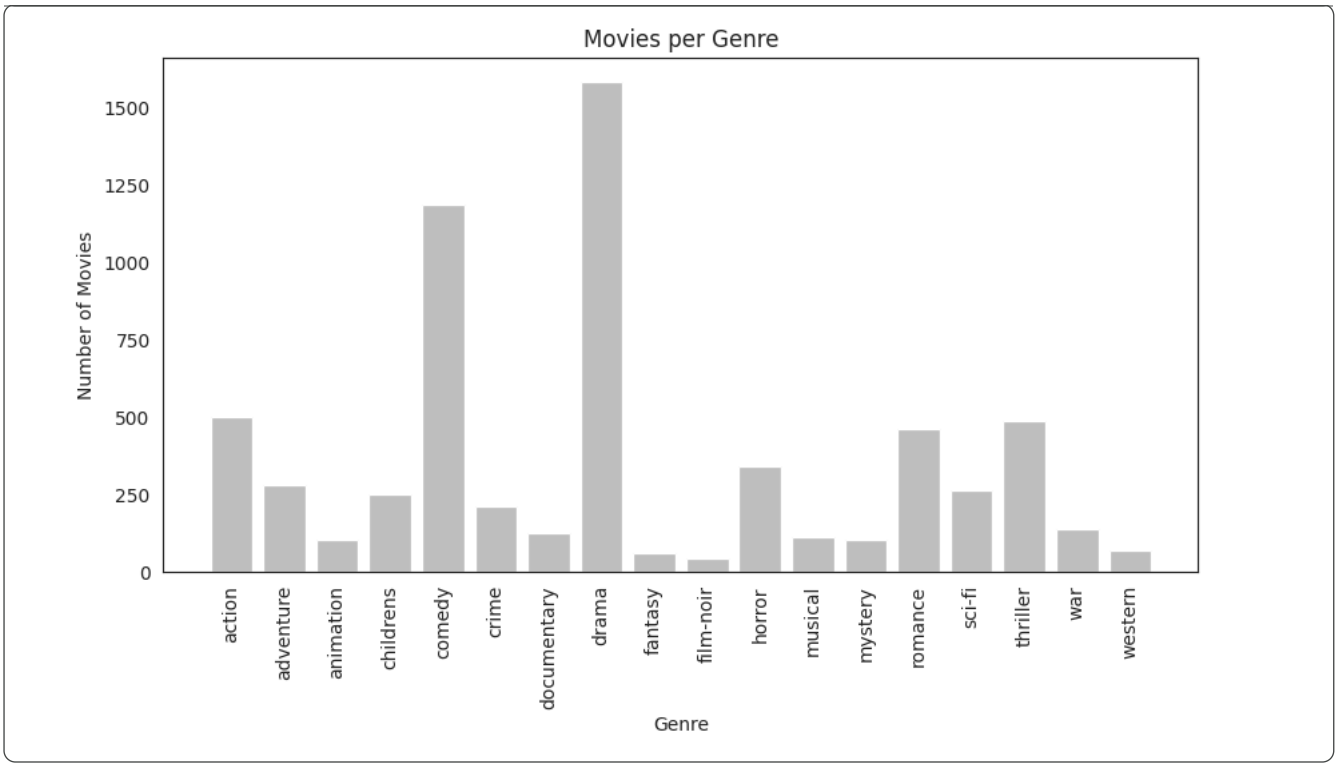
```
len(test)
```

```
18
```

```
print(type(pd.to_numeric(test)))
print(type(test.to_numpy().reshape(18,)[0]))
test2 = test.to_numpy().reshape(18,)
```

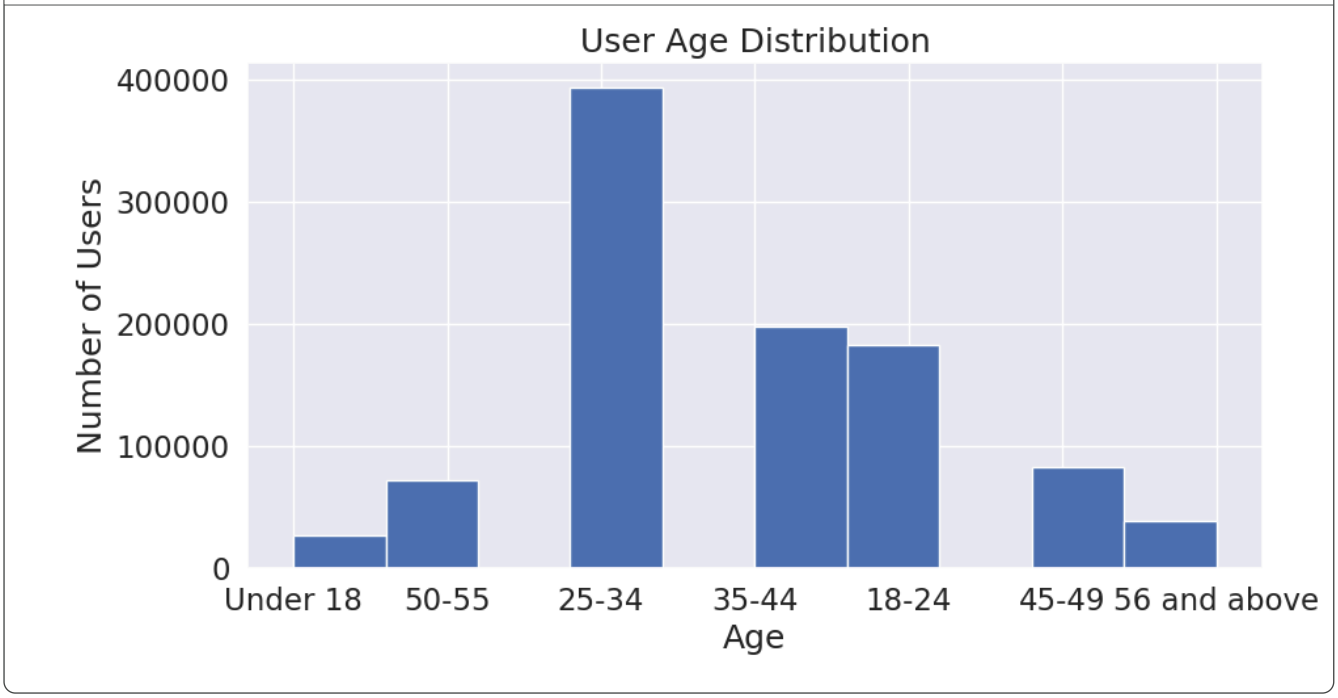
```
<class 'pandas.core.series.Series'>
<class 'numpy.int64'>
```

```
genre_list=['action', 'adventure','animation', 'childrens', 'comedy', 'crime', 'documentary', 'drama','fant
x = np.arange(18)
plt.figure(figsize = (10,5))
plt.bar(x, test2, color = 'silver')
plt.xticks(x, genre_list, rotation = 'vertical')
plt.xlabel('Genre')
plt.ylabel('Number of Movies')
plt.title('Movies per Genre')
sns.set(font_scale=1.5)
plt.show()
```



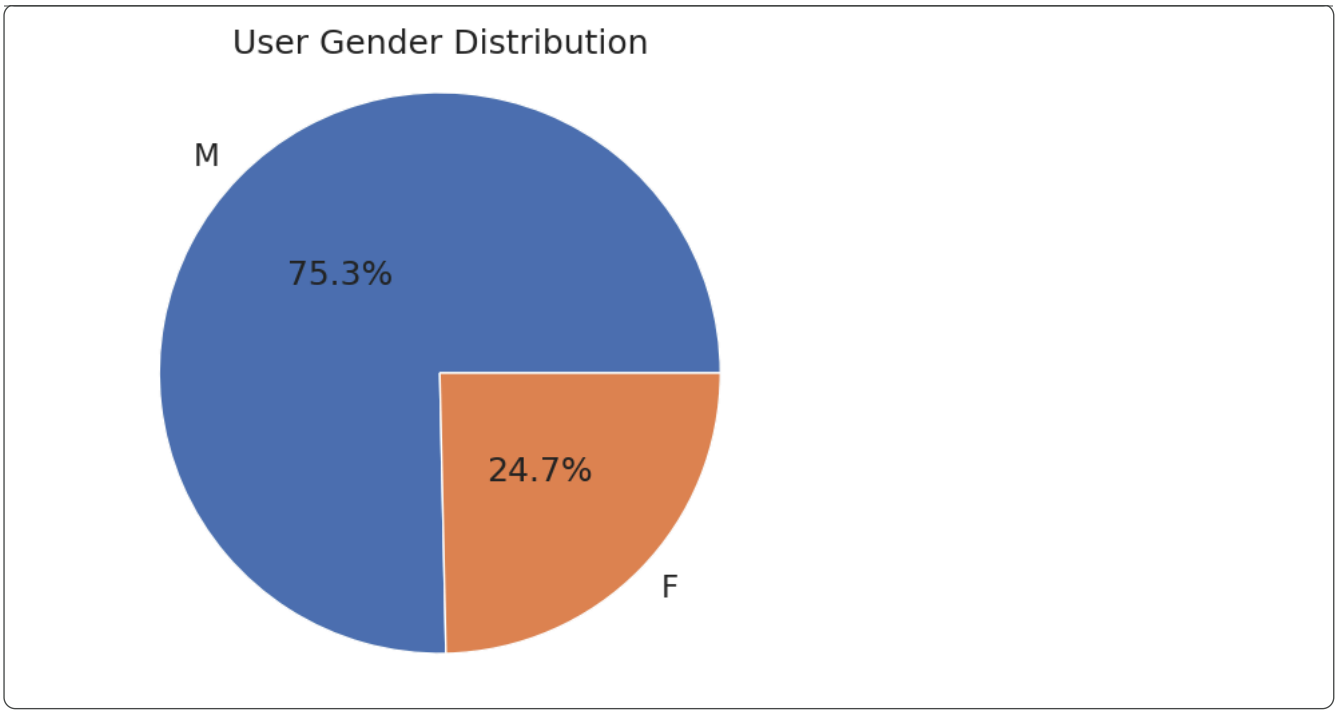
Comedy and drama are the most popular genres amongst movies watched and rated by users.

```
#Distribution of Age

data['Age'].hist(figsize=(10, 5))
plt.title('User Age Distribution')
plt.xlabel('Age')
plt.ylabel('Number of Users')
plt.show()
```



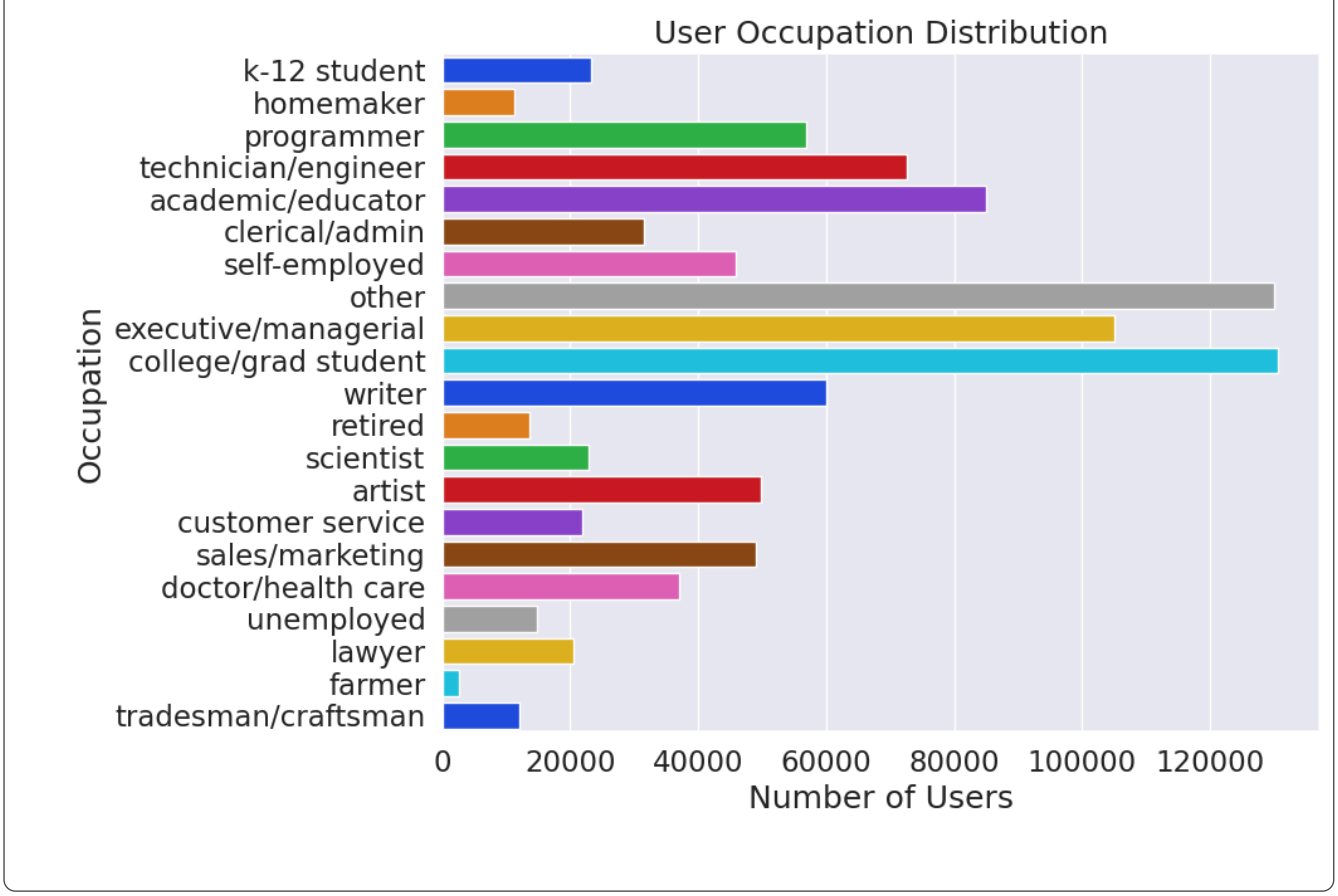
25-34 Age group people have rated the movies to the maximum.

```
#Distribution by Gender:

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()
```



Most users who have rated the movies are Male

```
#Distribution by Occupation:

plt.figure(figsize=(9, 7))
sns.countplot(y='Occupation', data=data, palette='bright')
plt.title('User Occupation Distribution')
plt.xlabel('Number of Users')
plt.ylabel('Occupation')
plt.show()
```

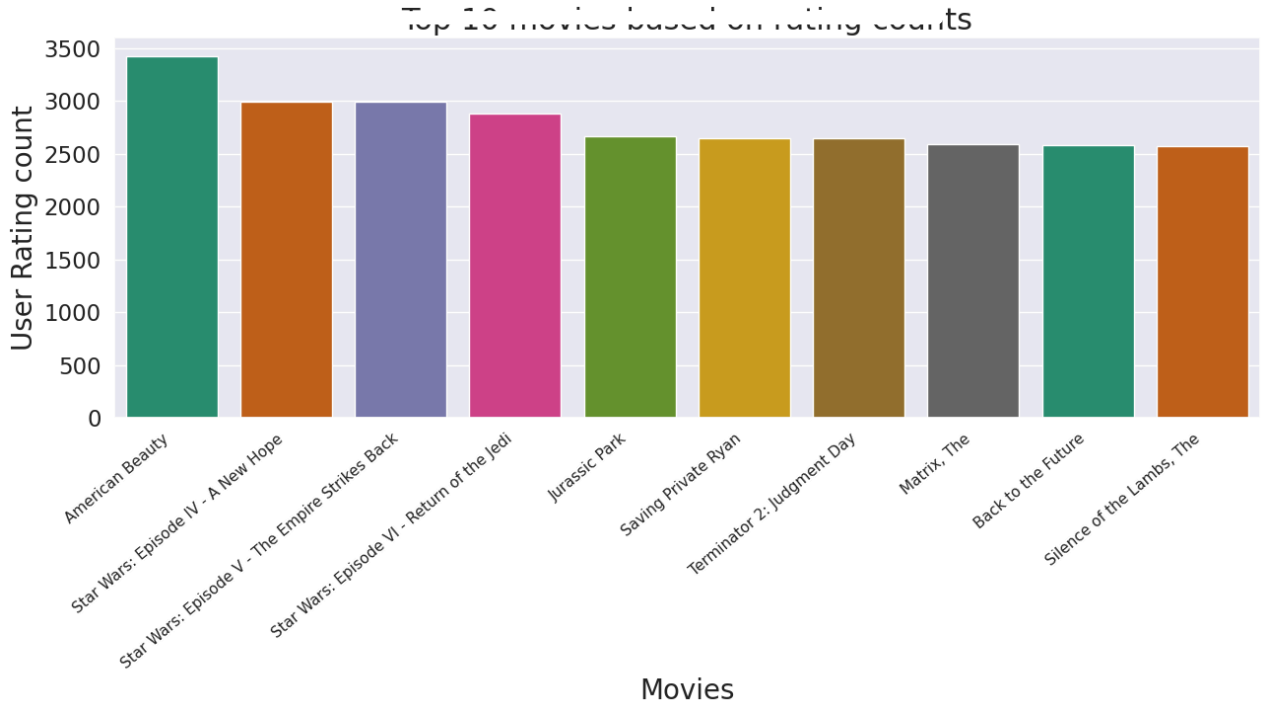


College/ Grad students and Others are the top categories who have rated the movies occupation-wise.

```
movies_rating_count = data.groupby(by = ['Title'])['Rating'].count().reset_index()[['Title', 'Rating']] ##
movies_rating_count.rename(columns = {'Rating': 'totalRatingCount'},inplace=True)
```

```
top10_movies=movies_rating_count[['Title', 'totalRatingCount']].sort_values(by = 'totalRatingCount',ascendi

plt.figure(figsize=(15,5))
ax=sns.barplot(x="Title", y="totalRatingCount", 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)
```

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



The movie, American Beauty has got maximum number of ratings by count.

Recommendation Systems

User-Interaction Matrix

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

```
matrix = pd.pivot_table(data, index='UserID', columns='Title', values='Rating', aggfunc='mean')
matrix.fillna(0, inplace=True) # Imputing 'NaN' values with Zero rating

print(matrix.shape)

matrix.head(10)
```

```
(6040, 3640)
```

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	187	2 Days in the Valley	20 Dates	20,000 Leagues Under the Sea
UserID														
1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
10	0.00	0.00	0.00	4.00	0.00	0.00	0.00	0.00	3.00	4.00	0.00	0.00	0.00	4.00
100	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
1000	0.00	0.00	0.00	0.00	0.00	0.00	0.00	4.00	0.00	0.00	0.00	0.00	0.00	0.00
1001	0.00	0.00	0.00	0.00	0.00	0.00	0.00	3.00	0.00	0.00	0.00	0.00	0.00	0.00
1002	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
1003	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
1004	0.00	0.00	0.00	0.00	0.00	0.00	0.00	4.00	0.00	0.00	0.00	0.00	0.00	0.00
1005	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
1006	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

```
# Checking the data sparsity
n_users = data['UserID'].nunique()
n_movies = data['MovieID'].nunique()
sparsity = round(1.0 - data.shape[0] / float( n_users * n_movies), 3)
print('The sparsity level of dataset is ' +  str(sparsity * 100) + '%')
```

```
The sparsity level of dataset is 95.5%
```

Pearson Correlation

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)

```
#Item-Based approach

#Taking a movie exmaple to check which 5 other movies are highly correlated with the given movie.

data[data['Title']=='Home Alone']
```

	MovieID	Title	Genres	Year	UserID	Rating	Timestamp	Gender	Age	Occupation	ZipCode	Datetime	Re
156020	586	Home Alone	[Children's, Comedy]	1990	10	3	978228747	F	35-44	academic/educator	95370	2000-12-31 02:12:27	
156021	586	Home Alone	[Children's, Comedy]	1990	11	1	978904356	F	25-34	academic/educator	04093	2001-01-07 21:52:36	
156022	586	Home Alone	[Children's, Comedy]	1990	18	4	978155233	F	18-24	clerical/admin	95825	2000-12-30 05:47:13	
156023	586	Home Alone	[Children's, Comedy]	1990	22	3	978154267	M	18-24	scientist	53706	2000-12-30 05:31:07	
156024	586	Home Alone	[Children's, Comedy]	1990	26	2	978140049	M	25-34	executive/managerial	23112	2000-12-30 01:34:09	
...	...	...	...	...	...	...	...	...	...	...	...	...	

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

```
UserID
1      0.00
10     3.00
100    0.00
1000   0.00
1001   0.00
       ...
995    0.00
996    0.00
997    0.00
998    0.00
999    0.00
Name: Home Alone, Length: 6040, dtype: float64
```

```
similar_movies = matrix.corrwith(movie_rating) #Finding similar movies
display(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.
```

Title	0
$1,000,000 Duck	0.15
'Night Mother	0.05
'Til There Was You	0.08
'burbs, The	0.25
...And Justice for All	0.04
...	...
Zed & Two Noughts, A	0.00
Zero Effect	0.15
Zero Kelvin (Kjærlighetens kjøtere)	-0.01
Zeus and Roxanne	0.09
eXistenZ	0.05

3640 rows × 1 columns

dtype: float64

Title	Correlation
Home Alone 2: Lost in New York	0.55
Mrs. Doubtfire	0.47
Liar Liar	0.46
Mighty Ducks, The	0.45
Sister Act	0.44

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.

```
item_sim = cosine_similarity(matrix.T) #Finding the similarity values between item-item using cosine_simila
item_sim
```

```
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.        ]])
```

```
item_sim.shape
```

```
(3640, 3640)
```

Item-Based Similarity

```
item_sim_matrix = pd.DataFrame(item_sim, index=matrix.columns, columns=matrix.columns)
item_sim_matrix.head() #Item-similarity Matrix
```

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	187	2 Days in the Valley	20 Dates	20,00 League Unde the Se
Title														
$1,000,000 Duck	1.00	0.07	0.04	0.08	0.06	0.00	0.06	0.19	0.09	0.06	0.03	0.02	0.02	0.1
'Night Mother	0.07	1.00	0.12	0.12	0.16	0.00	0.08	0.14	0.11	0.05	0.06	0.11	0.04	0.0
'Til There Was You	0.04	0.12	1.00	0.10	0.07	0.08	0.13	0.13	0.08	0.07	0.02	0.07	0.09	0.0
'burbs, The	0.08	0.12	0.10	1.00	0.14	0.00	0.19	0.25	0.17	0.20	0.10	0.18	0.05	0.1
...And Justice for All	0.06	0.16	0.07	0.14	1.00	0.00	0.08	0.18	0.21	0.12	0.11	0.20	0.04	0.1

User-Based Similarity

```
user_sim = cosine_similarity(matrix) #Finding the similarity values between user-user using cosine_similari
user_sim
```

```
array([[1.        , 0.25531859, 0.12396703, ..., 0.15926709, 0.11935626,
        0.12239079],
       [0.25531859, 1.        , 0.25964457, ..., 0.16569953, 0.13332665,
        0.24845029],
       [0.12396703, 0.25964457, 1.        , ..., 0.20430203, 0.11352239,
        0.30693676],
       ...,
       [0.15926709, 0.16569953, 0.20430203, ..., 1.        , 0.18657496,
        0.18563871],
       [0.11935626, 0.13332665, 0.11352239, ..., 0.18657496, 1.        ,
        0.10827118],
       [0.12239079, 0.24845029, 0.30693676, ..., 0.18563871, 0.10827118,
        1.        ]])
```

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

UserID	1	10	100	1000	1001	1002	1003	1004	1005	1006	1007	1008	1009	101	1010	1011	1012	1013	1014	1015
UserID																				
1	1.00	0.26	0.12	0.21	0.14	0.11	0.12	0.18	0.10	0.05	0.06	0.10	0.05	0.03	0.16	0.08	0.08	0.05	0.20	0.18
10	0.26	1.00	0.26	0.28	0.16	0.11	0.14	0.43	0.19	0.10	0.16	0.22	0.12	0.21	0.35	0.20	0.15	0.16	0.16	0.39
100	0.12	0.26	1.00	0.31	0.08	0.11	0.36	0.24	0.17	0.10	0.06	0.04	0.06	0.35	0.26	0.14	0.09	0.02	0.30	0.20
1000	0.21	0.28	0.31	1.00	0.10	0.05	0.20	0.36	0.33	0.13	0.04	0.08	0.12	0.28	0.25	0.12	0.12	0.05	0.18	0.22
1001	0.14	0.16	0.08	0.10	1.00	0.16	0.05	0.15	0.14	0.13	0.02	0.08	0.20	0.07	0.25	0.07	0.04	0.07	0.06	0.30

Nearest Neighbours

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

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

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

```
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
```

	Title1	Title2	Title3	Title4	Title5	Title6
0	0	735	416	286	584	3247
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

```
##With this for loop replacing the indices in the result dataframe with movie titles of that corresponding
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()
```

	Title1	Title2	Title3	Title4	Title5	Title6
0	$1,000,000 Duck	Computer Wore Tennis Shoes, The	Blackbeard's Ghost	Barefoot Executive, The	Candleshoe	That Darn Cat!
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

```
movie_name = 'Liar Liar'
result2.loc[result2['Title1']==movie_name] #5 nearest movies for the movie present in Title1.
```

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

Matrix Factorization

```
rm = data.pivot(index = 'UserID', columns ='MovieID', values = 'Rating').fillna(0)
rm.head()
```

MovieID	1	10	100	1000	1002	1003	1004	1005	1006	1007	1008	1009	101	1010	1011	1012	1013	1014	1015	1016
UserID																				
1	5.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
10	5.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.00	0.00	0.00	0.00	3.00	0.00	0.00	3.00	5.00
100	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
1000	5.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
1001	4.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

-Using Cmfrec library -->

```
user_itm = data[['UserID', 'MovieID', 'Rating']].copy()
user_itm.columns = ['UserId', 'ItemId', 'Rating']  # Lib requires specific column names
user_itm.head(2)
```

	UserId	ItemId	Rating
0	1	1	5
1	6	1	4

```
print(user_itm.shape)
print("No.of Users:",len(user_itm['UserId'].unique()))
print("No.of Items:",len(user_itm['ItemId'].unique()))
```

```
(996144, 3)
No.of Users: 6040
No.of Items: 3682
```

```
!pip install cmfrec
from cmfrec import CMF
```

```
Requirement already satisfied: cmfrec in /usr/local/lib/python3.12/dist-packages (3.5.1.post13)
Requirement already satisfied: cython in /usr/local/lib/python3.12/dist-packages (from cmfrec) (3.0.12)
Requirement already satisfied: numpy>=1.25 in /usr/local/lib/python3.12/dist-packages (from cmfrec) (2.0.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.12/dist-packages (from cmfrec) (1.16.3)
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist-packages (from cmfrec) (2.2.2)
Requirement already satisfied: findblas in /usr/local/lib/python3.12/dist-packages (from cmfrec) (0.1.26.post1)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas->cmfrec) (2.9.
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas->cmfrec) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas->cmfrec) (2025.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2->pandas->cmf
```

```
model = CMF(method="als", k=4, lambda_=0.1, user_bias=False, item_bias=False, verbose=False)
model.fit(user_itm) #Fitting the model
```

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

```
model.A_.shape, model.B_.shape #model.A_ gives the embeddings of Users and model.B_ gives the embeddings of I
```

```
((6040, 4), (3682, 4))
```

```
user_itm.Rating.mean(), model.glob_mean_  # Average rating and Global Mean
```

```
(np.float64(3.57998542379415), 3.5799853801727295)
```

```
rm__ = np.dot(model.A_, model.B_.T) + model.glob_mean_ #Calculating the predicted ratings
rmse = mean_squared_error(rm.values[rm > 0], rm__[rm > 0])#, squared=False) # calculating rmse value
print('Root Mean Squared Error: {:.3f}'.format(rmse))
mape =  mean_absolute_percentage_error(rm.values[rm > 0], rm__[rm > 0]) #calculating mape value
print('Mean Absolute Percentage Error: {:.3f}'.format(mape))
```

```
Root Mean Squared Error: 2.156
Mean Absolute Percentage Error: 0.421
```

Embeddings for User-user similarity

```
user=cosine_similarity(model.A_)

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
```

UserID	1	10	100	1000	1001	1002	1003	1004	1005	1006	1007	1008	1009	101	1010	1011	1012	1013	1014	10
UserID																				
1	1.00	-0.09	0.36	-0.25	0.72	0.33	-0.01	-0.63	-0.45	-0.57	-0.46	0.42	0.84	-0.03	0.05	-0.49	-0.46	0.10	0.76	0.
10	-0.09	1.00	-0.51	0.06	0.58	0.33	0.15	-0.57	0.04	0.78	0.29	0.37	-0.24	0.83	0.90	0.84	0.54	0.11	0.45	0.
100	0.36	-0.51	1.00	0.66	-0.07	0.49	0.66	0.44	0.45	-0.26	0.13	0.59	0.77	-0.01	-0.19	-0.33	0.17	0.43	0.32	0.
1000	-0.25	0.06	0.66	1.00	-0.20	0.54	0.92	0.58	0.89	0.53	0.57	0.69	0.24	0.51	0.27	0.42	0.84	0.47	0.17	0.
1001	0.72	0.58	-0.07	-0.20	1.00	0.60	-0.02	-0.88	-0.45	0.05	-0.00	0.50	0.53	0.56	0.69	0.21	-0.01	0.34	0.91	0.

```
itm=cosine_similarity(model.B_)

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

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	1.00	0.26	-0.06	-0.14	0.04	0.68	0.30	0.07	-0.30	0.50	0.81	-0.53	0.33	0.32	-0.22	0.43	0.79	-0.21	-0.51	-0.33	0.67
2	0.26	1.00	0.93	0.82	0.95	0.15	0.95	0.95	0.78	0.77	0.72	0.66	0.92	0.42	0.88	0.07	0.39	0.29	0.57	0.80	0.38
3	-0.06	0.93	1.00	0.78	0.97	-0.00	0.89	0.89	0.94	0.73	0.51	0.81	0.74	0.14	0.98	0.00	0.02	0.46	0.81	0.95	0.10
4	-0.14	0.82	0.78	1.00	0.84	-0.28	0.73	0.95	0.71	0.29	0.27	0.88	0.89	0.62	0.86	-0.29	0.32	0.09	0.58	0.79	0.21
5	0.04	0.95	0.97	0.84	1.00	-0.11	0.96	0.92	0.83	0.67	0.60	0.77	0.83	0.24	0.95	-0.17	0.19	0.23	0.65	0.87	0.08

```
#Taking an example:

movie_name='586'
movie_rating = itm_sim_matrix[movie_name] # Taking the ratings of that movie
print(movie_rating)
```

```
1       0.25
2       0.96
3       0.94
4       0.70
5       0.97
        ...
3948    0.53
3949   -0.38
3950    0.44
3951    0.11
3952    0.45
Name: 586, Length: 3682, dtype: float32
```

```
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.
```

	Correlation
2875	1.00
3482	1.00
3565	1.00
3594	1.00
653	1.00

```
item_mov = data[['MovieID', 'Title']].copy()
item_mov.drop_duplicates(inplace=True)
item_mov.reset_index(drop=True,inplace=True)

sim_df1= sim_df.copy()
sim_df1.reset_index(inplace=True)
sim_df1.rename(columns = {'index':'MovieID'}, inplace = True)
sim_mov = pd.merge(sim_df1,item_mov,on='MovieID',how='inner')
sim_mov.head(6)
```

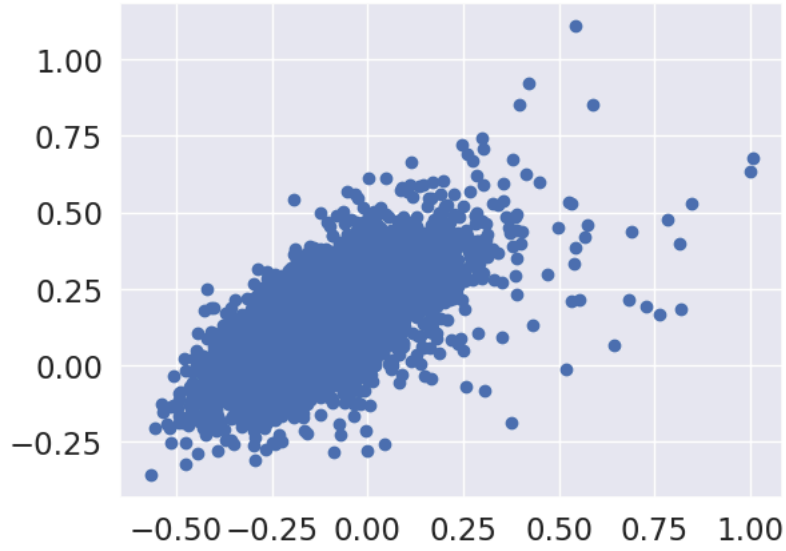
	MovieID	Correlation	Title
0	586	1.00	Home Alone
1	2875	1.00	Sommersby
2	3482	1.00	Price of Glory
3	3565	1.00	Where the Heart Is
4	3594	1.00	Center Stage
5	653	1.00	Dragonheart

```
model1 = CMF(method="als", k=2, lambda_=0.1, user_bias=False, item_bias=False, verbose=False)
model1.fit(user_itm)
```

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

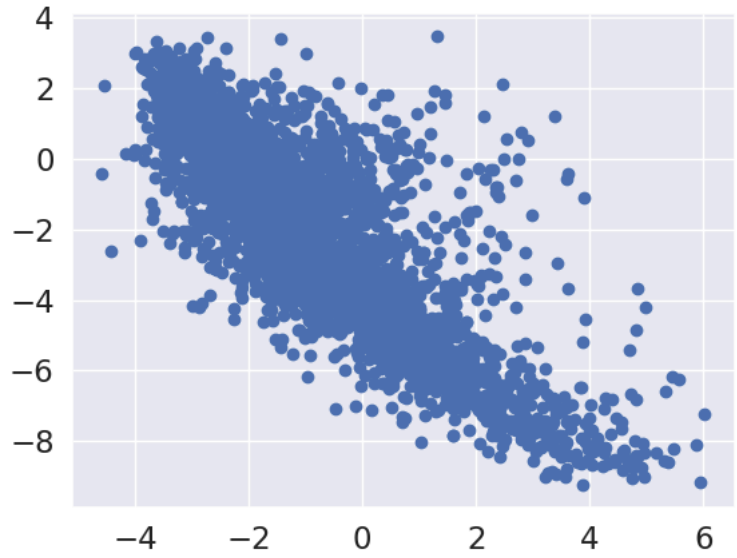
```
plt.scatter(model1.A_[:, 0], model1.A_[:, 1], cmap = 'hot')
```

```
<matplotlib.collections.PathCollection at 0x7b13acd6e6f0>
```




```
plt.scatter(model1.B_[:, 0], model1.B_[:, 1], cmap='hot')
```

```
<matplotlib.collections.PathCollection at 0x7b13ac533ce0>
```



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? : a.70s b. 90s c. 50s d.80s- **b.90s**
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 **0 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]]

```
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)
```

```
<Compressed Sparse Row sparse matrix of dtype 'int64'
        with 3 stored elements and shape (2, 2)>
  Coords        Values
  (0, 0)        1
  (1, 0)        3
  (1, 1)        7
```

LETs Try with Metrics

```
!pip install scikit-surprise
```

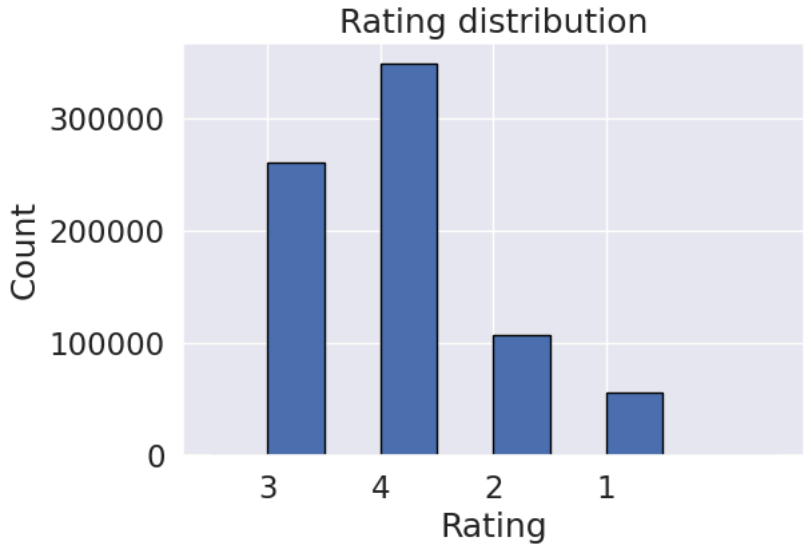
```
Requirement already satisfied: scikit-surprise in /usr/local/lib/python3.12/dist-packages (1.1.4)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from scikit-surprise) (1.5.2)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.12/dist-packages (from scikit-surprise) (2.0.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.12/dist-packages (from scikit-surprise) (1.16.3)
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.metrics.pairwise import cosine_similarity
from sklearn.neighbors import NearestNeighbors

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

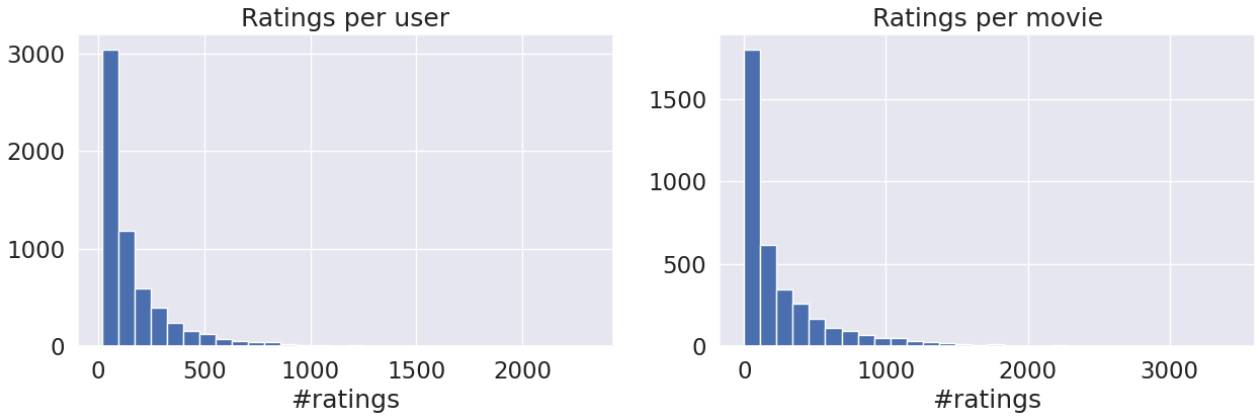
```
ratings.head(), users.head(), movies.head()
```

```
(  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,
   UserID Gender           Age            Occupation ZipCode
 0      1      F      Under 18          k-12 student   48067
 1      2      M  56 and above         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,
   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)
```

```
plt.figure(figsize=(6,4))
ratings['Rating'].hist(bins=np.arange(0.5,5.6,0.5), edgecolor='black')
plt.xlabel("Rating")
plt.ylabel("Count")
plt.title("Rating distribution")
plt.show()
```




```
ratings_per_user = ratings.groupby('UserID').size()
ratings_per_movie = ratings.groupby('MovieID').size()

fig, ax = plt.subplots(1,2, figsize=(15,4))
ax[0].hist(ratings_per_user, bins=30)
ax[0].set_title("Ratings per user")
ax[0].set_xlabel("#ratings")

ax[1].hist(ratings_per_movie, bins=30)
ax[1].set_title("Ratings per movie")
ax[1].set_xlabel("#ratings")
plt.show()
```




```
ratings_users = ratings.merge(users, on="UserID", how="left")
ratings_full  = ratings_users.merge(movies, on="MovieID", how="left")
display(ratings_full.head())
display(data)
```

	UserID	MovieID	Rating	Timestamp	Gender	Age	Occupation	ZipCode	Title	Genres	Year
0	1	1193	5	978300760	F	Under 18	k-12 student	48067	One Flew Over the Cuckoo's Nest	Drama	1975
1	1	661	3	978302109	F	Under 18	k-12 student	48067	James and the Giant Peach	Animation|Children's|Musical	1996
2	1	914	3	978301968	F	Under 18	k-12 student	48067	My Fair Lady	Musical|Romance	1964
3	1	3408	4	978300275	F	Under 18	k-12 student	48067	Erin Brockovich	Drama	2000
4	1	2355	5	978824291	F	Under 18	k-12 student	48067	Bug's Life, A	Animation|Children's|Comedy	1998

	MovieID	Title	Genres	Year	UserID	Rating	Timestamp	Gender	Age	Occupation	ZipCode	Datetime
0	1	Toy Story	[Animation, Children's, Comedy]	1995	1	5	978824268	F	Under 18	k-12 student	48067	2001-01-06 23:37:48
1	1	Toy Story	[Animation, Children's, Comedy]	1995	6	4	978237008	F	50-55	homemaker	55117	2000-12-31 04:30:08
2	1	Toy Story	[Animation, Children's, Comedy]	1995	8	4	978233496	M	25-34	programmer	11413	2000-12-31 03:31:36
3	1	Toy Story	[Animation, Children's, Comedy]	1995	9	5	978225952	M	25-34	technician/engineer	61614	2000-12-31 01:25:52

Build a User–Item Rating Matrix (Pearson + cosine)

```
ratings
```

	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
...	...	...	...	...
1000204	6040	1091	1	956716541
1000205	6040	1094	5	956704887
1000206	6040	562	5	956704746
1000207	6040	1096	4	956715648
1000208	6040	1097	4	956715569

1000209 rows × 4 columns

```
# Ensure 'Rating' column is numeric
ratings['Rating'] = pd.to_numeric(ratings['Rating'])

# Pivot: rows = users, columns = movies, values = ratings
user_item_matrix = ratings.pivot_table(index='UserID',
                                       columns='MovieID',
                                       values='Rating', aggfunc='mean')

print(user_item_matrix.shape)
user_item_matrix.head()
# matrix will be very sparse (lots of NaNs), which is normal.
```

```
(6040, 3706)
```

MovieID	1	10	100	1000	1002	1003	1004	1005	1006	1007	1008	1009	101	1010	1011	1012	1013	1014	1015	1016
UserID																				
1	5.00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
10	5.00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	5.00	NaN	NaN	NaN	3.00	NaN	NaN	3.00	5.00
100	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1000	5.00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1001	4.00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Item-based Collaborative Filtering – Pearson Correlation

- Compute Pearson similarity for a target movie

We won't compute the full correlation matrix for all movies (can be large). Instead, we write a function that, given a movie ID, finds similar movies via Pearson correlation.

```
def get_similar_movies_pearson(target_movie_id, min_common_users=30, top_n=10):
    """
    Compute Pearson correlation between a given movie and all others.
    - target_movie_id: movie ID in the matrix
    - min_common_users: minimum # users who rated both films
    - top_n: number of similar movies to return
    """
    # Ratings for the target movie (column)
    target_ratings = user_item_matrix[target_movie_id]

    # Movies that have at least some overlap of users
    correlations = []
    for movie_id in user_item_matrix.columns:
        if movie_id == target_movie_id:
            continue

        other_ratings = user_item_matrix[movie_id]

        # Users who rated both movies
        common_users = target_ratings.notna() & other_ratings.notna()
        n_common = common_users.sum()
        if n_common < min_common_users:
            continue

        corr = target_ratings[common_users].corr(other_ratings[common_users])
        if np.isnan(corr):
            continue
        correlations.append((movie_id, corr, n_common))

    # Sort by correlation
    correlations.sort(key=lambda x: x[1], reverse=True)
    return correlations[:top_n]
```

```
#Convenience wrapper to return titles
def print_similar_movies_pearson(movie_title, **kwargs):
    movie_id = movies.loc[movies['Title'] == movie_title, 'MovieID'].iloc[0]
    sims = get_similar_movies_pearson(movie_id, **kwargs)
    print(f"Top similar movies to '{movie_title}' (Pearson):")
    for mid, corr, n_common in sims:
        title = movies.loc[movies['MovieID'] == mid, 'Title'].values[0]
        print(f"{title:50s}  corr={corr:.3f}  common_users={n_common}")
```

```
print_similar_movies_pearson('Liar Liar', min_common_users=20, top_n=5)
```

```
Top similar movies to 'Liar Liar' (Pearson):
Heavyweights                                       corr=0.713  common_users=22
Ruling Class, The                                  corr=0.712  common_users=20
It Takes Two                                       corr=0.612  common_users=22
Scarlet Letter, The                                corr=0.609  common_users=33
Play it to the Bone                                corr=0.602  common_users=26
```

Item-based Collaborative Filtering – Cosine Similarity

For cosine similarity we typically work on a matrix where NaNs are replaced. A simple option: fill missing with 0 (meaning "no rating / neutral"). In production we'd be more careful, but for learning this is fine.

```
# Fill NaNs with 0 for cosine similarity
user_item_filled = user_item_matrix.fillna(0)

# Fit k-nearest neighbors using cosine distance
knn_model = NearestNeighbors(metric='cosine', algorithm='brute')
knn_model.fit(user_item_filled.T)   # transpose: rows=movies, cols=users
```

```
NearestNeighbors
NearestNeighbors(algorithm='brute', metric='cosine')
```

Function to fetch similar movies by cosine similarity

```
def get_similar_movies_cosine(movie_title, n_neighbors=10):
    movie_id = movies.loc[movies['Title'] == movie_title, 'MovieID'].iloc[0]
    movie_vector = user_item_filled.T.loc[movie_id].values.reshape(1,-1)

    distances, indices = knn_model.kneighbors(movie_vector, n_neighbors=n_neighbors+1)
    # first neighbor will be the movie itself (distance 0)
    similar_ids = user_item_filled.columns[indices.flatten()[1:]]  # skip self
    distances = distances.flatten()[1:]

    print(f"Top similar movies to '{movie_title}' (Cosine):")
    for mid, dist in zip(similar_ids, distances):
        title = movies.loc[movies['MovieID'] == mid, 'Title'].values[0]
        sim = 1 - dist
        print(f"{title:50s}  cosine_similarity={sim:.3f}")
```

```
get_similar_movies_cosine('Liar Liar', n_neighbors=5)
```

```
Top similar movies to 'Liar Liar' (Cosine):
Mrs. Doubtfire                                     cosine_similarity=0.557
Ace Ventura: Pet Detective                         cosine_similarity=0.517
Dumb & Dumber                                      cosine_similarity=0.513
Home Alone                                         cosine_similarity=0.511
Wayne's World                                      cosine_similarity=0.499
```

```
# Matrix Factorization (SVD from Surprise)

# Now we move to a rating prediction model and evaluate with RMSE, MAPE + ranking metrics NDCG and MRR.

#  🔹 Prepare data for Surprise
reader = Reader(line_format='user item rating timestamp', sep='::')
data = Dataset.load_from_file('ratings.dat', reader=reader)


# (Surprise will internally parse the user_id, movie_id, rating, timestamp from the file.)

#  🔹 Train-test split
trainset, testset = train_test_split(data, test_size=0.2, random_state=42)

#  🔹 Train SVD (matrix factorization)
algo = SVD(n_factors=50, n_epochs=20, biased=True, random_state=42)
algo.fit(trainset)

#  🔹 Predictions and classical errors (RMSE, MAPE)
predictions = algo.test(testset)

# RMSE from Surprise
rmse = accuracy.rmse(predictions, verbose=False)
print(f"RMSE: {rmse:.4f}")

# Compute MAPE manually
def compute_mape(preds):
```

```
    abs_pct_errors = []
    for p in preds:
        true = p.r_ui
        est  = p.est
        # avoid division by 0; ratings are 1–5 anyway
        abs_pct_errors.append(abs(true - est) / abs(true))
    return np.mean(abs_pct_errors)

mape = compute_mape(predictions)
print(f"MAPE: {mape:.4f}")
```

Ranking Metrics: NDCG@K and MRR@K

For ranking metrics we need to simulate a top-N recommendation scenario per user.

High-level idea:

For each user in the test set:

Collect all (movie, true_rating, predicted_rating)

Sort movies by predicted_rating descending

Define relevance = 1 if true_rating ≥ 4, else 0

Compute DCG@K and IDCG@K → NDCG@K

Compute reciprocal rank of first relevant item → RR@K

Then average across users → NDCG@K and MRR@K.

- Helper functions: DCG, NDCG, MRR

```
from collections import defaultdict

def dcg_at_k(relevances, k):
    """
    relevances: list/array of binary or graded relevance scores, ordered by rank.
    """
    relevances = np.array(relevances)[:k]
    if relevances.size == 0:
        return 0.0
    discounts = 1 / np.log2(np.arange(2, relevances.size + 2))
    return np.sum(relevances * discounts)

def ndcg_at_k(relevances, k):
    dcg = dcg_at_k(relevances, k)
    # sort by true relevance to get ideal DCG
    ideal_relevances = sorted(relevances, reverse=True)
    idcg = dcg_at_k(ideal_relevances, k)
    if idcg == 0:
        return 0.0
    return dcg / idcg

def mrr_at_k(relevances, k):
    # find first position with relevance > 0
    for i, rel in enumerate(relevances[:k]):
        if rel > 0:
            return 1.0 / (i + 1)
    return 0.0
```

Organize predictions by user

```
user_predictions = defaultdict(list)

for p in predictions:
    user_predictions[p.uid].append({
        'movie_id': p.iid,
        'true_rating': p.r_ui,
        'est_rating': p.est
    })
```

```
user_predictions = defaultdict(list)

for p in predictions:
    user_predictions[p.uid].append({
        'movie_id': p.iid,
        'true_rating': p.r_ui,
        'est_rating': p.est
    })
```

Compute NDCG@K and MRR@K over users

```
def evaluate_ranking_metrics(user_preds, k=10, relevance_threshold=4.0):
    ndcgs = []
    mrrs  = []

    for user, items in user_preds.items():
        # Sort movies for this user by predicted rating (desc)
        items_sorted = sorted(items, key=lambda x: x['est_rating'], reverse=True)

        # Relevance: 1 if true rating >= threshold else 0
        relevances = [1 if x['true_rating'] >= relevance_threshold else 0
                      for x in items_sorted]

        if len(relevances) == 0:
            continue

        ndcgs.append(ndcg_at_k(relevances, k))
        mrrs.append(mrr_at_k(relevances, k))

    mean_ndcg = np.mean(ndcgs) if ndcgs else 0.0
    mean_mrr  = np.mean(mrrs)  if mrrs else 0.0
    return mean_ndcg, mean_mrr

ndcg10, mrr10 = evaluate_ranking_metrics(user_predictions, k=10, relevance_threshold=4.0)
print(f"NDCG@10: {ndcg10:.4f}")
print(f"MRR@10 : {mrr10:.4f}")
```

Making Actual Recommendations for a User

Now that you have an MF model (algo), you can recommend top-N unseen movies for a user.

```
def recommend_movies_for_user(user_id, algo, movies_df, ratings_df, n_recs=10):
    # Movies the user has already rated
    rated_movies = set(ratings_df.loc[ratings_df['user_id'] == user_id, 'movie_id'])

    # Candidate movies = all movies minus those already rated
    all_movies = set(movies_df['movie_id'])
    candidates = list(all_movies - rated_movies)

    # Predict ratings for all candidate movies
    preds = [ (mid, algo.predict(user_id, mid).est) for mid in candidates ]
    preds.sort(key=lambda x: x[1], reverse=True)
    top = preds[:n_recs]

    # Attach titles
    recs = []
    for mid, score in top:
        title = movies_df.loc[movies_df['movie_id'] == mid, 'title'].values[0]
        recs.append((mid, title, score))
    return recs
```

```
sample_user = ratings['user_id'].iloc[0]
recs = recommend_movies_for_user(sample_user, algo, movies, ratings, n_recs=10)

print(f"Top 10 recommendations for user {sample_user}:")
for mid, title, score in recs:
```

```
        print(f"{title:50s}  predicted rating={score: 2f}")
```

More Explanation Model :

When you document this notebook for Zee:

Explain modeling choices

Item-based CF (Pearson & cosine) for interpretable “because you liked X”.

Matrix factorization (SVD) to capture latent taste factors and handle sparsity.

Report metrics

RMSE and MAPE ⟶ how close your predicted ratings are.

NDCG@10 and MRR@10 ⟶ how good your top-N recommendations are.

NDCG rewards getting many relevant movies high in the list.

MRR focuses on “how quickly do we show the first truly good movie?”.

Business framing

High NDCG ⟶ users see several good picks on their first screen ⟶ higher engagement.

Good MRR ⟶ at least one very relevant title appears early ⟶ reduces bounce.

Add demographic-aware models (e.g., using age/occupation as additional features).

Compare user-based vs item-based approaches.

Analysed by : **KASI**