Data Description:

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

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+"
- Occupation is chosen from the following choices:
  - 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"

Our Approach:

In this project, we'll be building a recommender system that is going to recommend movies to a user based on their preferences as well as the choices of other users who are similar to them.

What is a Recommender System?

A recommender engine, or a recommendation system is a subclass of information filtering system that seeks to predict the "rating" or "preference" a user would give to an item.

Types of Recommender Systems -

Recommender systems usually make use of either or both *Collaborative Filtering* and *Content-based Filtering* techniques.

Collaberative Filtering

Collaborative filtering is based on the assumption that people who agreed in the past will agree in the future, and that they will like similar kinds of items as they liked in the past. The system generates recommendations using only information about rating profiles for different users or items. By locating peer users/items with a rating history similar to the current user or item, they generate recommendations using this neighborhood.

Content-based Filtering

Content-based filtering methods are based on a description of the item and a profile of the user's preferences. These methods are best suited to situations where there is known data on an item (name, location, description, etc.), but not on the user. Content-based recommenders treat recommendation as a user-specific classification problem and learn a classifier for the user's likes and dislikes based on an item's features.

Table of Contents

1. Import Libraries

```
#!pip install cmfrec
#!pip uninstall scikit-surprise numpy
#!pip install numpy==1.23.5
#!pip install scikit-surprise

SEED=95

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
```

2. Set Options

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

3. Read Data and Data Formatting

3.1 Movies

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

```
(3883, 3)
```

{"summary":"{\n \"name\": \"movies\",\n \"rows\": 3883,\n \"fields\": [\n   {\n   \"column\": \"Movie ID::Title::Genres\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 3883,\n   \"samples\": [\n   \"1365::Ridicule (1996)::Drama\",\n   \"2706::American Pie (1999)::Comedy\",\n   \"3667::Rent-A-Cop (1988)::Action|Comedy\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Unnamed: 1\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 73,\n   \"samples\": [\n   \"|Sci\",\n   \"71):\",\n   \"er,\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Unnamed: 2\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 45,\n   \"samples\": [\n   \"Children's|Fan\",\n   \"(1975)::Comed\",\n   \"ler\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"movies"}

```
movies.drop(columns=['Unnamed: 1', 'Unnamed: 2'], axis=1,
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()
```

{"summary":"{\n \"name\": \"movies\",\n \"rows\": 3883,\n \"fields\": [\n   {\n   \"column\": \"MovieID\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 3883,\n   \"samples\": [\n   \"1365\",\n   \"2706\",\n   \"3667\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Title\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 3883,\

```
n        \"samples\": [\n          \"Ridicule (1996)\",\n
\"American Pie (1999)\",\n          \"Rent-A-Cop (1988)\"\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Genres\",\n      \"properties\":
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
360,\n        \"samples\": [\n          \"Action|Thriller|War\",\n
\"Crime\",\n          \"Action|Adventure|Sci-Fi|Thriller|War\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe","variable_name":"movies"}
```


3.2 Ratings

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

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

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

```
(1000209, 4)
```


```
{"type":"dataframe","variable_name":"ratings"}
```


3.3 Users

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

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

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)

print(users.shape)
users.head()
```

```
(6040, 5)
```

{"summary":"{\n \"name\": \"users\",\n \"rows\": 6040,\n \"fields\": [\n   {\n   \"column\": \"UserID\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 6040,\n   \"samples\": [\n   \"5530\",\n   \"711\",\n   \"4924\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Gender\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 2,\n   \"samples\": [\n   \"M\",\n   \"F\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Age\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 7,\n   \"samples\": [\n   \"Under 18\",\n   \"56 Above\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Occupation\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 21,\n   \"samples\": [\n   \"k-12 student\",\n   \"tradesman/craftsman\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Zip-code\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 3439,\n   \"samples\": [\n   \"02865\",\n   \"43213\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"users"}

4. Exploratory Data Analysis

4.1 Preparing the Dataset

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

# Remove year (including parentheses and any surrounding whitespace)
from Title
movies['Title'] = movies['Title'].str.replace(r'\s*\(\d{4}\)\s*', '',
regex=True)

# Strip any remaining whitespace
movies['Title'] = movies['Title'].str.strip()

# Display the result
movies.head()
```


{"summary":"{\n \"name\": \"movies\",\n \"rows\": 3883,\n
\"fields\": [\n   {\n   \"column\": \"MovieID\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 3883,\n   \"samples\": [\n
\"1365\",\n   \"2706\",\n   \"3667\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Title\",\n   \"properties\": {\
n   \"dtype\": \"string\",\n   \"num_unique_values\": 3841,\
n   \"samples\": [\n   \"Carmen Miranda: Bananas Is My
Business\",\n   \"Tigerland\",\n   \"Mifune (Mifunes
sidste sang)\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Genres\",\n   \"properties\": {\n   \"dtype\":
\"category\",\n   \"num_unique_values\": 360,\n
\"samples\": [\n   \"Action|Thriller|War\",\n
\"Crime\",\n   \"Action|Adventure|Sci-Fi|Thriller|War\"\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"Year\",\n   \"properties\":
{\n   \"dtype\": \"object\",\n   \"num_unique_values\": 81,\
n   \"samples\": [\n   \"1948\",\n   \"1995\",\n
\"1960\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe","variable_name":"movies"}


```
import pandas as pd

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()
```


{"summary":"{\n \"name\": \"dfmov\",\n \"rows\": 3858,\n
\"fields\": [\n   {\n   \"column\": \"MovieID\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 3858,\n   \"samples\": [\n
\"1927\",\n   \"1285\",\n   \"2746\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Title\",\n   \"properties\": {\
n   \"dtype\": \"string\",\n   \"num_unique_values\": 3816,\
n   \"samples\": [\n   \"Streetcar Named Desire, A\",\n
\"Devil and Max Devlin, The\",\n   \"Universal Soldier\"\n

```
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Genres\",\n      \"properties\":
{\n        \"dtype\": \"object\",\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Year\",\n      \"properties\": {\n        \"dtype\": \"object\",\n
\"num_unique_values\": 81,\n        \"samples\": [\n
\"1943\",\n          \"1995\",\n          \"1955\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"dfmov"}
```


4.1.1 Merge all above dataframes

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


```
{"type":"dataframe","variable_name":"df_1"}
```


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


```
{"type":"dataframe","variable_name":"data"}
```


Shape of the dataset

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

No. of rows:  996144
No. 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  Zip-code    996144 non-null  object
dtypes: object(11)
memory usage: 83.6+ MB
```

In this dataset we have **11 object columns**

4.1.2 Missing Values

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

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

{"summary":"{\n \"name\": \"missing_value\",\n \"rows\": 11,\n \"fields\": [\n   {\n   \"column\": \"Missing Value\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0,\n   \"min\": 0,\n   \"max\": 0,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Percentage\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.0,\n   \"min\": 0.0,\n   \"max\": 0.0,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   0.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

We have **no missing values** in the dataset

4.1.3 Feature Engineering

```
data['Datetime'] = pd.to_datetime(data['Timestamp'], unit='s') #Change the datatype from object to date_time
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  Zip-code    996144 non-null  object
 11  Datetime    996144 non-null  datetime64[ns]
dtypes: datetime64[ns](1), int32(2), object(9)
memory usage: 83.6+ MB
```

In the data we have **1 datetime, 2 integer and 9 object data type columns**

```
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()
```

{"type":"dataframe","variable_name":"data"}

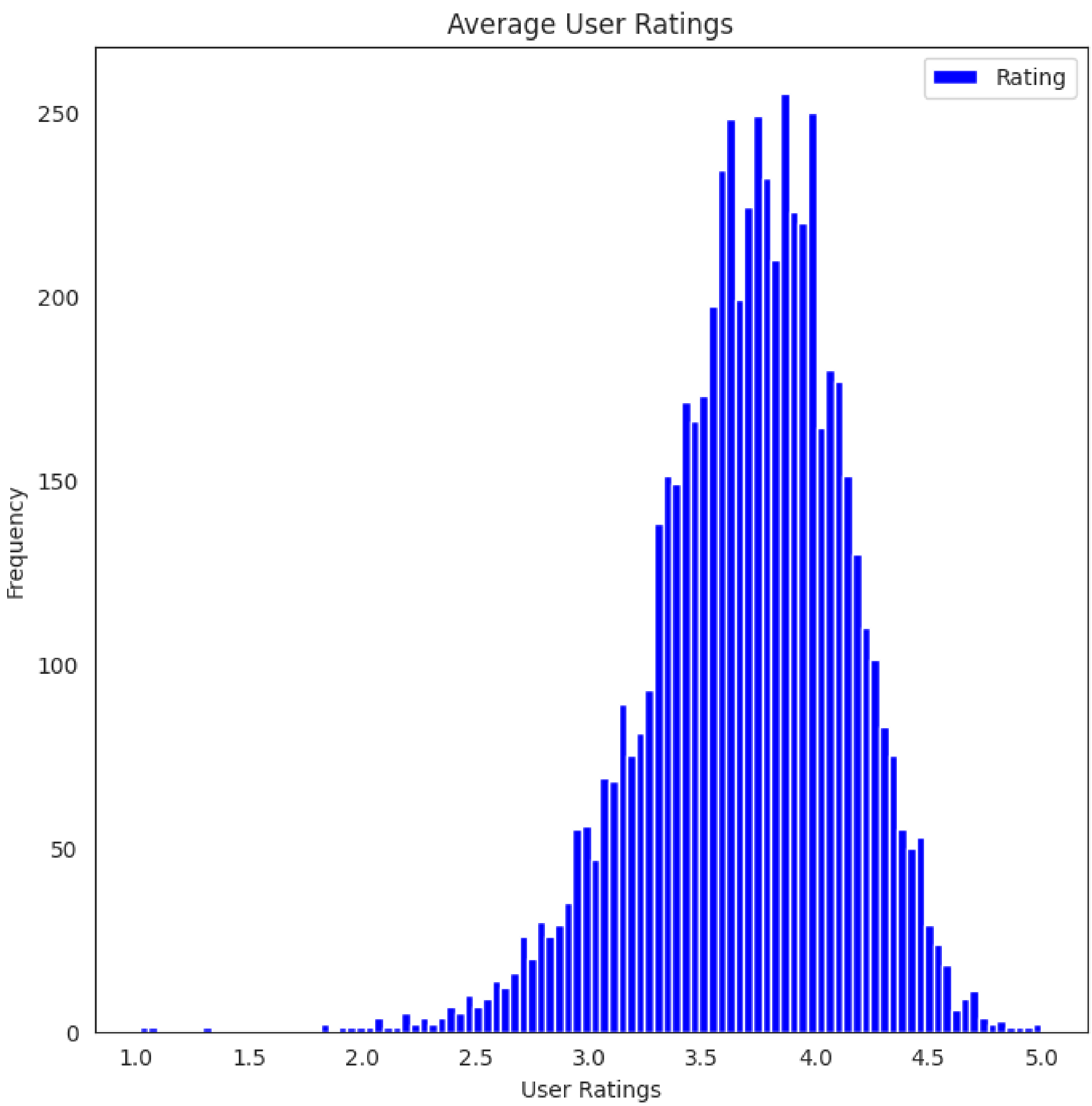
4.2 Understanding the Dataset

4.2.1 Analyse Features

Average User 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 =
'blue')
plt.plot()
plt.xlabel('User Ratings')
plt.title('Average User Ratings')
plt.ylabel('Frequency')
```

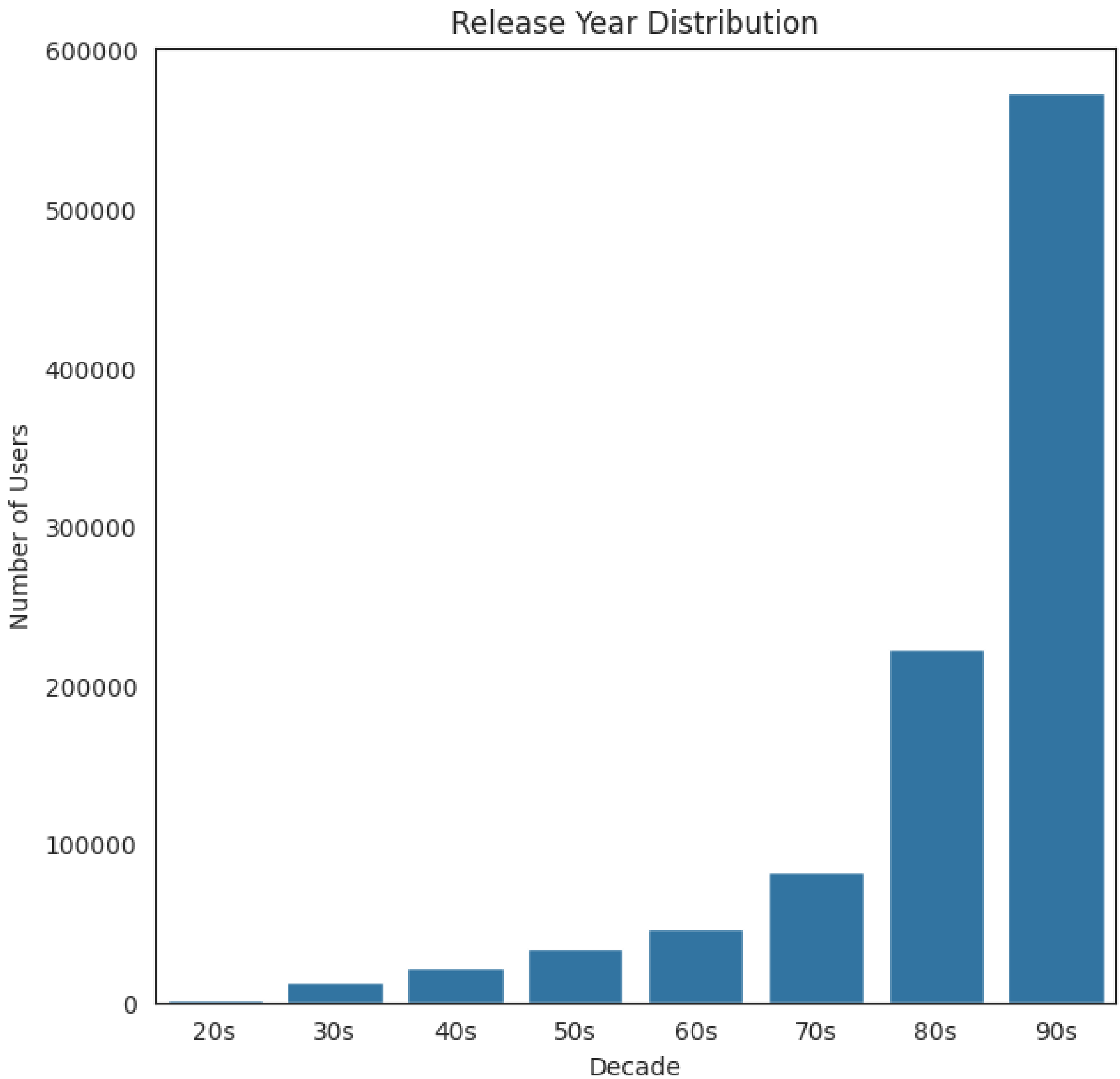
```
Text(0, 0.5, 'Frequency')

<Figure size 800x800 with 0 Axes>
```



From above plot, we can see that on average, users are rating movies **3.5-4** more frequently than any other rating. This makes sense since people are less inclined to rate movies lower than a 3 if they didn't enjoy the movie.

No.of movies 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()
```



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

```
l = dfmov.Genres.iloc[:5]

pd.get_dummies(l.apply(pd.Series).stack()).groupby(level=1).sum()
```

{"summary":"{\n \"name\": \"pd\",\n \"rows\": 3,\n \"fields\": [\n   {\n   \"column\": \"Adventure\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0,\n   \"min\": 0,\n   \"max\": 1,\n   \"num_unique_values\": 2,\n   \"samples\": [\n   0,\n   1\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Animation\",\n   \"properties\": {\n

\"dtype\": \"number\",\n   \"std\": 0,\n   \"min\": 0,\n
\"max\": 1,\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   0,\n   1\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"Children's\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 1,\n   \"min\": 0,\n
\"max\": 2,\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   2,\n   0\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"Comedy\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 1,\n   \"min\": 0,\n
\"max\": 3,\n   \"num_unique_values\": 3,\n   \"samples\":
[\n   3,\n   0\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"Drama\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0,\n   \"min\": 0,\n
\"max\": 1,\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   1,\n   0\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"Fantasy\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0,\n   \"min\": 0,\n
\"max\": 1,\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   1,\n   0\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"Romance\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0,\n   \"min\": 0,\n
\"max\": 1,\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   1,\n   0\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe"}

```
pd.Series(l.iloc[0])
```

```
0      Animation
1     Children's
2         Comedy
dtype: object
```

```
dfmov.head(2)
```

{"summary":"{\n \"name\": \"dfmov\",\n \"rows\": 3858,\n
\"fields\": [\n   {\n   \"column\": \"MovieID\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 3858,\n   \"samples\": [\n
\"1927\",\n   \"1285\",\n   \"2746\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Title\",\n   \"properties\": {\
n   \"dtype\": \"string\",\n   \"num_unique_values\": 3816,\
n   \"samples\": [\n   \"Streetcar Named Desire, A\",\n
\"Devil and Max Devlin, The\",\n   \"Universal Soldier\"\n

```
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Genres\",\n      \"properties\":
{\n        \"dtype\": \"object\",\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Year\",\n      \"properties\": {\n        \"dtype\": \"object\",\n
\"num_unique_values\": 81,\n        \"samples\": [\n
\"1943\",\n          \"1995\",\n          \"1955\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"dfmov"}
```


Top 10 Genres based on movies count

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


```
{"summary":"{\n  \"name\": \"genres_df\",\n  \"rows\": 3855,\n
\"fields\": [\n    {\n      \"column\": \"\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 0,\n
\"min\": 0,\n        \"max\": 1,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          1,\n          0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Action\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 0,\n
\"min\": 0,\n        \"max\": 1,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          1,\n          0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Adventure\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0,\n        \"min\": 0,\n        \"max\": 1,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          1,\n
0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Animation\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          0,\n          1\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Children's\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          0,\n          1\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Comedy\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          0,\n          1\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Crime\",\n      \"properties\": {\n        \"dtype\":
```

\"number\",\n   \"std\": 0,\n   \"min\": 0,\n
\"max\": 1,\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   1,\n   0\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"Documentary\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 0,\n   \"min\": 0,\n
\"max\": 1,\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   1,\n   0\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"Drama\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0,\n   \"min\": 0,\n
\"max\": 1,\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   1,\n   0\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"Fantasy\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0,\n   \"min\": 0,\n
\"max\": 1,\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   1,\n   0\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"Film-Noir\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 0,\n   \"min\": 0,\n
\"max\": 1,\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   1,\n   0\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"Horror\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0,\n   \"min\": 0,\n
\"max\": 1,\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   1,\n   0\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"Musical\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0,\n   \"min\": 0,\n
\"max\": 1,\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   1,\n   0\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"Mystery\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0,\n   \"min\": 0,\n
\"max\": 1,\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   1,\n   0\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"Romance\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0,\n   \"min\": 0,\n
\"max\": 1,\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   1,\n   0\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"Sci-Fi\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0,\n   \"min\": 0,\n
\"max\": 1,\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   1,\n   0\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n

```
\"column\": \"Thriller\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1,\n          0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"War\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1,\n          0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Western\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1,\n          0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"genres_df"}
```


```
### considering only the genre columns for the test
test = genres_df.iloc[:,0:].sum()
test=test.iloc[1:]
print(test)
```

```
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)
```

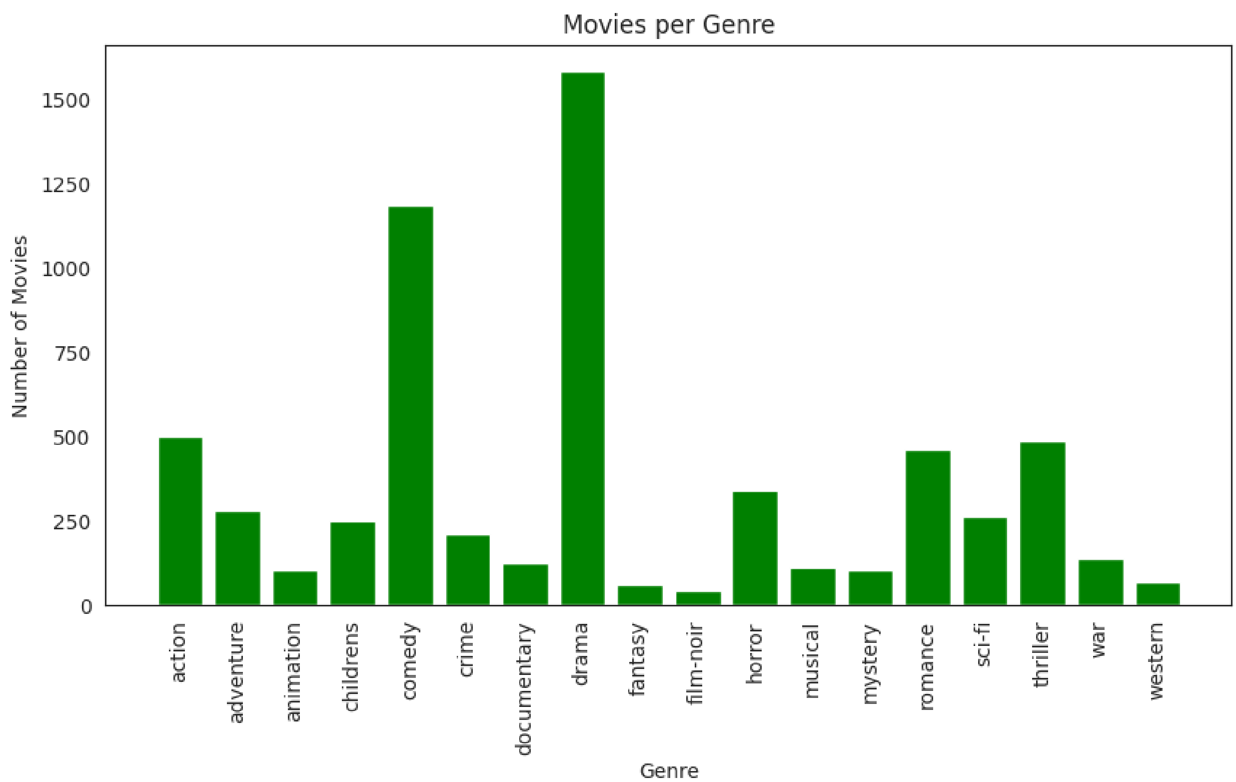
```
18
```

```
print(type(pd.to_numeric(test)))
print(type(test.to_numpy().reshape(18,)[0]))
# genre_sum = np.hstack((np.asarray(genre_list).reshape(18,1),
test.to_numpy().reshape(18,)))
```

```
# genre_sum[:,1] = genre_sum[:,1].astype('int64')
test2 = test.to_numpy().reshape(18,)
```

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

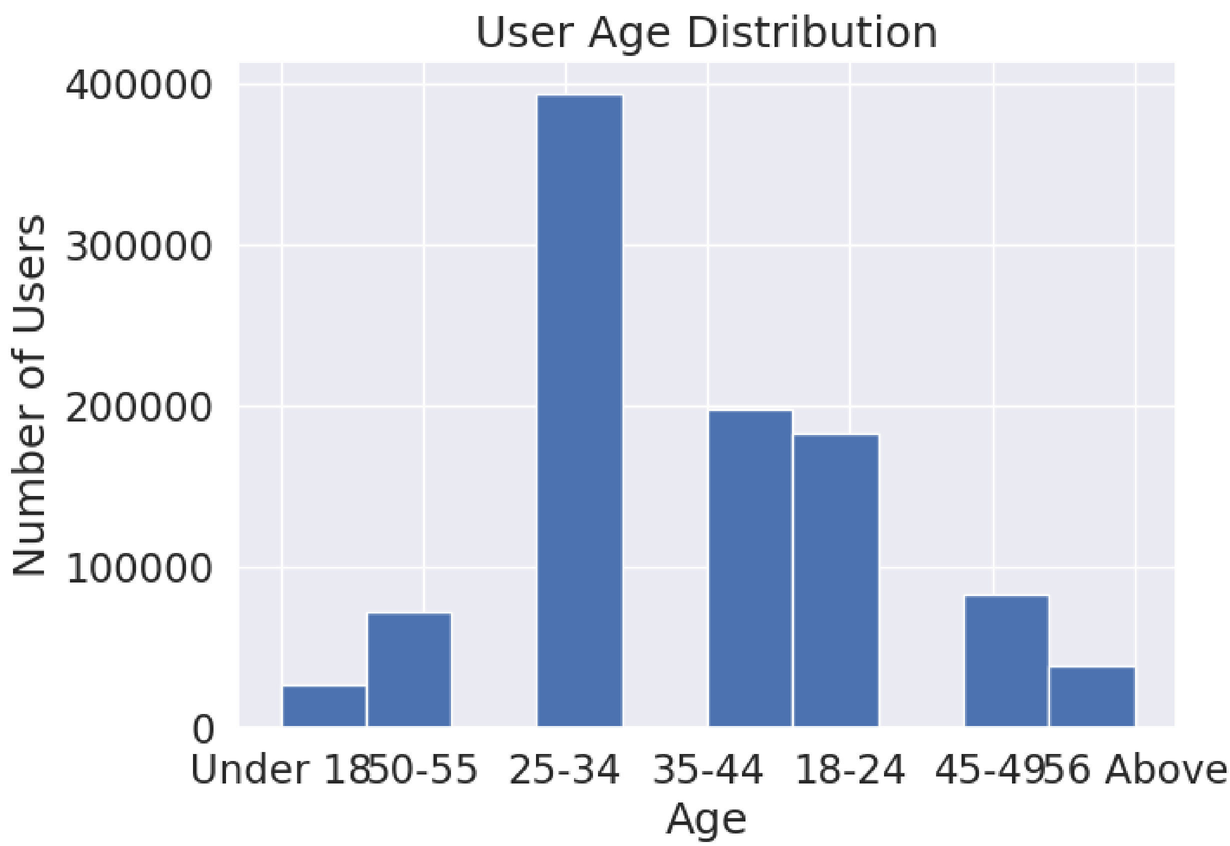
```
genre_list=['action', 'adventure','animation', 'childrens', 'comedy',
'crime', 'documentary', 'drama','fantasy', 'film-noir', 'horror',
'musical', 'mystery', 'romance', 'sci-fi','thriller', 'war',
'western']
x = np.arange(18)
plt.figure(figsize = (10,5))
plt.bar(x, test2, color = 'g')
plt.xticks(x, 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()
```



From the above plot we can infer that most the movies in the dataset belongs to **Comedy** and **Drama** genres.

Distribution by Age -

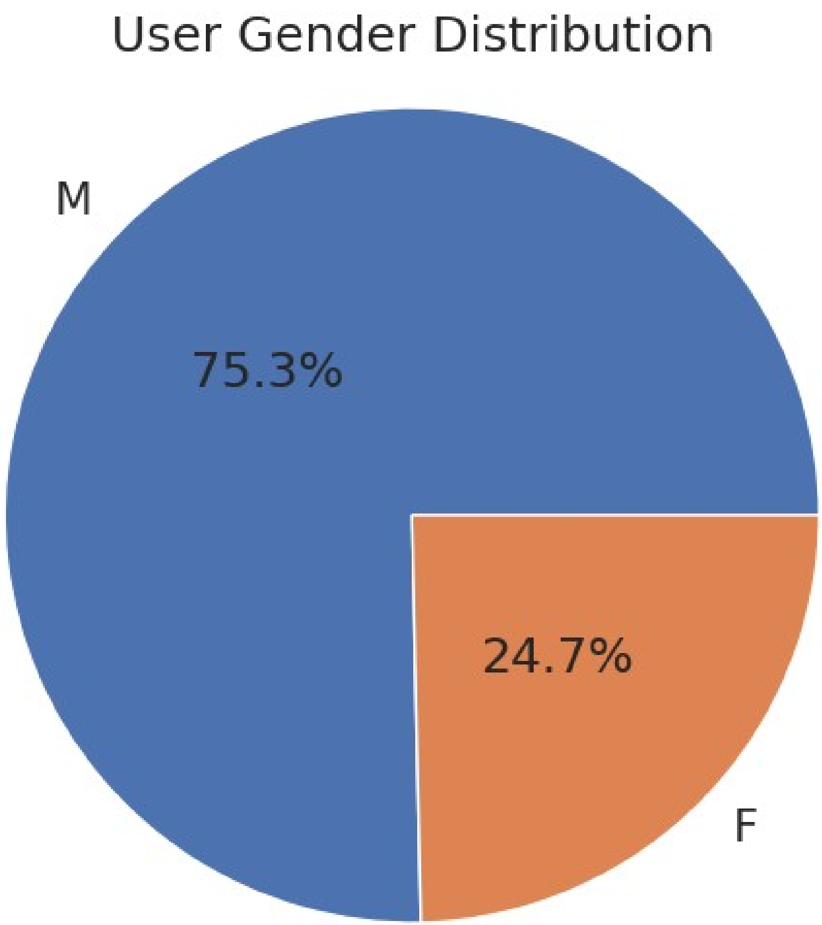
```
data['Age'].hist(figsize=(7, 5))
plt.title('User Age Distribution')
plt.xlabel('Age')
plt.ylabel('Number of Users')
plt.show()
```



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

Distribution by Gender -

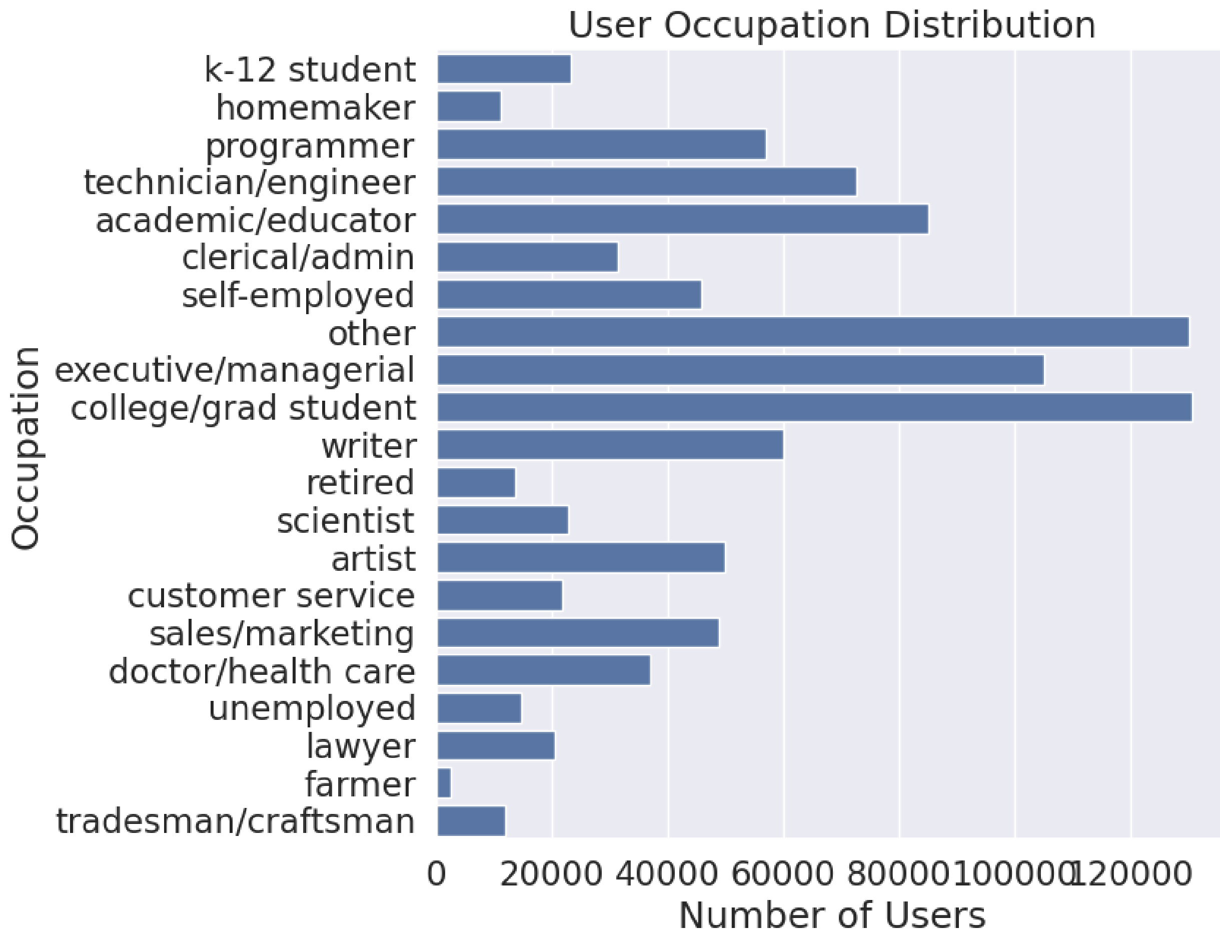
```
x = data['Gender'].value_counts().values
plt.figure(figsize=(7, 6))
plt.pie(x, center=(0, 0), radius=1.5, labels=['M','F'], autopct='%1.1f
%%', pctdistance=0.5)
plt.title('User Gender Distribution')
plt.axis('equal')
plt.show()
```



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

Distribution by Occupation -

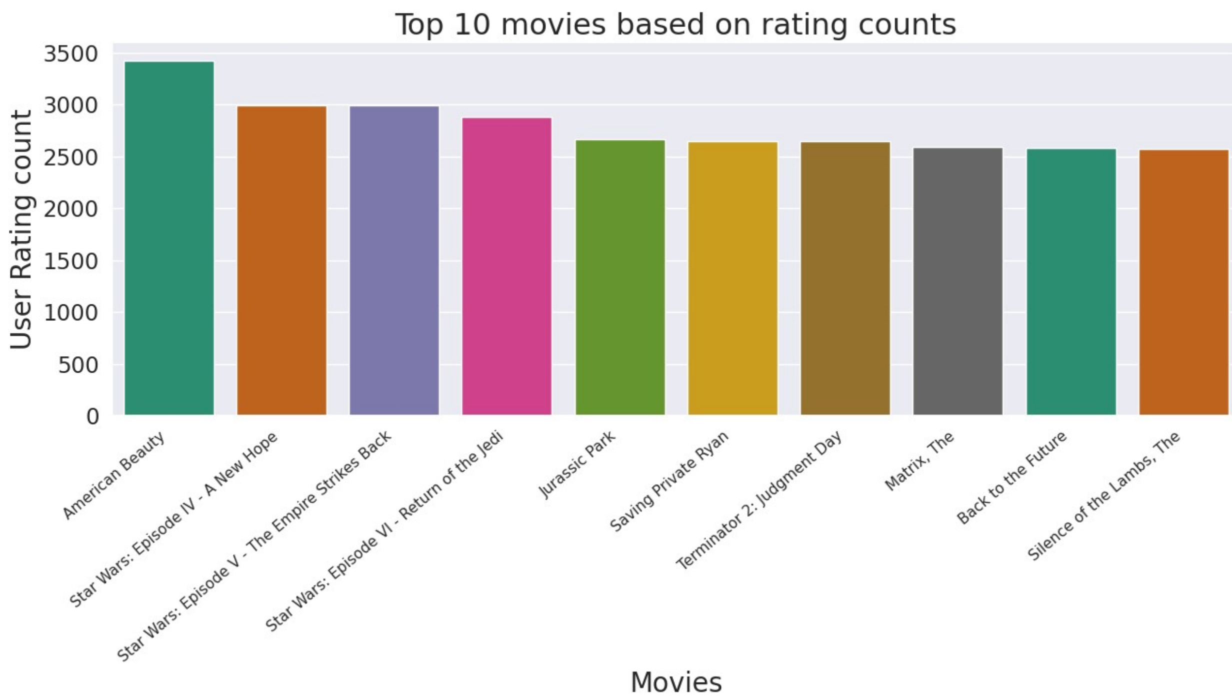
```
plt.figure(figsize=(7, 7))
sns.countplot(y='Occupation', data=data)
plt.title('User Occupation Distribution')
plt.xlabel('Number of Users')
plt.ylabel('Occupation')
plt.show()
```



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

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

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

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')
```



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

5. Recommendations 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)
```

{"type":"dataframe","variable_name":"matrix"}

```
# Checking 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)
```

```python
print('The sparsity level of dataset is ' +  str(sparsity * 100) +
'%')
```

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

5.1 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

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

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


```
{"summary":"{\n  \"name\": \"data[data['Title']=='Home Alone']\",\n
\"rows\": 675,\n  \"fields\": [\n    {\n      \"column\":
\"MovieID\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          \"586\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Title\",\n      \"properties\": {\n        \"dtype\": \"category\",\
n        \"num_unique_values\": 1,\n        \"samples\": [\n
\"Home Alone\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Genres\",\n      \"properties\": {\n        \"dtype\": \"object\",\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Year\",\n      \"properties\": {\n
\"dtype\": \"int32\",\n        \"num_unique_values\": 1,\n
\"samples\": [\n          1990\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"UserID\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 675,\n        \"samples\":
[\n          \"3630\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Rating\",\n      \"properties\": {\n        \"dtype\": \"int32\",\n
\"num_unique_values\": 5,\n        \"samples\": [\n          1\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
```

```
}\n    },\n    {\n      \"column\": \"Timestamp\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 675,\n        \"samples\": [\n
\"966539465\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Gender\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          \"M\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Age\",\n      \"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 7,\n        \"samples\": [\n          \"35-
44\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Occupation\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 21,\n
\"samples\": [\n          \"academic/educator\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Zip-code\",\n      \"properties\":
{\n        \"dtype\": \"string\",\n        \"num_unique_values\":
600,\n        \"samples\": [\n          \"10310\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Datetime\",\n      \"properties\":
{\n        \"dtype\": \"date\",\n        \"min\": \"2000-04-26
20:02:28\",\n        \"max\": \"2003-02-11 17:52:55\",\n
\"num_unique_values\": 675,\n        \"samples\": [\n          \"2000-
08-17 19:11:05\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ReleaseDec\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          \"90s\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


```
#movie_name = input("Enter a movie name: ")
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

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.
```


```
{"summary":"{\n  \"name\": \"sim_df\",\n  \"rows\": 5,\n  \"fields\":
[\n    {\n      \"column\": \"Title\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 5,\n
\"samples\": [\n          \"Mrs. Doubtfire\",\n          \"Sister
Act\",\n          \"Liar Liar\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Correlation\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.04282901229014881,\n        \"min\": 0.44461175615922277,\n
\"max\": 0.5472031329129602,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          0.4682810477824246,\n
0.44461175615922277,\n          0.45596650686479834\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


5.2 Cosine Similarty

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_similarity
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)
```

5.2.1 Item-Based Similarity

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

{"type":"dataframe","variable_name":"item_sim_matrix"}

5.2.2 User-Based Similarity

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

{"type":"dataframe","variable_name":"user_sim_matrix"}

5.3 Nearest Neighbors

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

```
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
```

{"summary":"{\n \"name\": \"#The result dataframe consits of the different indices of movies based on the distance\",\n \"rows\": 5,\n \"fields\": [\n   {\n   \"column\": \"Title1\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 1,\n   \"min\": 0,\n   \"max\": 4,\n   \"num_unique_values\": 5,\n   \"samples\": [\n   1,\n   4,\n   2\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Title2\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 639,\n   \"min\": 26,\n   \"max\": 1627,\n   \"num_unique_values\": 5,\n   \"samples\": [\n   807,\n   26,\n   1627\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Title3\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 1097,\n   \"min\": 72,\n   \"max\": 2529,\n   \"num_unique_values\": 5,\n   \"samples\": [\n   72,\n   726,\n   2529\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Title4\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 1182,\n   \"min\": 286,\n   \"max\": 3320,\n   \"num_unique_values\": 5,\n   \"samples\": [\n   2167,\n   894,\n   3320\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Title5\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 1181,\n   \"min\": 495,\n   \"max\": 3036,\n   \"num_unique_values\": 5,\n   \"samples\": [\n   3036,\n   495,\n   2588\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Title6\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 1111,\n   \"min\": 944,\n   \"max\": 3511,\n   \"num_unique_values\": 5,\n   \"samples\": [\n   3369,\n

944,\n   1999\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}


```
##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()
```


{"summary":"{\n \"name\": \"result2\",\n \"rows\": 3640,\n
\"fields\": [\n   {\n   \"column\": \"Title1\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 3607,\n   \"samples\": [\n
\"Spawn\",\n   \"Mr. Wrong\",\n   \"Man with the Golden
Gun, The\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Title2\",\n   \"properties\": {\n   \"dtype\":
\"category\",\n   \"num_unique_values\": 1815,\n
\"samples\": [\n   \"Porky's\",\n   \"Cheetah\",\n
\"Limbo\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Title3\",\n   \"properties\": {\n   \"dtype\":
\"category\",\n   \"num_unique_values\": 1763,\n
\"samples\": [\n   \"Blink\",\n   \"Airport '77\",\n
\"Arlington Road\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Title4\",\n   \"properties\": {\n   \"dtype\":
\"category\",\n   \"num_unique_values\": 1773,\n
\"samples\": [\n   \"Contempt (Le M\\u00e9pris)\",\n
\"Sex, Lies, and Videotape\",\n   \"Devil Girl From Mars\"\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"Title5\",\n   \"properties\":
{\n   \"dtype\": \"category\",\n   \"num_unique_values\":
1787,\n   \"samples\": [\n   \"Xiu Xiu: The Sent-Down Girl
(Tian yu)\",\n   \"Rocky IV\",\n   \"28 Days\"\
n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Title6\",\n   \"properties\": {\n   \"dtype\":
\"category\",\n   \"num_unique_values\": 1782,\n
\"samples\": [\n   \"Sleepy Hollow\",\n   \"Sliver\",\n
\"Blink\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe","variable_name":"result2"}

```
#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.
```


{"summary":"{\n \"name\": \"result2\",\n \"rows\": 1,\n \"fields\": [\n   {\n   \"column\": \"Title1\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 1,\n   \"samples\": [\n   \"Liar Liar\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Title2\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 1,\n   \"samples\": [\n   \"Mrs. Doubtfire\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Title3\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 1,\n   \"samples\": [\n   \"Ace Ventura: Pet Detective\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Title4\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 1,\n   \"samples\": [\n   \"Dumb & Dumber\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Title5\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 1,\n   \"samples\": [\n   \"Home Alone\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Title6\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 1,\n   \"samples\": [\n   \"Wayne's World\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

5.4 Matrix Factorization

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

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


{"type":"dataframe","variable_name":"rm"}

5.4.1 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)
```

```
{"type":"dataframe","variable_name":"user_itm"}
```

```
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
```

```
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 Items.
```

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

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

```
(3.57998542379415, 3.5799853801727295)
```

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

```
Root Mean Squared Error: 1.468
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
```

```
{"type":"dataframe","variable_name":"user_sim_matrix"}
```

```
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
```

{"type":"dataframe","variable_name":"itm_sim_matrix"}

```
movie_id='586'
movie_rating = itm_sim_matrix[movie_id] # 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.
```

{"summary":"{\n \"name\": \"sim_df\",\n \"rows\": 5,\n \"fields\":
[\n   {\n   \"column\": \"Correlation\",\n   \"properties\": {\
n   \"dtype\": \"number\",\n   \"std\":
0.0010979005942706333,\n   \"min\": 0.9963342542164175,\n
\"max\": 0.9992759377526796,\n   \"num_unique_values\": 5,\n
\"samples\": [\n   0.9978293972277665,\n
0.9963342542164175,\n   0.9971800200987978\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe"}

```
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()
```

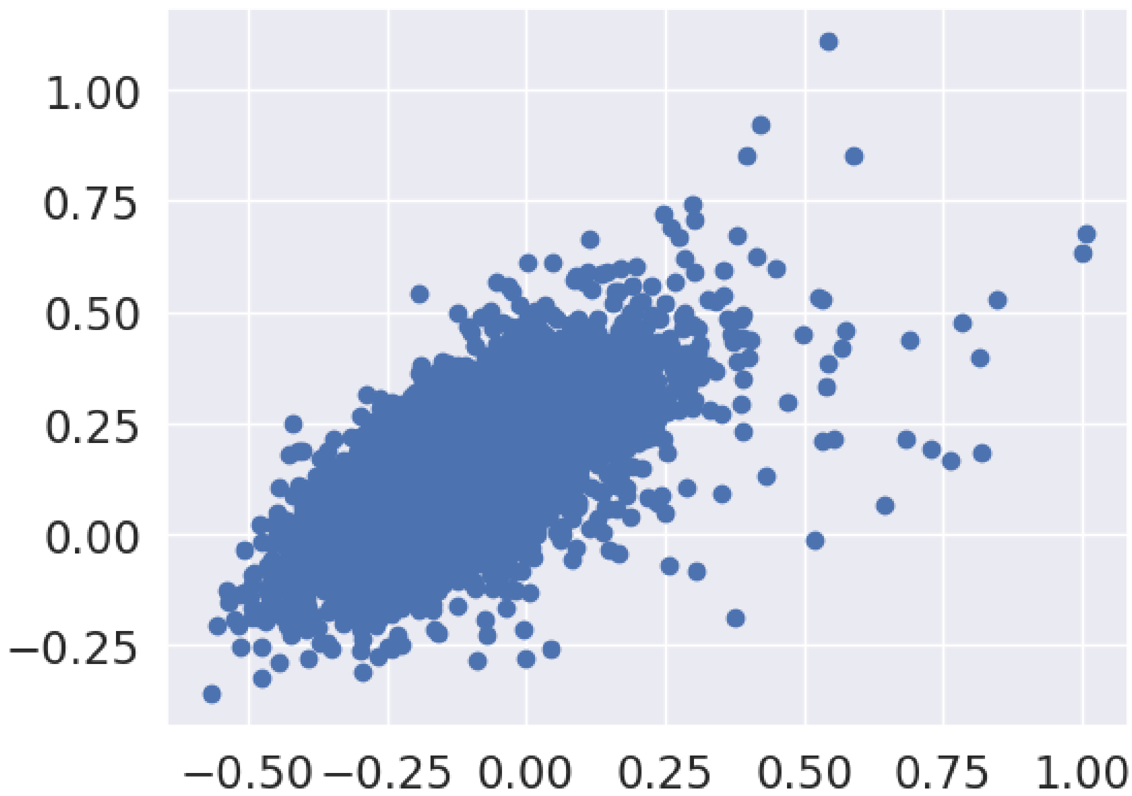
{"summary":"{\n \"name\": \"sim_mov\",\n \"rows\": 3682,\n \"fields\": [\n   {\n   \"column\": \"MovieID\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 3682,\n   \"samples\": [\n   \"2897\",\n   \"3902\",\n   \"2363\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Correlation\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.5356562435625112,\n   \"min\": -0.9630742003292803,\n   \"max\": 1.0,\n   \"num_unique_values\": 3658,\n   \"samples\": [\n   0.8669065868676699,\n   0.5156658550316853,\n   0.8911711923375352\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Title\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 3640,\n   \"samples\": [\n   \"Casper\",\n   \"Minnie and Moskowitz\",\n   \"Candyman\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"sim_mov"}

```
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)
```

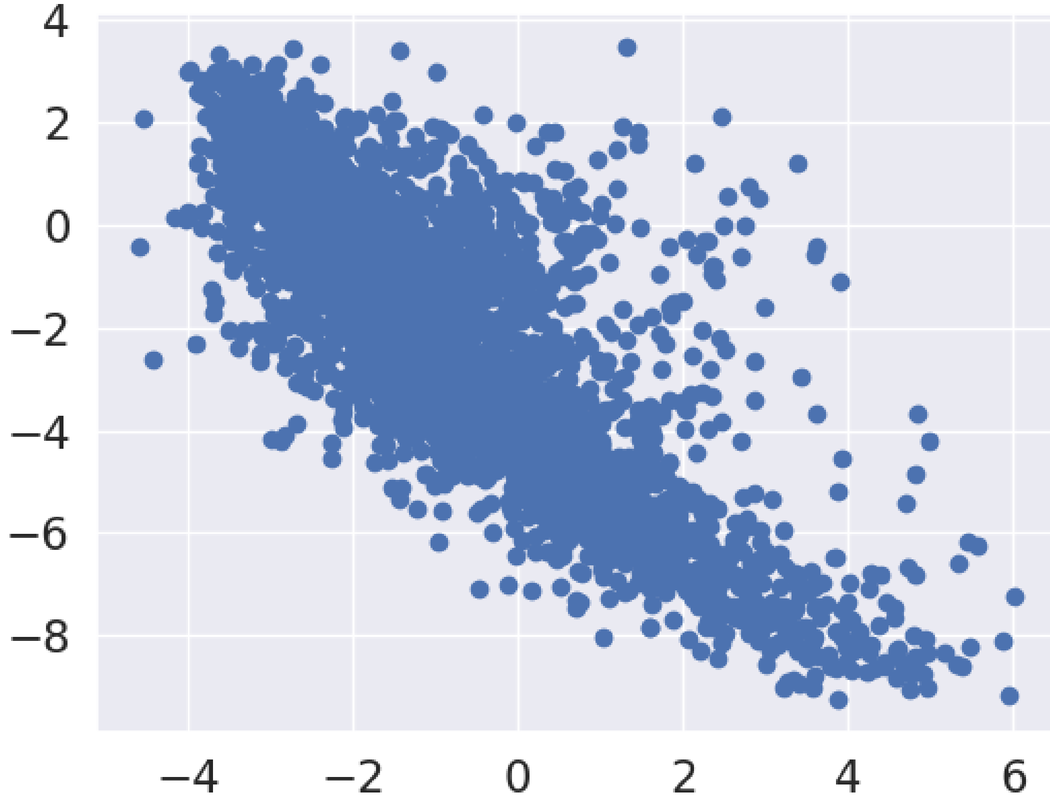
```
plt.scatter(model1.A_[:, 0], model1.A_[:, 1], cmap = 'hot')
```

```
<matplotlib.collections.PathCollection at 0x7b7bb270b490>
```




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

```
<matplotlib.collections.PathCollection at 0x7b7bb047a690>
```



5.4.2 Using Surprise Library

```
from surprise import Reader, SVD, Dataset
from surprise.model_selection import cross_validate

data.Rating.value_counts()

Rating
4    347758
3    260473
5    224639
2    107261
1     56013
Name: count, dtype: int64

##The Reader class is used to parse a file containing ratings.It
orders the data in format of (userid,title,rating) and even by
##considering the rating scale
user_itm = data[['UserID', 'Title', 'Rating']].copy()
reader = Reader(rating_scale=(1,5))
data1 = Dataset.load_from_df(user_itm[['UserID', 'Title', 'Rating']],
reader)
```

```
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
```

```
svd = SVD(n_factors=4)
cross_validate(svd, data1, measures=['rmse'], cv=3,
return_train_measures=True)
##The dataset is divided into train and test and with 3 folds the rmse
has been calculated
```

```
{'test_rmse': array([0.89218642, 0.89037035, 0.88951041]),
 'train_rmse': array([0.86280417, 0.86143625, 0.86048584]),
 'fit_time': (9.369915962219238, 10.91328740119934,
9.747197151184082),
 'test_time': (4.611389636993408, 3.3030149936676025,
3.9688572883605957)}
```

```
trainset = data1.build_full_trainset()
svd.fit(trainset) ##Fitting the trainset with the help of svd
```

```
<surprise.prediction_algorithms.matrix_factorization.SVD at
0x7b7bb01d71d0>
```

```
svd.pu
```

```
array([[ 0.31096954, -0.0414775 ,  0.0594095 , -0.14657091],
       [ 0.3594236 , -0.27538149, -0.23155216,  0.08617026],
       [-0.33449153, -0.05642853, -0.09473366,  0.00368887],
       ...,
       [-0.29126092,  0.14094672,  0.06545413, -0.03006094],
       [ 0.00132793, -0.09006503, -0.27691897,  0.22701811],
       [ 0.07752775, -0.17400341, -0.05592384, -0.16945732]])
```

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

```
((6040, 4), (3640, 4))
```

```
#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)
```

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

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


{"summary":"{\n \"name\": \"a[0:10] ##TOP 10\",\n \"rows\": 10,\n
\"fields\": [\n   {\n   \"column\": \"uid\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
0,\n   \"min\": 662,\n   \"max\": 662,\n
\"num_unique_values\": 1,\n   \"samples\": [\n   662\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"iid\",\n   \"properties\": {\
n   \"dtype\": \"string\",\n   \"num_unique_values\": 10,\n
\"samples\": [\n   \"Casablanca\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"r_ui\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 0,\n   \"min\": 4,\n
\"max\": 4,\n   \"num_unique_values\": 1,\n   \"samples\":
[\n   4\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"est\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 0.06863946738286657,\n   \"min\": 4.481828309873296,\n
\"max\": 4.716669220931391,\n   \"num_unique_values\": 10,\n
\"samples\": [\n   4.4823428669493115\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"details\",\n   \"properties\":
{\n   \"dtype\": \"object\",\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}


```
testset = trainset.build_anti_testset()

predictions_svd = svd.test(testset)

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

SVD - RMSE: 0.7033573186261063
SVD - MAE: 0.5442049960503313
```

Embeddings for user-user similarity using surprise library.

```
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
```


{"type":"dataframe","variable_name":"user_sim_matrix"}

Embeddings for item-item similarity using surprise library.

```
itm=cosine_similarity(svd.qi)

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


```
{"type":"dataframe","variable_name":"itm_sim_matrix"}
```


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

```
0      -0.25
1       0.77
2       0.86
3       0.20
4       0.81
        ...
3635    0.37
3636   -0.96
3637   -0.74
3638   -0.71
3639   -0.45
Name: 586, Length: 3640, dtype: float64
```

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


```
{"summary":"{\n  \"name\": \"sim_df\",\n  \"rows\": 5,\n  \"fields\":
[\n    {\n      \"column\": \"Correlation\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\":
0.0028335765633633204,\n        \"min\": 0.9933107309659163,\n
\"max\": 1.0,\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          0.9978090096682464,\n          0.9933107309659163,\n
0.9966903462351658\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```

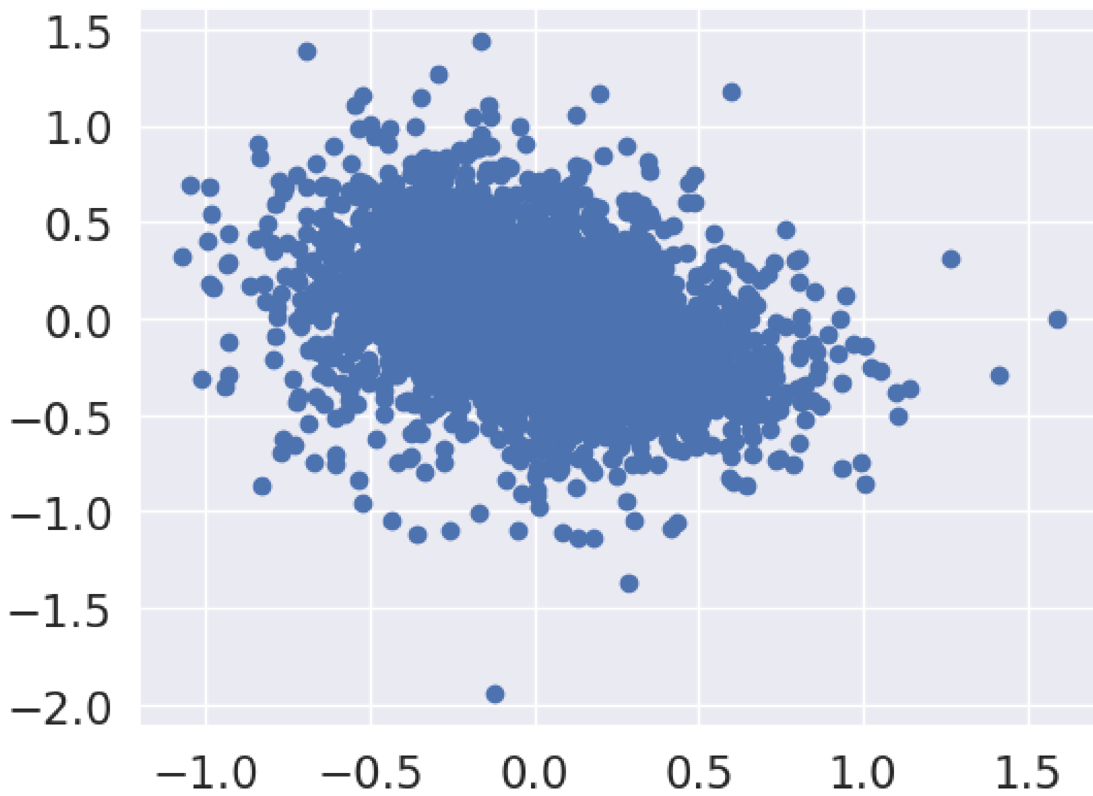

```
item_mov = data[['MovieID', 'Title']].copy()
item_mov.drop_duplicates(inplace=True)
item_mov.reset_index(drop=True,inplace=True)
item_mov.MovieID=item_mov.MovieID.astype('str')

sim_df1= sim_df.copy()
```

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

{"summary":"{\n \"name\": \"sim_mov\",\n \"rows\": 3379,\n \"fields\": [\n   {\n   \"column\": \"MovieID\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 3379,\n   \"samples\": [\n   \"83\",\n   \"227\",\n   \"233\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Correlation\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.678953775733866,\n   \"min\": -0.9973991036302694,\n   \"max\": 1.0,\n   \"num_unique_values\": 3379,\n   \"samples\": [\n   0.8903697526363363,\n   0.9102241075628142,\n   0.8754502909640419\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Title\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 3343,\n   \"samples\": [\n   \"Ridicule\",\n   \"Cable Guy, The\",\n   \"Twilight\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"sim_mov"}

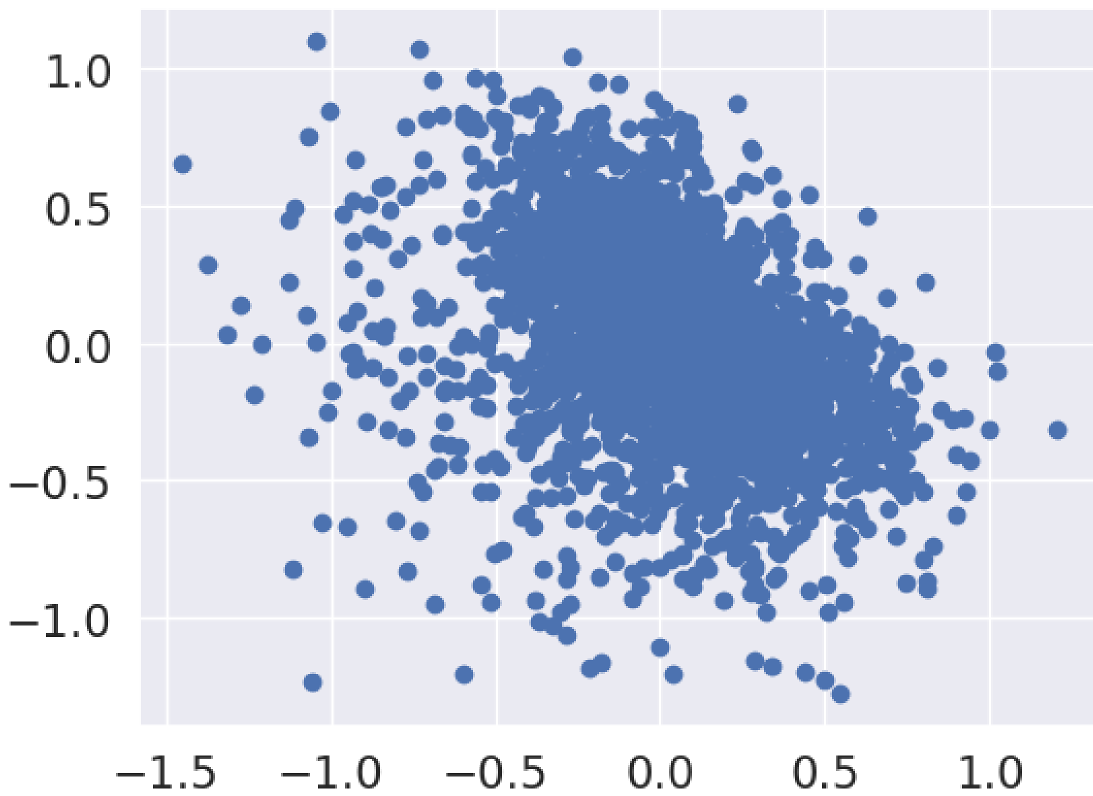
```
plt.scatter(svd.pu[:, 0], svd.pu[:, 1], cmap = 'hot')
```

```
<matplotlib.collections.PathCollection at 0x7b7bb10a0d10>
```




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

```
<matplotlib.collections.PathCollection at 0x7b79465d0790>
```



5.5 User-Based Approach(optional)

```
#Taking 6 movies names in random
mov_name = ['Hamlet', 'Dumb & Dumber', 'Ace Ventura: Pet Detective',
'Home Alone','Robin Hood','It Happened One Night']

#Finding the MovieID's for the above movies
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()))
mov_rating = [5,3,2,1,4,3]#Give the random user rating for the movies

user_choices = pd.DataFrame({'MovieID': mov_id, 'Title': mov_name,
'Rating': mov_rating})
user_choices.sort_values(by='MovieID') #User choices
```


```
{"summary":"{\n  \"name\": \"user_choices\",\n  \"rows\": 6,\n
\"fields\": [\n    {\n      \"column\": \"MovieID\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 6,\n        \"samples\": [\n
\"1411\",\n          \"231\",\n          \"905\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
```

n   },\n   {\n   \"column\": \"Title\",\n   \"properties\": {\
n   \"dtype\": \"string\",\n   \"num_unique_values\": 6,\n
\"samples\": [\n   \"Hamlet\",\n   \"Dumb & Dumber\",\n
\"It Happened One Night\"\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"Rating\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 1,\n   \"min\": 1,\n
\"max\": 5,\n   \"num_unique_values\": 5,\n   \"samples\":
[\n   3,\n   1,\n   4\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe"}

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

```
1810
```

```
common_movies = other_users.groupby(['UserID']) #Grouping the data
based on User who watched the common movies
common_movies = sorted(common_movies, key=lambda x: len(x[1]),
reverse=True) #Soring the data so that who watched more number of
common movies comes at the top.
common_movies[0]
```

```
(('1605',),
        UserID MovieID  Rating
 60682    1605     231       2
 92082    1605     344       3
 156207   1605     586       3
 214130   1605     905       4
 416462   1605    1411       3
 811059   1605    3034       4)
```

```
top_users = common_movies[:100] #Taking top 100 users who watched same
movies as in user choices.

#Calculating pearson correlation
pearson_corr = {}

for user_id, movies in top_users:
    movies = movies.sort_values(by='MovieID')
    movie_list = movies['MovieID'].values #Taking list of movieid's

    new_user_ratings =
user_choices[user_choices['MovieID'].isin(movie_list)]
['Rating'].values # Taking the new user rating values based on user
choices
    user_ratings = movies[movies['MovieID'].isin(movie_list)]
```

```
['Rating'].values #Taking the actual rating values of the movies

    corr = pearsonr(new_user_ratings, user_ratings) # Calculating the
correlation
    pearson_corr[user_id] = corr[0] #Correlation value for each UserID

pearson_df = pd.DataFrame(
    columns=['UserID', 'Similarity Index'],
    data=[(int(user_id[0]), corr) for user_id, corr in
pearson_corr.items()]
)
pearson_df = pearson_df.sort_values(by='Similarity Index',
ascending=False)[:10]
pearson_df['UserID'] = pearson_df['UserID'].astype('str')
pearson_df
```

{"summary":"{\n \"name\": \"pearson_df\",\n \"rows\": 10,\n \"fields\": [\n   {\n   \"column\": \"UserID\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 10,\n   \"samples\": [\n   \"4016\",\n   \"2109\",\n   \"2288\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Similarity Index\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.0688131333513843,\n   \"min\": 0.7559289460184545,\n   \"max\": 0.9683296637314887,\n   \"num_unique_values\": 9,\n   \"samples\": [\n   0.8000946913656627,\n   0.9438798074485388,\n   0.8660254037844385\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"pearson_df"}

```
users_rating = pearson_df.merge(data, on='UserID', how='inner')
#Merging the original data with pearson correlation values
users_rating['Weighted Rating'] = users_rating['Rating'] *
users_rating['Similarity Index'] # Calculating the Weighed rating for
each user and movie
users_rating = users_rating[['UserID', 'MovieID', 'Rating',
'Similarity Index', 'Weighted Rating']]
users_rating
```

{"summary":"{\n \"name\": \"users_rating\",\n \"rows\": 7400,\n \"fields\": [\n   {\n   \"column\": \"UserID\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 10,\n   \"samples\": [\n   \"4016\",\n   \"2109\",\n   \"2288\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"MovieID\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 2359,\n   \"samples\": [\n   \"2107\",\n   \"180\",\n   \"2946\"\n   ],\n   \"semantic_type\":

```
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"Rating\",\n      \"properties\": {\n        \"dtype\": \"int32\",\n        \"num_unique_values\": 5,\n        \"samples\": [\n          3,\n          1,\n          4\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"Similarity Index\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.06247139700621804,\n        \"min\": 0.7559289460184545,\n        \"max\": 0.9683296637314887,\n        \"num_unique_values\": 9,\n        \"samples\": [\n          0.8000946913656627,\n          0.9438798074485388,\n          0.8660254037844385\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"Weighted Rating\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 1.087881109686731,\n        \"min\": 0.7559289460184545,\n        \"max\": 4.841648318657443,\n        \"num_unique_values\": 45,\n        \"samples\": [\n          1.6001893827313254,\n          3.464101615137754,\n          2.5980762113533156\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe","variable_name":"users_rating"}
```


```
# Calculate sum of similarity index and weighted rating for each movie
grouped_ratings = users_rating.groupby('MovieID').sum()[['Similarity
Index', 'Weighted Rating']]

recommend_movies = pd.DataFrame()

# Add average recommendation score.
# We're calculating average recommendation score by dividing the
Weighted Rating by the Similarity Index.
recommend_movies['avg_reccomend_score'] = grouped_ratings['Weighted
Rating']/grouped_ratings['Similarity Index']
recommend_movies['MovieID'] = grouped_ratings.index
recommend_movies = recommend_movies.reset_index(drop=True)

# Select movies with the highest score i.e. 5
recommend_movies =
recommend_movies[(recommend_movies['avg_reccomend_score'] == 5)]

recommendations =
data[data['MovieID'].isin(recommend_movies['MovieID'])][['MovieID',
'Title']].sample(10)
recommendations
```


```
{"summary":"{\n  \"name\": \"recommendations\",\n  \"rows\": 10,\n  \"fields\": [\n    {\n      \"column\": \"MovieID\",\n      \"properties\": {\n        \"dtype\": \"string\",\n        \"num_unique_values\": 10,\n        \"samples\": [\n          \"3330\",\n          \"2874\",\n          \"3653\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
```

```
n    },\n    {\n      \"column\": \"Title\",\n      \"properties\": {\
n        \"dtype\": \"string\",\n        \"num_unique_values\": 10,\n
\"samples\": [\n          \"Splendor in the Grass\",\n
\"Pajama Game, The\",\n          \"Endless Summer, The\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"recommendations"}
```


5.6 Regression Based Rec Sys

```
from sklearn.preprocessing import StandardScaler

movies1.head()
```


```
{"summary":"{\n  \"name\": \"movies1\",\n  \"rows\": 3883,\n
\"fields\": [\n    {\n      \"column\": \"MovieID\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 3883,\n        \"samples\": [\n
\"1365\",\n          \"2706\",\n          \"3667\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Title\",\n      \"properties\": {\
n        \"dtype\": \"string\",\n        \"num_unique_values\": 3883,\
n        \"samples\": [\n          \"Ridicule (1996)\",\n
\"American Pie (1999)\",\n          \"Rent-A-Cop (1988)\"\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Genres\",\n      \"properties\":
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
360,\n        \"samples\": [\n          \"Action|Thriller|War\",\n
\"Crime\",\n          \"Action|Adventure|Sci-Fi|Thriller|War\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe","variable_name":"movies1"}
```


```
ratings1.head()
```


```
{"type":"dataframe","variable_name":"ratings1"}
```


```
users1.head()
```


```
{"summary":"{\n  \"name\": \"users1\",\n  \"rows\": 6040,\n
\"fields\": [\n    {\n      \"column\": \"UserID\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 6040,\n        \"samples\": [\n
\"5530\",\n          \"711\",\n          \"4924\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Gender\",\n      \"properties\":
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
2,\n        \"samples\": [\n          \"M\",\n          \"F\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Age\",\n      \"properties\": {\
n        \"dtype\": \"category\",\n        \"num_unique_values\": 7,\n
\"samples\": [\n          \"1\",\n          \"56\"\n        ],\n
```

```
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Occupation\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 21,\n        \"samples\": [\n          \"10\",\
n          \"18\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Zip-code\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 3439,\n
\"samples\": [\n          \"02865\",\n          \"43213\"\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe","variable_name":"users1"}
```


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


```
{"summary":"{\n  \"name\": \"m\",\n  \"rows\": 3883,\n  \"fields\": [\
n    {\n      \"column\": \"MovieID\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 3883,\n
\"samples\": [\n          \"1365\",\n          \"2706\",\n
\"3667\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Action\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.33630424227775096,\n        \"min\": 0.0,\n        \"max\":
1.0,\n        \"num_unique_values\": 2,\n        \"samples\": [\n
1.0,\n          0.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Adventure\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.2604192092818326,\n        \"min\":
0.0,\n        \"max\": 1.0,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          1.0,\n          0.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Animation\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.16203997047201232,\n        \"min\": 0.0,\n        \"max\": 1.0,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          0.0,\n
1.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Children's\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.24583543818318188,\n        \"min\":
0.0,\n        \"max\": 1.0,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          0.0,\n          1.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Comedy\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.4619053085430115,\n        \"min\": 0.0,\n        \"max\": 1.0,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          0.0,\n
1.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Crime\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.22698145067998357,\n        \"min\": 0.0,\n        \"max\":
```

```
1.0,\n        \"num_unique_values\": 2,\n        \"samples\": [\n
1.0,\n          0.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Documentary\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.17646371613063716,\n        \"min\":
0.0,\n        \"max\": 1.0,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          1.0,\n          0.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Drama\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 0.4919657978871249,\
n        \"min\": 0.0,\n        \"max\": 1.0,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          1.0,\n
0.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Fantasy\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 0.1258111753854382,\n        \"min\": 0.0,\n
\"max\": 1.0,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1.0,\n          0.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Film-Noir\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.10623748141562311,\n        \"min\": 0.0,\n        \"max\": 1.0,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          1.0,\n
0.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Horror\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.28361818721752075,\n        \"min\": 0.0,\n        \"max\":
1.0,\n        \"num_unique_values\": 2,\n        \"samples\": [\n
1.0,\n          0.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Musical\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 0.16870310153036064,\n        \"min\": 0.0,\n
\"max\": 1.0,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1.0,\n          0.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Mystery\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.16279544044103164,\n        \"min\": 0.0,\n        \"max\": 1.0,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          1.0,\n
0.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Romance\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 0.32482157755153546,\n        \"min\": 0.0,\n
\"max\": 1.0,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1.0,\n          0.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Sci-Fi\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.25304753722068224,\n        \"min\": 0.0,\n        \"max\": 1.0,\n
```

\"num_unique_values\": 2,\n   \"samples\": [\n   1.0,\n   0.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Thriller\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.33255495479013647,\n   \"min\": 0.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 2,\n   \"samples\": [\n   1.0,\n   0.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"War\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.18645636204424262,\n   \"min\": 0.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 2,\n   \"samples\": [\n   1.0,\n   0.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Western\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.1316540011798341,\n   \"min\": 0.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 2,\n   \"samples\": [\n   1.0,\n   0.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"m"}

```
from datetime import datetime
r = ratings1.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()
```

{"type":"dataframe","variable_name":"r"}

```
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)
```

{"summary":"{\n \"name\": \"users2\",\n \"rows\": 6040,\n \"fields\": [\n   {\n   \"column\": \"UserID\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 6040,\n   \"samples\": [\n   \"5530\",\n   \"711\",\n   \"4924\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Gender\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 2,\n   \"samples\": [\n   \"M\",\n   \"F\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Age\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 7,\n   \"samples\": [\n   \"1\",\n   \"56\"\n   ],\n

```
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Occupation\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 21,\n        \"samples\": [\n          \"10\",\
n          \"18\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Zip-code\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 3439,\n
\"samples\": [\n          \"02865\",\n          \"43213\"\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Rating\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.4296220822812284,\n        \"min\": 1.0153846153846153,\n
\"max\": 4.962962962962963,\n        \"num_unique_values\": 4014,\n
\"samples\": [\n          3.650557620817844,\n
3.6033333333333335\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"hour\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 7.1406577581328134,\n        \"min\": 0.0,\n        \"max\":
23.0,\n        \"num_unique_values\": 4079,\n        \"samples\": [\n
22.98,\n          19.06153846153846\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"users2"}
```


```
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)
```


```
{"summary":"{\n  \"name\": \"u\",\n  \"rows\": 6040,\n  \"fields\": [\
n    {\n      \"column\": \"UserID\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 6040,\n
\"samples\": [\n          \"5530\",\n          \"711\",\n
\"4924\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Age\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1.0000827917375172,\n        \"min\": -2.298525144720067,\n
\"max\": 1.966728948799723,\n        \"num_unique_values\": 7,\n
\"samples\": [\n          -2.298525144720067,\n
1.966728948799723,\n          0.33817738581943946\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"User_avg_rating\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.0000827917375272,\n        \"min\": -6.255597303699077,\n
\"max\": 2.9336537550127875,\n        \"num_unique_values\": 4014,\n
\"samples\": [\n          -0.12138939243085863,\n          -
```

0.23131902410749441,\n   1.1417419607868535\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"hour\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 1.0000827917375206,\n
\"min\": -1.7010165981870098,\n   \"max\": 1.520241861338681,\n
\"num_unique_values\": 4079,\n   \"samples\": [\n
1.5174407670260501,\n   0.9686417505436464,\n   -
1.3999364073641014\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe","variable_name":"u"}

```
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()
```

{"summary":"{\n \"name\": \"df_cat\",\n \"rows\": 6040,\n
\"fields\": [\n   {\n   \"column\": \"UserID\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 6040,\n   \"samples\": [\n
\"5530\",\n   \"711\",\n   \"4924\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Gender\",\n   \"properties\":
{\n   \"dtype\": \"boolean\",\n   \"num_unique_values\": 2,\
n   \"samples\": [\n   true,\n   false\n   ],\
n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"Occupation\",\n
\"properties\": {\n   \"dtype\": \"category\",\n
\"num_unique_values\": 21,\n   \"samples\": [\n   \"10\",\
n   \"18\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe","variable_name":"df_cat"}

```
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()
```

{"type":"dataframe","variable_name":"X"}

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

```python
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=SEED)

from sklearn.ensemble import GradientBoostingRegressor

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

rmse = np.sqrt(mean_squared_error(y_test, y_pred)) # calculating rmse value
print('Root Mean Squared Error: {:.3f}'.format(rmse))
```

```
Root Mean Squared Error: 1.007
```

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

```
Mean Absolute Percentage Error: 0.323
```

5.7 Ensemble Recommender System

```python
X1.head()
```

{"type":"dataframe","variable_name":"X1"}

```python
y = X1.pop('Rating')

X_train, X_test, y_train, y_test = train_test_split(X1, y, test_size=0.2, random_state=SEED)

X_train1 = X_train[['MovieID', 'UserID']].copy()
X_train = X_train.drop(columns = ['MovieID', 'UserID'])
user_item_train=pd.concat([X_train1,y_train],axis=1)

X_test1 = X_test[['MovieID', 'UserID']].copy()
X_test = X_test.drop(columns = ['MovieID', 'UserID'])
user_item_test=pd.concat([X_test1,y_test],axis=1)

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

user_item_test.columns = ['UserId', 'ItemId', 'Rating']  # Lib requires specific column names
user_item_test.head(2)
```

{"type":"dataframe","variable_name":"user_item_test"}

```
model = CMF(method="als", k=4, lambda_=0.1, user_bias=False,
item_bias=False, verbose=False,produce_dicts=True)
model.fit(user_item_test)
```

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

```
y_pred_mf = np.dot(model.A_, model.B_.T) + model.glob_mean_
```

```
df = pd.DataFrame(y_pred_mf,columns
=list(model.item_mapping_),index=list(model.user_mapping_))
df.head()
```

{"type":"dataframe","variable_name":"df"}

```
del svd
del model
del predictions_svd
del testset
import gc
gc.collect()
```

```
96
```

```
df1=df.unstack().reset_index()
df1.rename(columns={'level_0': 'ItemId', 'level_1':
'UserId',0:'Rating'}, inplace=True)
df1.tail()
```

{"summary":"{\n \"name\": \"df1\",\n \"rows\": 5,\n \"fields\": [\n
{\n   \"column\": \"ItemId\",\n   \"properties\": {\n
\"dtype\": \"category\",\n   \"num_unique_values\": 1,\n
\"samples\": [\n   \"1669\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"UserId\",\n   \"properties\":
{\n   \"dtype\": \"string\",\n   \"num_unique_values\": 5,\n
\"samples\": [\n   \"3353\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Rating\",\n   \"properties\":
{\n   \"dtype\": \"float32\",\n   \"num_unique_values\": 5,\
n   \"samples\": [\n   3.558867931365967\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe"}

```
df_mf = pd.merge(user_item_test, df1, on=['UserId','ItemId'],
how='inner')
df_mf.rename(columns={'Rating_x': 'True_rating', 'Rating_y':
'Mf_pred_ratings'}, inplace=True)
df_mf.head()
```

{"type":"dataframe","variable_name":"df_mf"}

```
df_gb=pd.DataFrame(y_pred_reg,columns =['reg_pred_ratings'])
df_reg= pd.concat([df_gb,df_mf['Mf_pred_ratings']],axis=1)
df_reg.head()
```

{"type":"dataframe","variable_name":"df_reg"}

```
y = df_mf['True_rating']
X_train, X_test, y_train, y_test = train_test_split(df_reg, y,
test_size=0.2, random_state=SEED)

from sklearn.linear_model import LinearRegression

model = LinearRegression().fit(X_train, y_train)

y_pred_en=model.predict(X_test)

rmse = np.sqrt(mean_squared_error(y_test, y_pred_en)) # calculating rmse value
print('Root Mean Squared Error: {:.3f}'.format(rmse))
```

```
Root Mean Squared Error: 0.757
```

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

```
Mean Absolute Percentage Error: 0.218
```

6. 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**

1. 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
```

```
#!sudo apt-get update
#!sudo apt-get install -y pandoc
#!sudo apt-get install -y texlive-xetex texlive-fonts-recommended
texlive-plain-generic
#!pip install --upgrade nbconvert
#!pip install nbconvert[webpdf]
#!playwright install chromium
#!jupyter nbconvert --to webpdf Zee_Recommender_System.ipynb
```