```
# Problem Statement
'''Create a Recommender System to show personalized movie recommendations based on ratings given by a user and other users similar to them
in order to improve user experience.'''
```

```
'Create a Recommender System to show personalized movie recommendations based on ratings given by a user and other users similar to them \nin order to improve user experience.'
```

```
import pandas as pd
import numpy as np
import matplotlib as mpl
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
import keras
from tensorflow.keras.optimizers import Adam
from keras.layers import Input, Embedding, Flatten
from keras.layers import dot
from pylab import rcParams
```

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

```
movies = pd.read_csv("/content/zee-movies.dat", delimiter='\t', encoding='latin1')
```

```
ratings = pd.read_csv("/content/zee-ratings.dat", delimiter='\t', encoding='latin1')
users = pd.read_csv("/content/zee-users.dat", delimiter='\t', encoding='latin1')
```

```
# DATA FORMATTING
```

```
movies.head()
```

	Movie ID::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

Next steps: Generate code with movies View recommended plots

```
delimiter = '::'

# Split the existing columns in the DataFrame
movies_split = movies['Movie ID::Title::Genres'].str.split(delimiter, expand=True)
movies_split.columns = ['Movie ID', 'Title', 'Genres']

# Display the first few rows of the DataFrame
movies_split.head()
```

	Movie ID	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

Next steps: Generate code with movies_split View recommended plots

```
movies = movies_split
```

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

```
movies.sample(5)
```

	MovieID	Title	Genres
930	942	Laura (1944)	Crime|Film-Noir|Mystery
159	161	Crimson Tide (1995)	Drama|Thriller|War
1472	1504	Hollow Reed (1996)	Drama
617	621	My Favorite Season (1993)	Drama
2674	2743	Native Son (1986)	Drama

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

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

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

```
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	Zip-code
0	1	F	Under 18	k-12 student	48067
1	2	M	56 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

Next steps: Generate code with `users` View recommended plots

```
# Merging the dataframes
```

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

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

```
df_2 = pd.merge(df_1, users, how='inner', on='UserID')
df_2.head()
```

	MovieID	Title	Genres	UserID	Rating	Timestamp	Gender	Age	Occupation	Zip-code
0	1	Toy Story (1995)	Animation|Children's|Comedy	1	5	978824268	F	Under 18	k-12 student	48067
1	48	Pocahontas (1995)	Animation|Children's|Musical|Romance	1	5	978824351	F	Under 18	k-12 student	48067
2	150	Apollo 13 (1995)	Drama	1	5	978301777	F	Under 18	k-12 student	48067

```
data = df_2.copy(deep=True)
data.sample(10)
```

	MovieID	Title	Genres	UserID	Rating	Timestamp	Gender	Age	Occupation	Zip-code
504259	1290	Some Kind of Wonderful (1987)	Drama|Romance	5268	2	961169166	F	25-34	artist	68502
821264	296	Pulp Fiction (1994)	Crime|Drama	4516	5	964856871	M	18-24	college/grad student	53142
335422	3087	Scrooged (1988)	Comedy	3539	5	967016649	F	25-34	college/grad student	77006
439980	1376	Star Trek IV: The Voyage Home (1986)	Action|Adventure|Sci-Fi	4510	4	964989294	M	45-49	executive/managerial	92503
602674	1219	Psycho (1960)	Horror|Thriller	3224	4	968444451	F	25-34	sales/marketing	93428
975658	3551	Marathon Man (1976)	Thriller	1452	4	974756416	F	56 Above	retired	90732

```
# Performing Exploratory Data Analysis
```

```
print("No. of rows: ", data.shape[0])
print("No. of columns: ", data.shape[1])
```

```
No. of rows:  1000209
No. of columns:  10
```

```
# So there are 1000209 rows and 10 columns
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000209 entries, 0 to 1000208
Data columns (total 10 columns):
 #   Column      Non-Null Count    Dtype 
---  ------      --------------    ----- 
 0   MovieID     1000209 non-null  object
 1   Title       1000209 non-null  object
 2   Genres      1000209 non-null  object
 3   UserID      1000209 non-null  object
 4   Rating      1000209 non-null  object
 5   Timestamp   1000209 non-null  object
 6   Gender      1000209 non-null  object
 7   Age         1000209 non-null  object
 8   Occupation  1000209 non-null  object
```

```
 9   Zip-code    1000209 non-null  object
dtypes: object(10)
memory usage: 83.9+ MB
```

```
#Feature Engineering
```

```
data['Rating'].unique()
```

```
array(['5', '4', '3', '2', '1'], dtype=object)
```

```
Start coding or generate with AI.
```

```
0          5
1          5
2          5
3          4
4          5
          ..
1000204    4
1000205    2
1000206    5
1000207    3
1000208    5
Name: Rating, Length: 1000209, dtype: object
```

```
data.replace({'Rating':{'5:':'5'}}, inplace=True)
```

```
data['Rating'] = data['Rating'].astype('int32')
```

```
data['Datetime'] = pd.to_datetime(data['Timestamp'],
                                  unit='s')
```

```
data['ReleaseYear'] = data['Title'].str.rsplit(' ', 1).str[1]
data['ReleaseYear'] = data['ReleaseYear'].str.lstrip("(").str.rstrip(")")
```

```
data['ReleaseYear'].unique()
```

```
array(['1995', '1977', '1993', '1992', '1937', '1991', '1996', '1964',
       '1939', '1958', '1950', '1941', '1965', '1982', '1975', '1987',
       '1962', '1989', '1985', '1959', '1997', '1998', '1988', '1942',
       '1947', '1999', '1980', '1983', '1986', '1990', '2000', '1994',
       '1978', '1961', '1984', '1972', '1976', '1981', '1973', '1974',
       '1940', '1963', '1952', '1954', '1953', '1944', '1968', '1957',
       '1946', '1949', '1951', '1971', '1979', '1967', '1966', '1948',
```

```
        '1933', '1970', '1969', '1930', '1955', '1956', '1934', '1920',
        '1925', '1938', '1960', '1935', '1932', '1931', '1945', '1943',
        '1936', '1929', '1926', '1927', '1922', '1919', '1921', '1923',
        '1928'], dtype=object)
```

```
data['ReleaseYear'] = data['ReleaseYear'].astype('int32')
```

```
data['Title'] = data['Title'].str.rsplit(' ', 1).str[0]
```

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

```
data.sample(5)
```

	MovieID	Title	Genres	UserID	Rating	Timestamp	Gender	Age	Occupation	Zip-code	Datetime	ReleaseYear	ReleaseDec
931483	1230	Annie Hall	Comedy|Romance	2140	5	974637635	M	45-49	sales/marketing	46804	2000-11-19 12:40:35	1977	70s
661209	2918	Ferris Bueller's Day Off	Comedy	5744	5	958363764	F	25-34	college/grad student	66044	2000-05-15 04:09:24	1986	80s
540037	1292	Being There	Comedy	5643	4	958889850	F	35-44	academic/educator	84108	2000-05-21 06:17:30	1979	70s
421236	3441	Red Dawn	Action|War	4335	3	965339165	M	35-44	other	74011	2000-08-03 21:46:05	1984	80s
31940	2846	Adventures of Milo and Otis, The	Children's	392	3	976661135	M	18-24	executive/managerial	20037	2000-12-12 22:45:35	1986	80s

```
# Data Cleaning
```

```
# Checking for null values
```

```
data.isna().sum()
```

```
MovieID         0
Title           0
Genres          0
UserID          0
Rating          0
Timestamp       0
```

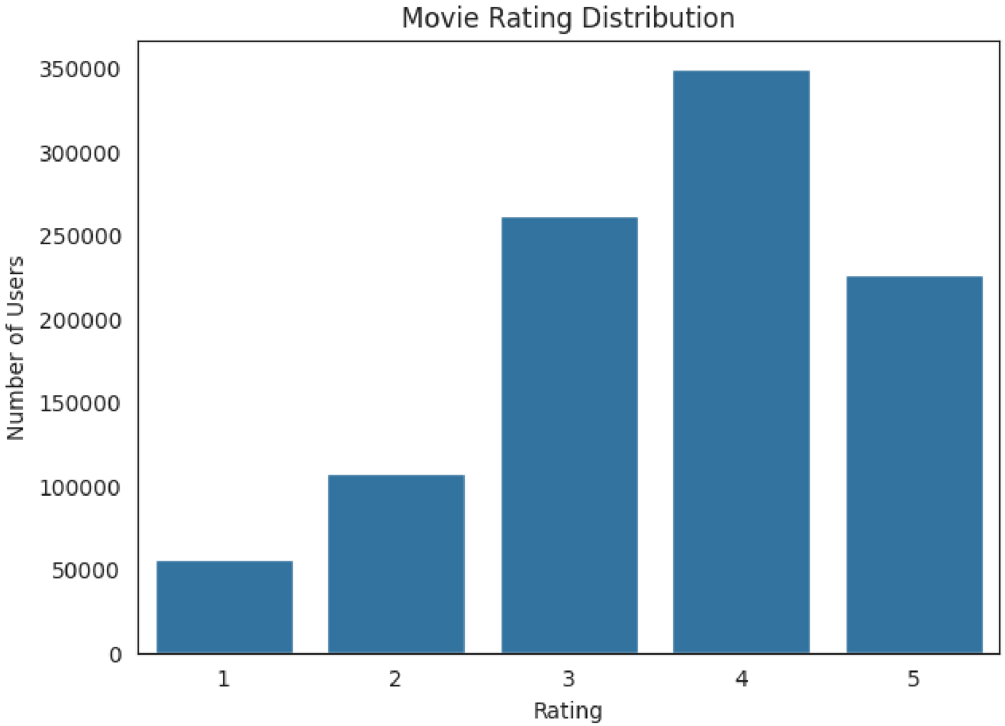
```
Gender          0
Age             0
Occupation      0
Zip-code        0
Datetime        0
ReleaseYear     0
ReleaseDec     45
dtype: int64
```

```
duplicate_rows = data[data.duplicated()]
print("No. of duplicate rows: ", duplicate_rows.shape[0])
```

```
No. of duplicate rows:  0
```

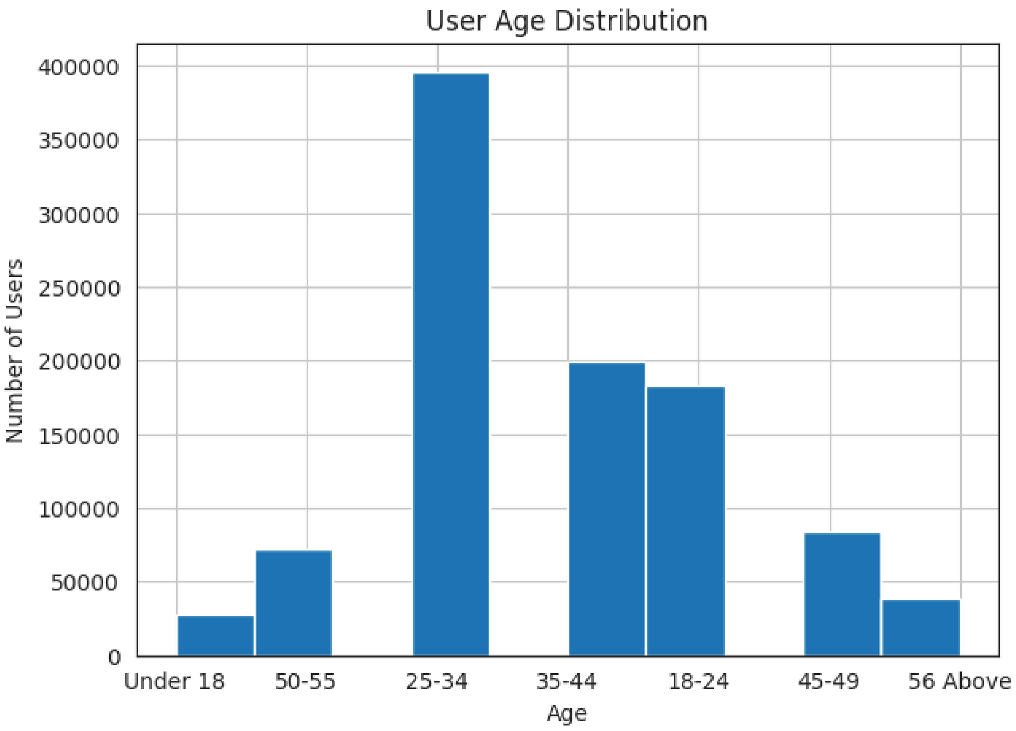
```
# Data Visulization
```

```
# Distribution of Movie ratings-
plt.figure(figsize=(7, 5))
sns.countplot(x='Rating', data=data)
plt.title('Movie Rating Distribution')
plt.xlabel('Rating')
plt.ylabel('Number of Users')
plt.show()
```



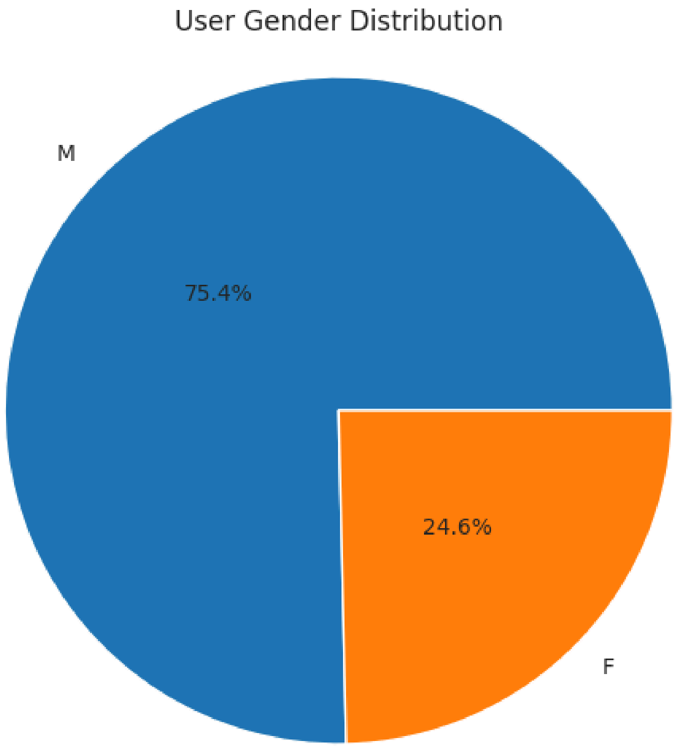

```
# As we can see most of the users have given rating 4
```

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




```
# As we can see users of age group from 25 - 34 are more
```

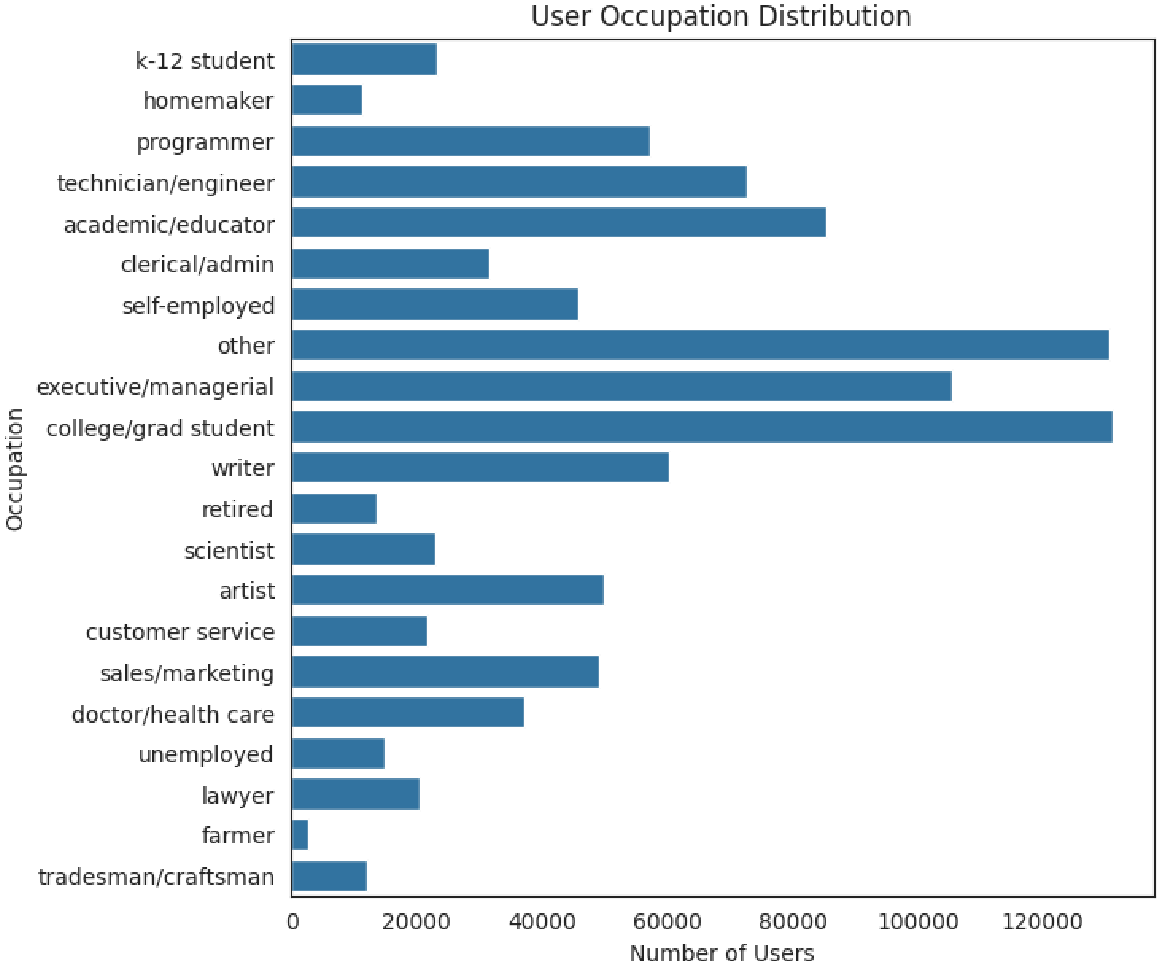
```
# Distribution by Gender
x = data['Gender'].value_counts().values
plt.figure(figsize=(7, 6))
plt.pie(x, center=(0, 0), radius=1.5, labels=['M','F'], autopct='%1.1f%%', pctdistance=0.5)
plt.title('User Gender Distribution')
plt.axis('equal')
plt.show()
data['Gender'].value_counts()
```




```
M    753769
F    246440
Name: Gender, dtype: int64
```

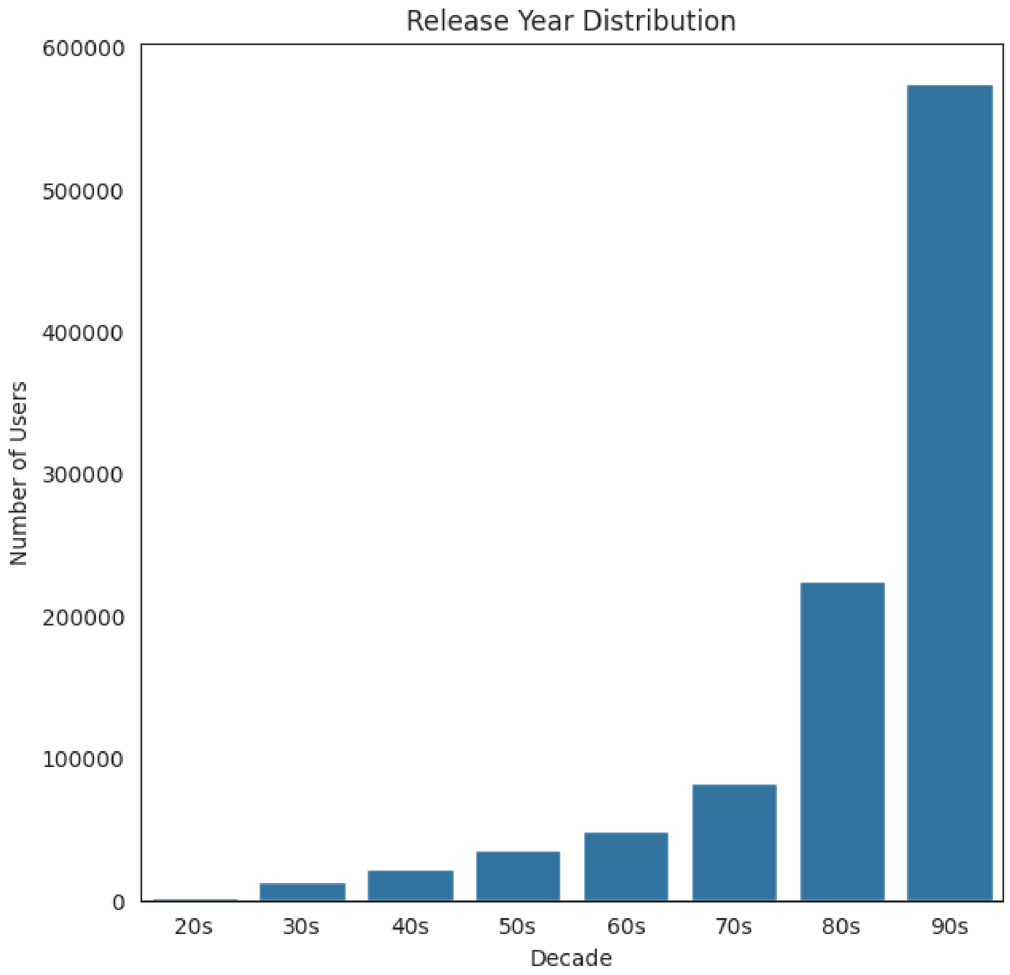
```
# As we can see compare female users male users are more by 75.4 percent
```

```
# Distribution by occupation
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()
```




```
# As we can see college/grad student and other category user are more
```

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




```
# As we can see movies which are released in 90s are more compared to others
```

```
# Grouping the data
```

```
# Average rating
data.groupby('Title')['Rating'].mean().sort_values(ascending=False).head(10)
```

```
Title
Smashing Time                          5.00
Song of Freedom                        5.00
One Little Indian                      5.00
```

```
Ulysses (Ulisse)                    5.00
Baby, The                           5.00
Follow the Bitch                    5.00
Schlafes Bruder (Brother of Sleep)  5.00
Gate of Heavenly Peace, The         5.00
Bittersweet Motel                   5.00
Lured                               5.00
Name: Rating, dtype: float64
```

```
# No. of ratings -
data.groupby('Title')['Rating'].count().sort_values(ascending=False).head(10)
```

```
Title
American Beauty                                  3428
Star Wars: Episode IV - A New Hope               2991
Star Wars: Episode V - The Empire Strikes Back   2990
Star Wars: Episode VI - Return of the Jedi       2883
Jurassic Park                                    2672
Saving Private Ryan                              2653
Terminator 2: Judgment Day                       2649
Matrix, The                                      2590
Back to the Future                               2583
Silence of the Lambs, The                        2578
Name: Rating, dtype: int64
```

```
df = pd.DataFrame(data.groupby('Title')['Rating'].agg([('Avg rating', 'mean')]))
df['No. of ratings'] = pd.DataFrame(data.groupby('Title')['Rating'].count())
```

```
df.sample(10)
```

Title	Avg rating	No. of ratings
Pollyanna	3.40	136
Night Tide	2.00	1
Quatermass II	3.21	38
Chairman of the Board	1.96	24
Jason's Lyric	3.27	40
Black Beauty	3.36	94
Edward Scissorhands	3.59	1473
My Name Is Joe	3.60	35
Shall We Dance? (Shall We Dansu?)	4.14	350
Homegrown	3.37	104

```
# In a Collaborative Filtering Recommender System:

# Memory-based methods utilize the entire dataset to make recommendations, while model-based methods create a model based on the dataset to make predict

# User-based approach compares a user's preferences with those of other users to make recommendations,
# while item-based approach compares the similarities between items to make recommendations based on the items a user has liked or interacted with.
```

```
# Pivot Table -
```

```
# Creating a pivot table of movie titles and userid -
mat = pd.pivot_table(data, index='UserID', columns='Title', values='Rating', aggfunc='mean')
mat.head(10)
```

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	20,0 Leagu Und the S
UserID														
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
10	NaN	NaN	NaN	4.00	NaN	NaN	NaN	NaN	3.00	4.00	NaN	NaN	NaN	4
100	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
1000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4.00	NaN	NaN	NaN	NaN	NaN	N
1001	NaN	NaN	NaN	NaN	NaN	NaN	NaN	3.00	NaN	NaN	NaN	NaN	NaN	N
1002	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
1003	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
1004	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4.00	NaN	NaN	NaN	NaN	NaN	N
1005	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
1006	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N

```
# Imputing 'NaN' values with Zero rating -

mat.fillna(0, inplace=True)

mat.shape
```

```
(6040, 3664)
```

```
# Pearson Correlation -

movie_name = input("Enter a movie name: ")
movie_rating = mat[movie_name]
```

```
Enter a movie name: 2001: A Space Odyssey
```

```
similar_movies = mat.corrwith(movie_rating)
```

```
sim_df = pd.DataFrame(similar_movies, columns=['Correlation'])
sim_df.sort_values('Correlation', ascending=False, inplace=True)
```

```
sim_df.iloc[1: , :].head()
```

Title	Correlation
Close Encounters of the Third Kind	0.52
Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb	0.51
Clockwork Orange, A	0.48
Blade Runner	0.47
Alien	0.45

```
# So these are the movies correlated with 2001: A Space Odyssey
```

```
# Cosine Similarity -
```

```
item_sim = cosine_similarity(mat.T)
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 similarity matrix -
item_sim_mat = pd.DataFrame(item_sim, index=mat.columns, columns=mat.columns)
item_sim_mat.head()
```

rk ty	Dark Command	Dark Crystal, The	Dark Half, The	Date with an Angel	Daughter of Dr. Jeckyll	Daughters of the Dust	Dave	Davy Crockett, King of the Wild Frontier	Day of the Beast, The (El Día de la bestia)	Day the Earth Stood Still, The	Daylight	Days of Heaven	Days of Thunder	Day
06	0.00	0.06	0.05	0.10	0.00	0.00	0.10	0.09	0.05	0.06	0.06	0.04	0.05	
08	0.06	0.08	0.07	0.09	0.00	0.09	0.11	0.08	0.06	0.06	0.06	0.16	0.06	
06	0.00	0.05	0.04	0.09	0.00	0.01	0.14	0.00	0.05	0.02	0.07	0.02	0.11	
17	0.06	0.24	0.12	0.18	0.00	0.04	0.25	0.11	0.09	0.13	0.17	0.05	0.20	
11	0.03	0.08	0.07	0.03	0.00	0.07	0.14	0.11	0.03	0.12	0.13	0.11	0.12	

```
user_sim = cosine_similarity(mat)
user_sim
```

```
array([[1.        , 0.2547356 , 0.12396703, ..., 0.15926709, 0.11935626,
         0.12239079],
        [0.2547356 , 1.        , 0.25905171, ..., 0.16532118, 0.13302222,
         0.24788299],
        [0.12396703, 0.25905171, 1.        , ..., 0.20430203, 0.11352239,
         0.30693676],
        ...,
        [0.15926709, 0.16532118, 0.20430203, ..., 1.        , 0.18657496,
         0.18563871],
        [0.11935626, 0.13302222, 0.11352239, ..., 0.18657496, 1.        ,
         0.10827118],
```

```
       [0.12239079, 0.24788299, 0.30693676, ..., 0.18563871, 0.10827118,
        1.        ]])
```

```
# User similarity matrix -
```

```
user_sim_mat = pd.DataFrame(user_sim, index=mat.index, columns=mat.index)
user_sim_mat.head()
```

UserID	1	10	100	1000	1001	1002	1003	1004	1005	1006	1007	1008	1009	101	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	102	10
UserID																										
1	1.00	0.25	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	0.13	0.15	0.18	0.11	0.12	0.
10	0.25	1.00	0.26	0.28	0.16	0.11	0.14	0.43	0.19	0.10	0.16	0.22	0.12	0.20	0.35	0.20	0.15	0.16	0.16	0.39	0.20	0.29	0.24	0.34	0.16	0.
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	0.29	0.20	0.17	0.14	0.34	0.19	0.01	0.
1000	0.21	0.28	0.31	1.00	0.10	0.05	0.20	0.36	0.32	0.13	0.04	0.08	0.12	0.28	0.25	0.12	0.12	0.05	0.18	0.22	0.09	0.20	0.36	0.20	0.10	0.
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	0.29	0.10	0.07	0.17	0.17	0.

```
# Nearest Neighbours -
csr_mat = sparse.csr_matrix(mat.T.values)
csr_mat
```

```
<3664x6040 sparse matrix of type '<class 'numpy.float64'>'
        with 997085 stored elements in Compressed Sparse Row format>
```

```
# Fitting the model with 'cosine similarity' as the distance metric and 5 (five) as the no. of nearest neighbors.
knn = NearestNeighbors(n_neighbors=5, metric='cosine', n_jobs=-1)
knn.fit(csr_mat)
```

```
▾           NearestNeighbors
NearestNeighbors(metric='cosine', n_jobs=-1)
```

```
# Making recommendations for a movie of the user's choice -
movie_name = input("Enter a movie name: ")
movie_index = mat.columns.get_loc(movie_name)
```

```
Enter a movie name: Date with an Angel
```

```
distances, indices = knn.kneighbors(mat[movie_name].values.reshape(1, -1), n_neighbors = 11)


for i in range(0, len(distances.flatten())):
    if i == 0:
        print('Recommendations for the movie: {0}\n'.format(movie_name))
    else:
        print('{0}: {1}, with distance of {2}'.format(i, mat.columns[indices.flatten()[i]], round(distances.flatten()[i], 3)))
```

```
Recommendations for the movie: Date with an Angel

1: Police Academy 2: Their First Assignment, with distance of 0.732
2: Who's That Girl?, with distance of 0.754
3: Grease 2, with distance of 0.759
4: Mr. Mom, with distance of 0.761
5: Harry and the Hendersons, with distance of 0.761
6: Volunteers, with distance of 0.763
7: Fatal Beauty, with distance of 0.767
8: Blind Date, with distance of 0.776
9: Under the Rainbow, with distance of 0.777
10: Police Academy 4: Citizens on Patrol, with distance of 0.778
```

```
# Matrix Factorization -
# Creating embeddings for both users and movies -


users = data.UserID.unique()
movies = data.MovieID.unique()

userid2idx = {o:i for i,o in enumerate(users)}
movieid2idx = {o:i for i,o in enumerate(movies)}


data['UserID'] = data['UserID'].apply(lambda x: userid2idx[x])
data['MovieID'] = data['MovieID'].apply(lambda x: movieid2idx[x])
split = np.random.rand(len(data)) < 0.8
train = data[split]
valid = data[~split]
print(train.shape, valid.shape)
```

```
(800174, 13) (200035, 13)
```

```
n_movies = len(data['MovieID'].unique())
n_users = len(data['UserID'].unique())

n_latent_factors = 64  # Hyperparamter
```

```
user_input = Input(shape=(1, ), name='user_input', dtype='int64')


user_embedding = Embedding(n_users, n_latent_factors, name='user_embedding')(user_input)


user_vec = Flatten(name='FlattenUsers')(user_embedding)


movie_input = Input(shape=(1, ), name='movie_input', dtype='int64')
movie_embedding = Embedding(n_movies, n_latent_factors, name='movie_embedding')(movie_input)
movie_vec = Flatten(name='FlattenMovies')(movie_embedding)


sim = dot([user_vec, movie_vec], name='Simalarity-Dot-Product', axes=1)
model = keras.models.Model([user_input, movie_input], sim)


model.compile(optimizer=Adam(learning_rate=1e-4), loss='mse')


# Let's see the model's summary -
model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 user_input (InputLayer)     [(None, 1)]                  0         []

 movie_input (InputLayer)    [(None, 1)]                  0         []

 user_embedding (Embedding)  (None, 1, 64)                386560    ['user_input[0][0]']

 movie_embedding (Embedding  (None, 1, 64)                237184    ['movie_input[0][0]']
 )

 FlattenUsers (Flatten)      (None, 64)                   0         ['user_embedding[0][0]']

 FlattenMovies (Flatten)     (None, 64)                   0         ['movie_embedding[0][0]']

 Simalarity-Dot-Product (Do  (None, 1)                    0         ['FlattenUsers[0][0]',
 t)                                                                  'FlattenMovies[0][0]']

==================================================================================================
Total params: 623744 (2.38 MB)
Trainable params: 623744 (2.38 MB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________________
```

```
# Model Training -


model_hist = model.fit([train.UserID, train.MovieID], train.Rating,
                       batch_size=128, epochs=20,
                       validation_data = ([valid.UserID, valid.MovieID], valid.Rating),
                       verbose=1)
```

```
Epoch 1/20
6252/6252 [==============================] - 86s 14ms/step - loss: 13.9668 - val_loss: 13.2739
Epoch 2/20
6252/6252 [==============================] - 74s 12ms/step - loss: 9.3569 - val_loss: 5.0552
Epoch 3/20
6252/6252 [==============================] - 62s 10ms/step - loss: 3.0838 - val_loss: 1.9997
Epoch 4/20
6252/6252 [==============================] - 62s 10ms/step - loss: 1.5474 - val_loss: 1.2788
Epoch 5/20
6252/6252 [==============================] - 64s 10ms/step - loss: 1.1140 - val_loss: 1.0332
Epoch 6/20
6252/6252 [==============================] - 62s 10ms/step - loss: 0.9551 - val_loss: 0.9337
Epoch 7/20
6252/6252 [==============================] - 66s 11ms/step - loss: 0.8874 - val_loss: 0.8886
Epoch 8/20
6252/6252 [==============================] - 67s 11ms/step - loss: 0.8558 - val_loss: 0.8668
Epoch 9/20
6252/6252 [==============================] - 67s 11ms/step - loss: 0.8393 - val_loss: 0.8550
Epoch 10/20
6252/6252 [==============================] - 66s 11ms/step - loss: 0.8292 - val_loss: 0.8476
Epoch 11/20
6252/6252 [==============================] - 61s 10ms/step - loss: 0.8221 - val_loss: 0.8418
Epoch 12/20
6252/6252 [==============================] - 65s 10ms/step - loss: 0.8159 - val_loss: 0.8370
Epoch 13/20
6252/6252 [==============================] - 69s 11ms/step - loss: 0.8102 - val_loss: 0.8326
Epoch 14/20
6252/6252 [==============================] - 65s 10ms/step - loss: 0.8043 - val_loss: 0.8280
Epoch 15/20
6252/6252 [==============================] - 67s 11ms/step - loss: 0.7983 - val_loss: 0.8235
Epoch 16/20
6252/6252 [==============================] - 64s 10ms/step - loss: 0.7920 - val_loss: 0.8185
Epoch 17/20
6252/6252 [==============================] - 66s 10ms/step - loss: 0.7854 - val_loss: 0.8143
Epoch 18/20
6252/6252 [==============================] - 65s 10ms/step - loss: 0.7785 - val_loss: 0.8093
Epoch 19/20
6252/6252 [==============================] - 60s 10ms/step - loss: 0.7714 - val_loss: 0.8043
Epoch 20/20
6252/6252 [==============================] - 61s 10ms/step - loss: 0.7642 - val_loss: 0.7997
```

```
# Model Evaluation -
y_pred = model.predict([valid.UserID, valid.MovieID], verbose=0)
```

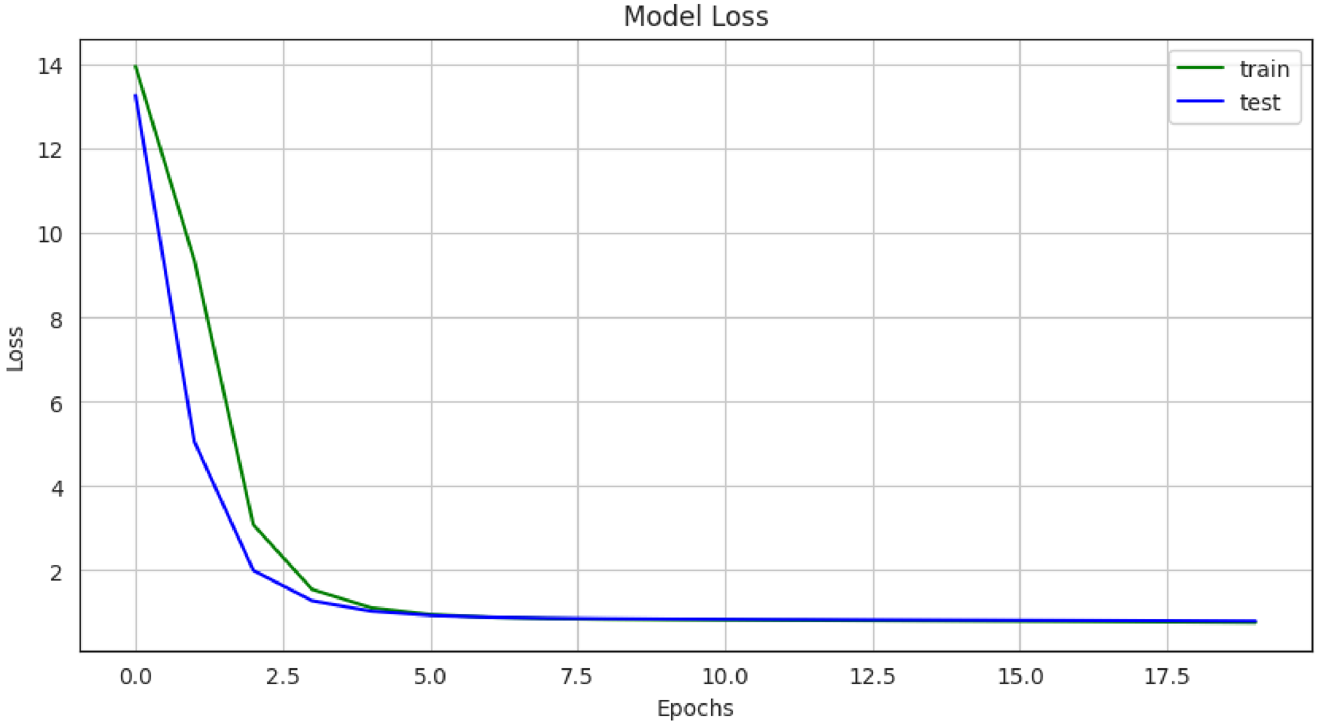
```
y_pred_class = np.argmax(y_pred, axis=-1)


# Calculating the RMSE -
from sklearn.metrics import mean_squared_error
rmse = mean_squared_error(valid.Rating, y_pred, squared=False)
print('Root Mean Squared Error: {:.3f}'.format(rmse))
```

```
Root Mean Squared Error: 0.894
```

```
# Calculating the MAPE -
from sklearn.metrics import mean_absolute_percentage_error
mape =  mean_absolute_percentage_error(valid.Rating, y_pred)
print('Mean Absolute Percentage Error: {:.3f}'.format(mape))
```

```
Mean Absolute Percentage Error: 0.275
```

```
# Plotting the Model Loss -
rcParams['figure.figsize'] = 10, 5
plt.plot(model_hist.history['loss'] , 'g')
plt.plot(model_hist.history['val_loss'] , 'b')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.legend(['train', 'test'], loc='upper right')
plt.grid(True)
plt.show()
```




```
mov_name = ['Mrs. Doubtfire', 'Dumb & Dumber', 'Ace Ventura: Pet Detective', 'Home Alone']
```

```
mov_id = []
for mov in mov_name:
    id = data[data['Title'] == mov]['MovieID'].iloc[0]
    mov_id.append(id)
```

```
mov_rating = list(map(int, input("Rate these movies respectively: ").split()))
```

```
Rate these movies respectively: 4 3 4 4
```

```
# Creating a dataframe for a new user's choices.
user_choices = pd.DataFrame({'MovieID': mov_id,
                             'Title': mov_name,
                             'Rating': mov_rating})
user_choices.sort_values(by='MovieID')
```

	MovieID	Title	Rating
2	316	Ace Ventura: Pet Detective	4
3	328	Home Alone	4
1	981	Dumb & Dumber	3

```
# Users who have watched the same movies as the new users.
other_users = data[data['MovieID'].isin(user_choices['MovieID'].values)]
other_users = other_users[['UserID', 'MovieID', 'Rating']]
other_users['UserID'].nunique()
```

```
1466
```

```
# Sorting old users by the count of most movies in common with the new user.
common_movies = other_users.groupby(['UserID'])
common_movies = sorted(common_movies, key=lambda x: len(x[1]), reverse=True)
common_movies[0]
```

```
(9,
       UserID  MovieID  Rating
 1687       9      981       1
 1705       9      316       1
 1738       9     1012       3
 1749       9      328       2)
```

```
top_users = common_movies[:100]
```

```
# Calculating a Similarity Score for each user using Pearson Correlation function -
pearson_corr = {}

for user_id, movies in top_users:
    movies = movies.sort_values(by='MovieID')
    movie_list = movies['MovieID'].values

    new_user_ratings = user_choices[user_choices['MovieID'].isin(movie_list)]['Rating'].values
    user_ratings = movies[movies['MovieID'].isin(movie_list)]['Rating'].values

    corr = pearsonr(new_user_ratings, user_ratings)
    pearson_corr[user_id] = corr[0]
```